



Empowering Intelligence

HailoRT User Guide

Release 5.2.0
6 January 2026

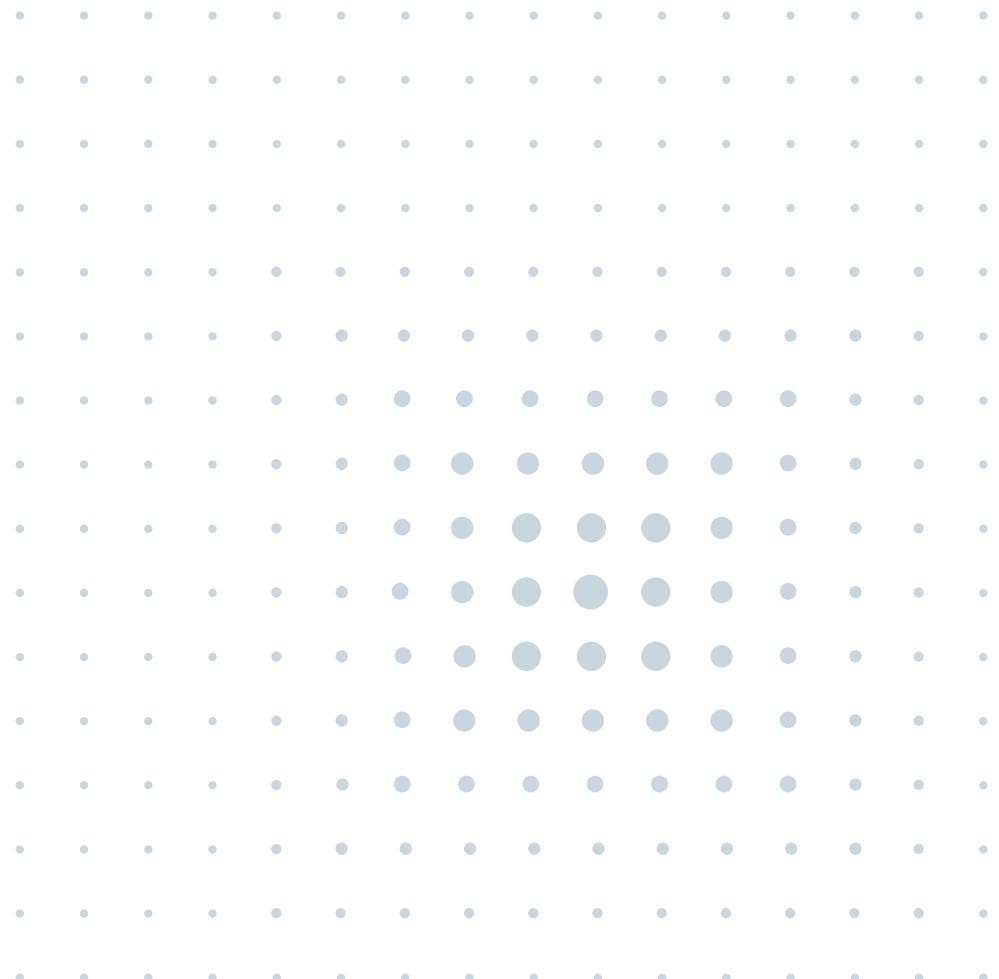


Table of Contents

I User Guide	2
1 HailoRT Overview	3
1.1 Included in this Package	3
1.2 HailoRT Library (C/C++ API)	4
1.3 HailoRT Python API	4
1.4 HailoRT GenAI	4
1.5 HailoRT CLI	4
1.6 PCIe Driver	4
1.7 Yocto Project Layer	4
1.8 Hailo Firmware	4
1.9 Hailo GStreamer Plugin	5
2 Changelog	6
3 Installation	14
3.1 System Requirements	14
3.2 Validating Successful PCIe Driver Installation on Linux	15
3.3 Installation on Ubuntu	15
3.4 Installation on Yocto-based Linux Distribution	19
3.5 Compiling from Sources	19
3.6 Linux Installation Troubleshooting	21
3.7 Installation on Windows	22
4 Installation of HailoRT Inside a Docker Container	24
4.1 Docker Installation	24
4.2 Running HailoRT Container from Pre-Built Docker Image	24
5 Tutorials	26
5.1 C Inference Tutorial	27
5.2 C++ Inference Tutorial	41
5.3 GenAI Inference Tutorial	49
5.4 Python Async Inference Tutorial - Single Model	62
5.5 Python Async Inference Tutorial - Multiple Models	63
5.6 Python Power Measurement Tutorial	65
5.7 Python LLM (Large Language Model) Tutorial	67
5.8 Function Calling / Tool Use Example	69
5.9 Python VLM (Vision Language Model) Tutorial	74
5.10 Python Speech2Text Tutorial	84
6 Running Inference	88
6.1 Inference Stages	88
6.2 Data Formats	90
6.3 Environment Variables	94
6.4 Model Scheduler and Compiling Models Together	94
6.5 Model Scheduler	95
6.6 Model Scheduler Optimizations	96
6.7 Stream Multiplexer	97
6.8 Multi-Process Service	98
6.9 VStream API	99
7 Generative-AI	100
7.1 LLM / VLM C++ API	100

7.2 Speech-to-Text C++ API	101
8 Command Line Tools	102
8.1 Scan Tool	102
8.2 Parse-HEF Tool	102
8.3 Inference	102
8.4 Benchmarking Tool	104
8.5 Monitor	104
8.6 HailoRT Profiler	105
8.7 Firmware Tools	106
9 PCIe Driver	108
9.1 Manual Setup	108
9.2 Parameters	109
10 SoC Features	110
10.1 Temperature Monitoring	110
10.2 User Configuration	111
10.3 Power Modes	113
11 Yocto	114
11.1 The Meta-Hailo Layers	114
11.2 Recipes in Meta-Hailo-Libhailort	114
11.3 Recipes in Meta-Hailo-Accelerator	115
11.4 Recipes in Meta-Hailo-Vpu	115
11.5 Integrating with an Existing Yocto Environment	116
11.6 Validating the Integration's Success	116
11.7 Offline Builds	117
12 Integration with Frameworks	120
12.1 GStreamer	120
13 Development Guidelines	121
13.1 Common Errors	121
13.2 Tips and Best Practices	122
II API Reference	124
14 HailoRT C API Reference	125
14.1 Device and Control API Functions	125
14.2 VDevice API Functions	132
14.3 HEF API Functions	133
14.4 Network Group API Functions	137
14.5 Virtual Stream API Functions	142
14.6 Stream API Functions	149
14.7 Transformation API Functions	154
14.8 Power Measurement API Functions	159
14.9 Multiple Networks API Functions	160
14.10 Other API Definitions	161
15 HailoRT C++ API Reference	211
15.1 Device and Control API Functions	211
15.2 VDevice API Functions	223
15.3 HEF API Definitions and Functions	227
15.4 Network group API Definitions and Functions	234
15.5 Stream API Functions	239
15.6 Transformation API Functions	248
15.7 InferModel API	257
15.8 Virtual Stream API Functions	266
15.9 DMA Buffer Mapping API Functions	274

15.10 Common HailoRT Utility Functions	275
15.11 Runtime Statistics	279
15.12 Error Handling	281
15.13 Type Definitions	282
16 HailoRT GenAI C++ API Reference	283
16.1 LLM API	283
16.2 VLM API	292
16.3 Speech2Text API	299
17 HailoRT Python API Reference	304
17.1 <code>hailo_platform.pyhailort.pyhailort</code>	304
18 HailoRT Python GenAI API Reference	338
18.1 <code>hailo_platform.genai</code>	338
Python Module Index	356

Disclaimer and Proprietary Information Notice

Copyright

© 2026 Hailo Technologies Ltd ("Hailo"). All Rights Reserved.

No part of this document may be reproduced or transmitted in any form without the expressed, written permission of Hailo. Nothing contained in this document should be construed as granting any license or right to use proprietary information for that matter, without the written permission of Hailo.

This version of the document supersedes all previous versions.

General Notice

Hailo, to the fullest extent permitted by law, provides this document "as-is" and disclaims all warranties, either express or implied, statutory or otherwise, including but not limited to the implied warranties of merchantability, non-infringement of third parties' rights, and fitness for particular purpose.

Although Hailo used reasonable efforts to ensure the accuracy of the content of this document, it is possible that this document may contain technical inaccuracies or other errors. Hailo assumes no liability for any error in this document, and for damages, whether direct, indirect, incidental, consequential or otherwise, that may result from such error, including, but not limited to loss of data or profits.

The content in this document is subject to change without prior notice and Hailo reserves the right to make changes to content of this document without providing a notification to its users.

Part I

User Guide

1. HailoRT Overview

Hailo SW products are a set of frameworks and tools that enable you to compile, run and evaluate neural networks on Hailo devices. HailoRT is part of Hailo's **runtime environment**:

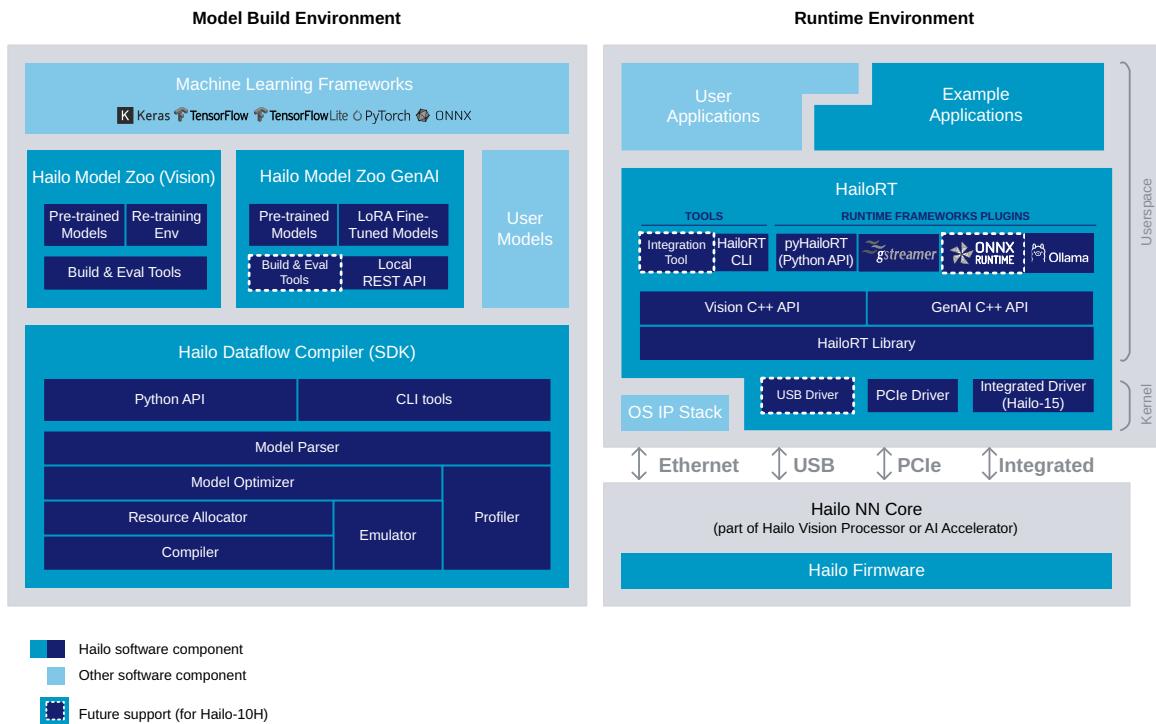


Figure 1. Detailed block diagram of Hailo software packages

Note: HailoRT supports all Hailo devices.

1.1. Included in this Package

This software package includes the following parts:

- HailoRT library to run inference from C/C++ programs
- HailoRT Python package
- HailoRT GenAI
- HailoRT CLI
- PCIe driver
- Yocto layer
- Hailo Firmware
- Hailo GStreamer Plugin
- Additional files such as an installation script and a configuration file

1.2. HailoRT Library (C/C++ API)

The HailoRT library is a user-space run time library responsible for operating the Hailo device. It allows both to control the Hailo device and to transfer data to and from it. The HailoRT library implements robust and intuitive API's in C/C++ for optimized performance using the Hailo device.

1.3. HailoRT Python API

The HailoRT Python package wraps the library and exposes a Python interface that allows the loading of models to the device and to send and receive data from it.

1.4. HailoRT GenAI

HailoRT GenAI is a dedicated component within HailoRT that enables efficient execution of generative AI models on Hailo devices. Unlike traditional inference workflows focused on static image processing, generative AI models introduce unique challenges due to their size, dynamic structure, and context-driven behavior. HailoRT GenAI addresses these needs by offering a robust runtime framework that offloads generative inference to the Hailo device.

1.5. HailoRT CLI

HailoRT CLI is the command line application used to control Hailo devices, run inference, and collect inference statistics and device events.

1.6. PCIe Driver

The Hailo PCIe driver is required when working via the PCIe interface; it links the HailoRT library and the device. It also loads the device's firmware when working through this interface. The device driver is used to manage the Hailo device, communicate with the device, and transfer data to and from the device.

1.7. Yocto Project Layer

The Yocto layer allows the user to integrate Hailo's software into an existing Yocto environment. It includes recipes for the HailoRT library, Python package and the PCIe driver.

1.8. Hailo Firmware

The firmware that runs on the Hailo device. It manages the boot process and the control of the Hailo device.

1.9. Hailo GStreamer Plugin

The Hailo GStreamer plugin provides the `hailonet` element which can infer GStreamer frames according to the configured network. This element can be used multiple times in a GStreamer pipeline to infer multiple networks in parallel.

2. Changelog

HailoRT v5.2.0 (January 2026)

General

- Performance improvements of 5-10% for vision models.
- Added support for Windows OS in HailoRT for Hailo-10H devices.
 - Supported on Windows 10 and Windows 11 (64-bit).
 - Supported on PCIe interface only.
 - See the *HailoRT Windows Installation Guide* for more information.

GenAI

- Added support for language detection and repetition penalty for Speech2Text API (Whisper models)
 - Automatically detect the language of the audio input if not specified.
 - Control over the generation process to prevent the repetition of tokens with the new parameter `repetition_penalty` in `hailort::genai::Speech2TextGeneratorParams`
 - See *Speech2Text tutorial* for examples
- Added support for tools in LLM API
 - `hailort::genai::LLM::generate()` and `hailo_platform.pyhailort.pyhailort.LLM.generate()` overloads for direct generation with JSON structured prompts and tools without explicitly creating generators
 - `hailort::genai::LLMGenerator::write()` overloads for writing prompts and tools to the generator
 - See *LLM tutorial* for examples
- Added video support for VLM API
 - Added support for `input_videos` parameter for video input in `hailort::genai::VLMGenerator::generate()` and `hailort::genai::VLM::generate()`.
 - Videos are represented as a vector of raw frames, with each video being a `std::vector<MemoryView>`
 - Structured prompts support `{"type": "video"}` content type for video placeholders
 - See *VLM tutorial* for examples

PyHailoRT

- Added API for Speech2Text language detection and repetition penalty - see *Speech2Text tutorial*
- Added video support for VLM API
 - Added support for `videos` parameter for video input in `hailo_platform.pyhailort.pyhailort.VLM.generate()` and `hailo_platform.pyhailort.pyhailort.VLM.generate_all()`
 - Videos are represented as a list of lists of numpy arrays, with each video being a list of frames
 - Structured prompts support `{"type": "video"}` content type for video placeholders
 - See *VLM tutorial* for examples

InferModel Examples

- Added *Multi Process Example* which demonstrates how to run multiprocess inference on Hailo-10H devices, using the HailoRT C++ API.

API

- Deprecated the `hailo_detection_with_byte_mask_t::mask` field in `hailo_detection_with_byte_mask_t`. Use `hailo_detection_with_byte_mask_t::mask_offset` instead.

HailoRT v5.1.1 (November 2025)

GenAI

- Removed Text2Image API
- Added API for saving and loading the context content of the LLM and VLM models
 - **LLM:** Added `hailort::genai::LLM::save_context()`, `hailort::genai::LLM::load_context()`
 - **VLM:** Added `hailort::genai::VLM::save_context()`, `hailort::genai::VLM::load_context()`
- Added API for getting the current context usage of the LLM and VLM models, and for getting the maximum context capacity of the LLM and VLM models
 - **LLM:** Added `hailort::genai::LLM::get_context_usage_size()`, `hailort::genai::LLM::max_context_capacity()`
 - **VLM:** Added `hailort::genai::VLM::get_context_usage_size()`, `hailort::genai::VLM::max_context_capacity()`

PyHailoRT

- Fixed Python 3.13 support for PyHailoRT (By moving the Python 3.13 whl to Numpy 2.x)
- Added API for Speech2Text - see [Python GenAI API Reference](#) and [Speech2Text tutorial](#)
- Added API for saving and loading the context content of the LLM and VLM models
 - **LLM:** Added `hailo_platform.pyhailort.pyhailort.LLM.save_context()`, `hailo_platform.pyhailort.pyhailort.LLM.load_context()`
 - **VLM:** Added `hailo_platform.pyhailort.pyhailort.VLM.save_context()`, `hailo_platform.pyhailort.pyhailort.VLM.load_context()`
- Added API for getting the current context usage of the LLM and VLM models, and for getting the maximum context capacity of the LLM and VLM models
 - **LLM:** Added `hailo_platform.pyhailort.pyhailort.LLM.get_context_usage_size()`, `hailo_platform.pyhailort.pyhailort.LLM.max_context_capacity()`
 - **VLM:** Added `hailo_platform.pyhailort.pyhailort.VLM.get_context_usage_size()`, `hailo_platform.pyhailort.pyhailort.VLM.max_context_capacity()`
- See [Python GenAI API Reference](#) and [LLM tutorial](#) and [VLM tutorial](#) for context management examples

API

- Added restriction on using format order `hailo_format_order_t::HAILO_FORMAT_ORDER_HAILO_NMS_WITH_BYTE_MASK` for Hailo-10 devices.

HailoRT v5.1.0 (October 2025)**General**

- Added support for Ubuntu 24.04.
- Added support for Python-3.12 and Python-3.13.
- Removed support for Ubuntu 20.04.
- Removed support for Python-3.8 and Python-3.9.
- Removed the 'hailo' command line tool, use 'hailortcli' instead.

GenAI

- Added support for Speech-to-Text API
 - New introduced classes:
 - * `hailort::genai::Speech2Text`
 - * `hailort::genai::Speech2TextParams`
 - * `hailort::genai::Speech2TextGeneratorParams`
 - * `hailort::genai::Speech2Text::SegmentInfo`
 - For the full example, please refer to [HailoRT C++ speech-to-text tutorial](#).
- Added a warning message when the GenAI-model's conversation context is full. In such case it is recommended to clear the context.
- Added memory optimization support for LLM and VLM APIs to enable running large models on small-memory devices
 - New parameter `optimize_memory_on_device` in `hailort::genai::LLMParams` and `hailort::genai::VLMParams`
 - Enables client-side tokenization to reduce device memory usage
 - Requires libhailort to be compiled with `HAILO_BUILD_CLIENT_TOKENIZER=ON`
 - See [Memory Optimization and Compilation Requirements](#) for detailed documentation
- Added support for aborting an ongoing generation for LLM, VLM and Text2Image APIs
 - **LLM / VLM:** Added `hailort::genai::LLMGeneratorCompletion::abort()`
 - * Aborts the current generation by injecting the model's generation recovery sequence
 - * Flush any tokens that were generated but not yet read
 - * Set the generation status to `hailort::genai::LLMGeneratorCompletion::Status::ABORTED`
 - * Allow a new generation to start once this function returns
 - **Text2Image:** Added `hailort::genai::Text2ImageGenerator::abort()`
 - * If a generation is in progress, it will be aborted, and the corresponding generate call will return `hailo_status::HAILO_OPERATION_ABORTED`
 - * If no generation is in progress, this function has no effect and returns `hailo_status::HAILO_SUCCESS`.
 - * Allow a new generation to start once this function returns

PyHailoRT

- Added API for LLM - see [Python GenAI API Reference](#) and [LLM tutorial](#)
- Added API for VLM - see [Python GenAI API Reference](#) and [VLM tutorial](#)

- Added API for Text2Image - see [Python GenAI API Reference](#) and Text2Image tutorial

API

- Deprecate the `hailort::Hef::get_hef_device_arch()` function. Use `hailort::Hef::get_compatible_device_archs()` instead (same HEF can be compiled for multiple device architectures).
- `ddr_noc_total_transactions` field in `hailo_performance_stats_t` is not supported.

Scheduler

- Scheduler performance improvements.

HailoRT v5.0.1 (August 2025)

General

- libhailort and its components (hailortcli, hailort_server, examples, etc.) changed to use C++17.

API

- Users can now specify an IP address in the `hailo_vdevice_params_t::device_id` field to connect to devices over Ethernet.

Drivers

- Renamed Linux PCIe driver: `hailo_pci` -> `hailo1x_pci`.

GenAI

- Fixed a bug in VLM, by replacing the `vision_pad` token with `image_pad`. [VLM \(Vision-Language Model\) - vlm_example](#) was updated to use the new token.
- Added support for generation-recovery sequence
 - `hailort::genai::LLM::get_generation_recovery_sequence()`,
`hailort::genai::VLM::get_generation_recovery_sequence()`
 - `hailort::genai::LLM::set_generation_recovery_sequence()`,
`hailort::genai::VLM::set_generation_recovery_sequence()`
- Added support for setting costume stop tokens
 - `hailort::genai::LLM::get_stop_tokens()`, `hailort::genai::VLM::get_stop_tokens()`
 - `hailort::genai::LLM::set_stop_tokens()`, `hailort::genai::VLM::set_stop_tokens()`
- Added support for LLM and VLM prompt template and structured prompt
 - `hailort::genai::LLM::prompt_template()`
 - `hailort::genai::VLM::prompt_template()`
 - Added `hailort::genai::VLMGenerator::generate()` overload for JSON-based structured messages
 - Supports multi-modal content with automatic image count validation
 - reffer to [HailoRT C++ chat tutorial](#) and [VLM tutorial](#) for examples
- Added direct generation methods for enhanced convenience and simplified workflows
 - **LLM:** Added `hailort::genai::LLM::generate()` overloads for direct generation with JSON structured prompts without explicitly creating generators
 - **VLM:** Added `hailort::genai::VLM::generate()` overloads for direct generation with JSON structured messages and input frames without explicitly creating generators

- **Text2Image:** Added `hailort::genai::Text2Image::generate()` overloads for direct generation without explicitly creating generators
- These methods are convenience wrappers that automatically create generators internally, making the API more user-friendly for simple use cases
- All direct generation methods support both custom parameters and default parameters

InferModel Examples

- Added [Async Inference DMA Buffer Example](#) which demonstrates how to run inference using DMA buffer.

HailoRT v5.0.0 (July 2025)

General

- Starting July 2025, support for Hailo-8, Hailo-8R, and Hailo-8L devices will be available only in HailoRT version 4.x. HailoRT version 5.x will continue to support only the Hailo-10 and Hailo-15 device families. Before upgrading to HailoRT version 5.x, please ensure HailoRT driver version 4.x is [Uninstalled](#) from your machine.
- HailoRT now supports [GenAI](#), including LLMs (large language models), Text-to-image (stable diffusion) and VLMs (vision language models) exclusively on Hailo-10H.
- HailoRT now supports Hailo-10H and Hailo-15L.
- Known limitations in Hailo-10H (expected to be solved in future HailoRT releases):
 - PyHailoRT in Hailo-10H is not supported for generative AI models.
 - C++ vStream API is not supported for Hailo-10H.
 - Hailo-10H is supported only on Linux.
 - [Multi-Process Service](#) is not supported for Hailo-10H.

API

- Changes the logic of the `hailo_set_scheduler_timeout` function to improve the scheduling mechanism. The time period is measured from the time the first frame enters the queue. It resets when a switch occurs and there are still frames in the queue.
- Added the function `hailort::Device::fetch_logs()` and `hailort::Device::get_max_logs_size()` to retrieve logs from the Hailo device. Also, added a new enum `hailo_log_type_t` to be used in those APIs.

Note: Reading a log clears its content, so subsequent reads will return only new log entries added after the last read.

- Added a new struct `hailo_tensor_metadata_t` to be used with GStreamer output tensor instead of `hailo_vstream_info_t`.

CLI

- Changed default run mode of `hailortcli run2` to `full_async`.
- Removed the option to pass asterisk (*) on the `--device-id` and `--bdf` options of commands that support them.
- Added the command `hailortcli logs` which dump log to console, with the selection of `hailo_log_type_t` and the option to `--follow`.

InferModel Examples

- Added [Multi Model Inference Example](#) which demonstrates how to run inference on multiple models using the same VDevice.

PyHailoRT

- Renamed HRT_4_Power_Measurement_Tutorial.ipynb to HRT_2_Power_Measurement_Tutorial.ipynb.
- scheduling_algorithm default value is changed to HailoSchedulingAlgorithm.ROUND_ROBIN, meaning the scheduler is on by default.

HailoRT v4.21.0 (April 2025)

API

- `hailo_format_order_t::HAILO_FORMAT_ORDER_HAILO_NMS` for format order was removed. Instead, use `hailo_format_order_t::HAILO_FORMAT_ORDER_HAILO_NMS_BY_CLASS`.
- Added the function `hailort::VDevice::create_shared()`
- It is now possible to configure an `hailort::InferModel` more than once. may be useful for creating multiple pipelines with the same model.
- Added the option to create `hailort::InferModel` from `std::shared_ptr` to a `hailort::Buffer`, pointing to the `hailort::Hef` data - `hailort::VDevice::create_infer_model()`.
- Added the option to create `hailort::Hef` from `std::shared_ptr` to a `hailort::Buffer` - `hailort::Hef::create()`.
- Added `query_performance_stats()` and `query_health_stats()` APIs to the `hailort::Device` class.

Note: Example which demonstrates how to use the APIs was added: [Query performance and health stats example](#)

- Added `gpio_mask` to `hailo_extended_device_information_t`.

PyHailoRT

- When using `Multi-Process Service` with `VDevice` being shared between processes, the user must set the environment variables `GRPC_ENABLE_FORK_SUPPORT=1` and `GRPC_POLL_STRATEGY=poll` before forking.

HailoRT v4.20.1 (January 2025)

Hailo-15

- Fixed a performance degradation when using DMA-BUF as raw pointers, which affected all usages of HailoNet in Hailo-15

HailoNet

- Fixed a bug which displayed warnings when HailoNet was run with the flag pass-through enabled

HailoRT v4.20.0 (January 2025)**LibHailoRT**

- Rename some ::hailo_status values
 - HAILO_DRIVER_FAIL -> `hailo_status::HAILO_DRIVER_OPERATION_FAILED`
 - HAILO_PCIE_DRIVER_NOT_INSTALLED -> `hailo_status::HAILO_DRIVER_NOT_INSTALLED`
- Device creation documentation updated to only support long format BDF (DDDD:BB:DD.F)

API

- HAILO_FORMAT_ORDER_HAIGO_NMS format order is deprecated. Instead, use `hailo_format_order_t::HAILO_FORMAT_ORDER_HAIGO_NMS_BY_CLASS`

Note: For networks which are currently using NMS, the default format order is `hailo_format_order_t::HAILO_FORMAT_ORDER_HAIGO_NMS_BY_CLASS`

- Added `hailo_format_order_t::HAILO_FORMAT_ORDER_HAIGO_NMS_BY_SCORE` format - which orders all detections together by their score (preview)

Note: `hailo_format_order_t::HAILO_FORMAT_ORDER_HAIGO_NMS_BY_SCORE` is not supported in Sync infer, and is also not supported in pyhailort

Async API

- Improved stability of Asynchronous Inference API

CLI

- The --no-power option was removed from `hailortcli benchmark`
- Fixed a bug in temperature measurement when using `hailortcli run2`

PyHailoRT

- Fixed a bug in models with the format order `HAILO_NMS_WITH_BYTE_MASK`
- Fixed a compilation bug when using multi config generators.
- Edited `Python Sync Inference Tutorial for Multiple Models </tutorials_notebooks/notebooks/HRT_3_Infer_Pipeline_Inference_Multiple_Models_Tutorial.ipynb>` to demonstrate `batch_size` setting, and utilize VDevice and `Multi-Process service`
- Removed deprecated `Device/Control.configure` from PyHailoRT (One can use `configure()` or `configure()` instead)

Multi-Process Service

- Fixed a bug in Windows permissions on user application side when working with `Multi-Process service`

HailoNet

- The `vdevice-key` property (which was previously deprecated) is now removed from `hailonet`. use `vdevice-group-id` instead

Note: For Windows based OS: In order for the `hailonet` to be detected as part of gstreamer plugins, both `gsthalo.lib` and `gsthalo.dll` must be manually copied from `C:\Program Files\HailoRT\HailoNet` to `C:\gstreamer\1.0\msvc_x86_64\lib\gstreamer-1.0`

Firmware

- Allows using Hailo-8 in virtualized environments (preview)

HailoRT PCIe Driver for Linux

- Add support for linux kernel 6.12
- Always reload the firmware on driver load
- Pass `O_CLOEXEC` flag to `open()` to avoid file descriptor inheritance (which may cause `hailo_status::HAILO_DEVICE_IN_USE` error)
- Allows working with DMA-BUF, even if the buffer was passed as raw pointer from user space

3. Installation

This chapter presents the system requirements and installation instructions per operating system/distribution.

With *system requirements* that comply with the description below, Hailo provides several possible ways to install HailoRT, depending on the operating system:

- Installing on *Ubuntu*
- Installing on a *Yocto-based distribution*
- Installing on *other Linux distributions* by compiling from source code
- Installing on *Windows*

3.1. System Requirements

For the installation of HailoRT, the following minimum system requirements are necessary:

- Linux or Windows
- 2+ GB RAM (4+ GB RAM recommended)
- Physical connection, depending on the board:
 - PCIe interface
 - M.2 connector for the M.2 board

Table 1. Hailo-10H M.2 board



- Python 3.10, 3.11, 3.12 or 3.13 (optional)

Note: Additional requirements may be needed depending on the installation method, see below.

Note: The PCIe driver has been tested on the following Linux kernel versions 5.13.0, 5.15.0, 5.18.0, 5.19.5, 6.2.0, 6.8.0, and 6.12.18.

Note: The RAM requirement depends on the number of the network's inputs and outputs, and on the data throughput.

3.2. Validating Successful PCIe Driver Installation on Linux

Make sure your device is connected to a PCIe/M.2/mPCIe slot. Run the following command:

```
lspci | grep Co-processor
```

The expected output is one of the following:

```
3c:00.0 Co-processor: Hailo Technologies Ltd. Hailo-10H AI Processor (rev 01)
```

or

```
3c:00.0 Co-processor: Device 1e60:45C4 (rev 01)
```

If the second output is received, update the PCIe IDs file on your device by running:

```
sudo update-pciids
```

If the Hailo device does not appear in the result, please ensure the following:

- The device is connected properly to the slot.
- The PCIe driver installation did not output any error messages.
- The machine was rebooted after installing the PCIe driver.
- The PCIe slot is working properly.

3.3. Installation on Ubuntu

There are several ways to install HailoRT on Ubuntu:

- As part of the **Hailo Software Suite** – Please refer to the suite installation instructions inside the Hailo Dataflow Compiler User Guide.
- Using a **Docker container** (without the full suite) – Please refer to the [Docker instructions](#).
- Using the **Ubuntu installer**.

Note: Supported versions - Ubuntu 22.04/24.04, x86_64/aarch64.

The rest of this section focuses on the Ubuntu installer.

3.3.1. Ubuntu Installer Requirements

Note: These software requirements are additionally required *system requirements*.

- build-essential package (needed for compiling the PCIe driver)
- (Optional) bison, flex, libelf-dev and dkms packages (needed to register the PCIe driver using DKMS)
- (Optional) curl (needed to download the firmware when installing without the PCIe driver)
- (Optional) cmake (needed for compiling the HailoRT examples)
- (Optional) pip and virtualenv (needed for pyhailort)

3.3.2. Installing HailoRT on Ubuntu

The HailoRT Ubuntu offers several files, select the files according to your requirements:

1. **HailoRT PCIe driver** and FW-hailort-pcie-driver_<version>.all.deb
2. **HailoRT** for the platform architecture - one of the following:
 - hailort_<version>.amd64.deb - HailoRT for amd64
 - hailort_<version>.arm64.deb - HailoRT for arm64
 - hailort_<version>.armel.deb - HailoRT for armel
 - hailort_<version>.armv4.deb - HailoRT for armv4
3. **PyHailoRT** for the the platform architecture and installed Python version - one of the following:
 - hailort-<version>-cp310-cp310-linux_x86_64.whl - PyHailoRT for python3.10, x86_64
 - hailort-<version>-cp311-cp311-linux_x86_64.whl - PyHailoRT for python3.11, x86_64
 - hailort-<version>-cp312-cp312-linux_x86_64.whl - PyHailoRT for python3.12, x86_64
 - hailort-<version>-cp313-cp313-linux_x86_64.whl - PyHailoRT for python3.13, x86_64
 - hailort-<version>-cp310-cp310-linux_aarch64.whl - PyHailoRT for python3.10, aarch64
 - hailort-<version>-cp311-cp311-linux_aarch64.whl - PyHailoRT for python3.11, aarch64
 - hailort-<version>-cp312-cp312-linux_aarch64.whl - PyHailoRT for python3.12, aarch64
 - hailort-<version>-cp313-cp313-linux_aarch64.whl - PyHailoRT for python3.13, aarch64

Note: PyHailoRT is optional and is needed for using the Python API

Note: PyHailoRT supports numpy 1.x for python3.12 and below, and numpy 2.x for python3.13 and above.

Download

Download the relevant files for your environment from our [Developer Zone](#).

To install HailoRT `hailort_<version>_<arch>.deb` is required. Information about machine's architecture can be found using:

```
# .deb architecture  
dpkg --print-architecture
```

To install the PCIe driver `hailort-pcie-driver_<version>_all.deb` is required.

Optional - To install PyHailoRT, you'll also need: `hailort-<version>-<python-tag>-<abi-tag>-<platform-tag>.whl`.

You can find your python version and architecture using:

```
# Python version major and minor digits  
python -V | cut -d. -f1,2 | tr -dc '0-9.' | xargs  
# .whl architecture  
uname -m
```

Installation of the PCIe Driver Only

Run the following command to install only the PCIe driver:

```
sudo dpkg --install hailort-pcie-driver_<version>_all.deb
```

Note: A prompt regarding using DKMS (Dynamic Kernel Module Support) is expected and it is recommended to approve it.

Note: PC restart is required after driver installation.

Installation with the PCIe Driver

Run the following command to install HailoRT including the PCIe driver:

```
sudo dpkg --install hailort_<version>_${(dpkg --print-architecture)}.deb hailort-  
→pcie-driver_<version>_all.deb
```

This command will install HailoRT. It will also compile the PCIe driver for the machine's kernel version and install the driver. The command needs root permissions (sudo).

Note: PC restart is required after driver installation.

After boot, you can use the `hailortcli tool` and run `hailortcli scan` to validate that the device is identified:

```
hailortcli scan
```

The scan command should find the device.

Note: The PCIe driver is not signed. In some systems it means that secure boot has to be disabled to load the driver. Users who wish to use secure boot while HailoRT PCIe driver is not signed, can use MOK (Machine-Owner Key), which

can be used for (locally) signing third-party drivers. Please notice this is an advanced feature and is recommended for users who are familiar with the process of locally-signing drivers.

Identifying Device's Serial Number

Run the following command to obtain the serial number from the device:

```
hailortcli fw-control identify
```

Installation of pyHailoRT into a New Environment

Run the following command to install PyHailoRT into a new virtual environment:

```
virtualenv -p python<python_version> hailo_platform_venv && . hailo_platform_venv/
  ↪bin/activate && pip install ./hailort-<version>-<python_tag>-<abi_tag>-<platform_
  ↪tag>.whl
```

For example, for installing PyHailoRT **v4.17.0** to a **Python-3.10** environment on a **x86_64 platform**, the command would be:

```
virtualenv -p python3.10 hailo_platform_venv && . hailo_platform_venv/bin/activate &
  ↪& pip install ./hailort-4.17.0-cp310-cp310-linux_x86_64.whl
```

Installation of pyHailoRT into an Existing Environment

Run the following command to install PyHailoRT into existing virtual environment:

```
source <virtualenv>/bin/activate && pip install ./hailort-<version>-<python_tag>-
  ↪<abi_tag>-<platform_tag>.whl
```

3.3.3. Uninstalling HailoRT

- Remove the PCIe driver:

```
sudo dpkg --purge hailort-pcie-driver
```

- Remove HailoRT (excluding pyhailort):

```
sudo dpkg --purge hailort
```

- Remove pyhailort (if installed):

```
source hailo_platform_venv/bin/activate  
pip uninstall hailort
```

3.4. Installation on Yocto-based Linux Distribution

See the [Yocto](#) page for details about Yocto integration.

3.5. Compiling from Sources

3.5.1. Compiling Hailo PCIe Driver from Sources

See the [PCIe driver](#) page for details about compiling the PCIe driver from sources.

3.5.2. Compiling HailoRT from Sources

Using HailoRT with other Linux distributions is possible via source compilation. On Ubuntu, it is even sometimes useful to compile from sources, for example in order to keep ABI integrity. HailoRT sources can be cloned from GitHub using:

```
git clone https://github.com/hailo-ai/hailort.git
```

Compiling the sources is done with the following command:

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release && cmake --build build --config release
```

The compilation will create two artifacts:

- Binary called `hailortcli` located in `build/hailort/hailortcli/`
- Library called `libhailort.so.<version>` located in `build/hailort/libhailort/src/`

Note: By adding `--target install` to the CMake command, HailoRT artifacts will be installed on the machine.

Linux:

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release && sudo cmake --build build --config release --target install
```

After installation (either by installer or CMake install) one can link with libhailort by using CMake's `find_package()`. See '`hailort/libhailort/examples/CMakeLists.txt`' for reference.

Note: By adding `-DHAILO_BUILD_EXAMPLES=1` to the CMake command, examples targets will be added to the project (useful for debug build of C++ examples, instead of linking them with pre-installed HailoRT).

Note: For GenAI memory optimization support, add `-DHAILO_BUILD_CLIENT_TOKENIZER=ON` to the CMake command. This enables client-side tokenization to reduce device memory usage for large models. See [Memory Optimization and Compilation Requirements](#) for detailed documentation.

Note: When building from sources, some additional tools must be installed. For example, gcc for arm or python-dev if building pyhailort.

Run the command:

```
build/hailort/hailortcli/hailortcli
```

Note: To compile sources on Windows please see [Windows compile from sources](#)

3.5.3. Compiling Specific HailoRT Targets

Compiling a specific target is supported using CMake's API --target

Supported targets are: libhailort, hailortcli

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release && cmake --build build --config release -  
→--target libhailort
```

Some HailoRT targets require additional cmake flags to be built first.

Compilation of HailoRT-Python-Binding (i.e., `hailo_platform` wheel):

```
# Run from hailort/libhailort/bindings/python/platform dir  
python setup.py bdist_wheel
```

Note: The wheel depends on libhailort. Make sure libhailort is installed before building the (.whl) file.

Note: To change the build type (Release, Debug, etc.) of the pyhailort library, set the environment variable CMAKE_BUILD_TYPE. Default is Release.

Cross-Compilation of HailoRT-Python-Binding (i.e., `hailo_platform` wheel):

For cross-compilation, set the following environment variables to `setup.py bdist_wheel`:

- CMAKE_TOOLCHAIN_FILE - path to the toolchain file (defines the compiler and linker for cross-compilation)
- HAILORT_INCLUDE_DIR - path to the include directory of the HailoRT library
- LIBHAILORT_PATH - path to the HailoRT library
- PYTHON_EXECUTABLE - path to the python executable. See [<https://pybind11.readthedocs.io/en/stable/compiling.html#findpython-mode>](https://pybind11.readthedocs.io/en/stable/compiling.html#findpython-mode)
- PYBIND11_FINDPYTHON - see pybind documentation [<https://pybind11.readthedocs.io/en/stable/compiling.html#findpython-mode>](https://pybind11.readthedocs.io/en/stable/compiling.html#findpython-mode)

Note: Since the wheel architecture is set by default to the native architecture, the user is required to pass to `bdist_wheel` the argument --plat-name - which indicates the target architecture (e.g. `linux_aarch64`).

```
# Run from hailort/libhailort/bindings/python/platform dir  
LIBHAILORT_PATH=<LIB_PATH> HAILORT_INCLUDE_DIR=<INC_PATH> CMAKE_TOOLCHAIN_FILE=  
→<TOOLCHAIN_PATH> python setup.py bdist_wheel --plat-name=linux_aarch64
```

Compilation of hailort-gstreamer-binding:

```
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release -DHAILO_BUILD_GSTREAMER=1 && cmake --  
→build build --config release --target gsthailo
```

Note: This step requires GStreamer to be installed on your system. Refer to the official installation guide: <https://gstreamer.freedesktop.org/documentation/installing/index.html>

Note: On Windows, by default, the gstreamer files are expected in the following path: C:/gstreamer/1.0/msvc_x86_64. This path can be changed by setting the CMake variable GSTREAMER_ROOT_DIR.

Compilation of hailort-examples:

```
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release -DHAILO_BUILD_EXAMPLES=1 && cmake --  
→build build --config release --target hailort_examples
```

3.6. Linux Installation Troubleshooting

This section contains common possible issues that you may encounter after connecting Hailo-10H and installing the driver and library.

3.6.1. Common Errors

Improper PCIe device enumeration

How to verify? From the terminal, run

```
lspci | grep hailo
```

The device should be listed in the terminal output, see [driver validation](#)

Possible root cause Improper mechanical installation

Possible solution Verify the module is properly attached and secured into the M.2 slot

Possible root cause Slot is not functional

Possible solution Verify the slot in use is a valid M.2 slot. Check to see if the slot is disabled in the platform BIOS

Device driver is not properly installed

How to verify? From the terminal, run

```
lsmod | grep hailo1x_pci
```

The device should be listed in the terminal output

Possible root cause Driver not installed

Possible solution Re-install the driver, see [driver installation only](#)

Device firmware not loaded

How to verify? From the terminal, run

```
dmesg | grep hailo
```

The firmware load process and events appear there - and if either the load process has ended with success or failure

Possible root cause Firmware not loaded

Possible solution If the firmware load has failed (during boot) - the reason may be specified in the log. Re-install the driver, see [driver installation only](#)

Module not identified by HailoRT

How to verify? From the terminal, run

```
hailortcli scan
```

The module should be listed in the terminal output

Possible root cause HailoRT library not installed correctly

Possible solution Re-install the library, see [HailoRT installation](#)

3.7. Installation on Windows

3.7.1. Windows Requirements

Note: These software installations are additionally required: [system requirements](#).

- Windows 10/11 64-bit
- (Optional) CMake and Visual Studio Build Tools – in order to compile applications that use HailoRT

Note: Validated on Visual Studio Build Tools 2019.

3.7.2. Windows Installation Instructions

1. Connect the Hailo device to the host.
2. Download the installer file `hailort_<version>_windows_installer.msi` from Hailo's website.

Note: It is recommended to remove old HailoRT versions before installing a new version.

3. Run the installer and follow the instructions. Check the `pyHailoRT` box if you wish to install this feature.
4. Reboot the host.
5. After boot, you can use the `hailortcli tool` and run `hailortcli scan` to validate that the device is identified.
6. If `pyHailoRT` – preview was checked, the Python wheel will be found in the directory `C:\Program Files\HailoRT\python`. Refer to section [Installation of pyHailoRT into a new environment](#) to finish the installation of pyHailoRT.

Note: The .whl cannot be installed (via pip) from the Program Files folder.

Note: Gstreamer Support: In order for the hailonet to be detected as part of gstreamer plugins, both gsthailo.lib and gsthailo.dll must be manually copied from C:\Program Files\HailoRT\HailoNet to C:\gstreamer\1.0\msvc_x86_64\lib\gstreamer-1.0

3.7.3. Compiling HailoRT from Sources on Windows

Note: Compilation of HailoRT from sources is recommended, for example, in order to keep ABI integrity. See [clone HailoRT](#) and [compile HailoRT](#). Notice that when compiling the HailoRT library from sources, the library will not be signed.

To compile HailoRT on Windows, run:

```
cmake -S . -Bbuild -A=x64 -DCMAKE_BUILD_TYPE=Release && cmake --build build --config release --target install
```

Compilation Notes:

- This section refers only to HailoRT library itself and not the PCIe driver. The PCIe driver should be installed using the .msi installation file (can be downloaded from our [Developer Zone](#)).
- When compiling for Windows, add the define NOMINMAX to prevent collisions.

The compilation will create three artifacts:

- Binary called `hailortcli.exe` located in `build\hailort\hailortcli\Release\`
- Library called `libhailort.lib` located in `build\hailort\libhailort\src\Release\`
- Library(DLL) called `libhailort.dll` located in `build\hailort\libhailort\src\Release\`

Run the compilation with the command:

```
hailortcli.exe
```

4. Installation of HailoRT Inside a Docker Container

For easier installation, HailoRT can be installed via a Docker container. The HailoRT Docker image contains all of the HailoRT components, including libhailort, hailortcli, pyhailort, and the HailoRT GStreamer plugin. The HailoRT PCIe driver must be installed separately on the system, prior to using HailoRT inside Docker.

4.1. Docker Installation

To install Docker:

- Install curl

```
sudo apt-get install -y curl
```

- Install Docker

```
curl -fsSL https://get.docker.com -o get-docker.sh  
sh get-docker.sh
```

- Add your user (which has root privileges) to the Docker group

```
sudo usermod -aG docker $USER
```

- Reboot or log out in order to apply the changes to the group

4.2. Running HailoRT Container from Pre-Built Docker Image

Note: Requirements: Docker package, either docker.io 20.10.07 (from Ubuntu repository), or docker-ce 20.10.6 (from Docker website).

Using this method, the following intermediate steps are handled by the script:

- Installing the relevant Linux kernel headers inside the container.
- Running the container with required arguments.

Note: The directory from which you use the Docker should contain following Hailo files:

- `hailo_docker_hailort_VERSION.tar`
- `run_hailort_docker.sh`

4.2.1. Installing PCIe Driver on the Host

The following step is required if the PC does not have a working `hailo_pci` driver installed, or its version is different from the current one. Install the PCIe driver on the host. See [how to download the pcie driver](#) and [how to install PCIe driver](#)

Note: PC restart is required after driver installation. After PC restart, one should be able to resume to existing container or to create a new one and start working with Hailo device.

4.2.2. Running the Container

In order to use HailoRT release Docker image, one should run the following script:

```
./run_hailort_docker.sh <hailort_image_name> <hailort_container_name>
```

For example:

```
./run_hailort_docker.sh hailo_docker_hailort_4.0.tar hailort_01
```

4.2.3. Docker Flags

Useful Docker flags to utilize Hailo devices in an application docker:

```
-v /dev:/dev \
-v /lib/firmware:/lib/firmware \
-v /lib/udev/rules.d:/lib/udev/rules.d \
-v /lib/modules:/lib/modules \
--device=/dev/hailo0:/dev/hailo0 \
```

Note: When using several Hailo devices, define the additional devices accordingly. For example, for a 2nd device, add:

```
--device=/dev/hailo1:/dev/hailo1 \
```

5. Tutorials

The tutorials below go through the steps in C, C++ and Python to run inference on Hailo devices and their purpose is to demonstrate HailoRT API usage.

Note: These tutorials are part of the Hailo AI Software Suite.

Hailo offers an additional set of [Application Code Examples](#), which are more application-oriented.

To download the tutorials, clone the tutorials from GitHub:

```
git clone https://github.com/hailo-ai/hailort.git
```

Run the download script to download the HEFs used by the tutorials:

- Ubuntu

```
cd hailort/hailort/scripts/ && ./download_hefs.sh && cd -
```

- Windows:

```
Double-click hailort\hailort\scripts\download_hefs.cmd
```

Note: The C and C++ tutorials require HailoRT to be installed.

Note: If running only C and C++ tutorials, PyHailoRT installation can be skipped.

Note: Python tutorials require HailoRT and PyHailoRT to be installed, and the additional following packages:

1. jupyter
2. matplotlib

To install these packages, inside the virtual environment where PyHailoRT is installed, use:

```
pip install jupyter matplotlib
```

Note: The C++ and genai examples utilize exceptions with the function `hailort::Expected::expect()`, so they require compiling with exception support enabled. In order to change an example to work without exceptions, remove the `hailort::Expected::expect()` usage, and replace it with `hailort::Expected::has_value()`, `hailort::Expected::value()` and `hailort::Expected::release()`.

For example, to create a *VDevice* without exceptions:

```
auto vdevice_expected = hailort::VDevice::create();
if (!vdevice_expected.has_value()) {
    // Handle error
    std::cerr << "Failed to create VDevice: " << vdevice_expected.status() << std::endl;
    return;
}
auto vdevice = vdevice_expected.release();
```

5.1. C Inference Tutorial

In this section various examples of the HailoRT API are provided. The code mentioned below is included in the release under `hailort/libhailort/examples/c` in the GitHub repository. An installed HailoRT is a prerequisite for running the examples. See [documentation](#).

All function calls are based on the header provided in `hailort/include/hailo/hailort.h`. See the [API documentation](#) for more details.

Note: In order to compile the examples from their current directory write permissions are required. If this is not available, copy the examples directory to another location which does have required permissions.

Note: The C examples are not supported on Hailo-10H.

5.1.1. Compilation and Execution

Compiling examples is done using the following commands from the examples directory:

```
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release  
cmake --build build --config release
```

In order to compile a specific example, either add the example name as a target with a `c` prefix, or run the `cmake` command from the target example directory:

```
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release  
cmake --build build --config release --target c_vstreams_example
```

```
cd c/vstreams_example  
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release  
cmake --build build --config release
```

Note: When copying an example to a different directory, make sure to copy the `common` directory as well.

Running examples:

Run the examples using the following command, from the examples directory:

```
build/c/<example_name>/c_<example_name> [params...]
```

Cross-Compile Tutorials

Add the following options while configuring the examples using CMake:

- `DCMAKE_TOOLCHAIN_FILE=<toolchain file>`
- `DCMAKE_FIND_ROOT_PATH=<target dir>`
- `DCMAKE_FIND_ROOT_PATH_MODE_PACKAGE=ONLY`

Note: For more details, see:

- [Cross Compiling With CMake](#)
 - [find_package — CMake 3.26.1 Documentation](#)
-

5.1.2. Inference Using Virtual Streams – *vstreams_example*

See: [hailort/libhailort/examples/c/vstreams_example.c](#)

Summary

- Demonstrates basic inference of a shortcut network (i.e inputs are sent through the vdevice and right back out, without any changes made to the data).
- The program uses the Hailo virtual device with default values for initialization.
- The data is sent to the vdevice via input vstreams and received via output vstreams.
- The data is transformed before sent and after receiving in a different thread using the virtual stream pipeline.

Code Structure

Device initialization The `VDevice` is created without passing any parameters, which means `hailo_vdevice_params_t` is used with default values. Because it is enabled by default, the model scheduler is enabled (with its default scheduling scheme - Round Robin). For that reason, there is no need to activate/deactivate the network groups in the user application, since when the scheduler is enabled, the activation and deactivation is done automatically.

Used APIs: `hailo_create_vdevice()`

Configuring the vdevice from HEF The next step is to create a `hailo_hef`, and to use it to configure the vdevice for inference. Initiate `hailo_configure_params_t` with default values, configure the vdevice to get a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params_by_vdevice()` and `hailo_configure_vdevice()`.

Creating vstreams The first stage is to initialize vstream params (both input and output), then the second stage is to create the vstreams.

Used APIs: `hailo_make_input_vstream_params()`, `hailo_make_output_vstream_params()`, `hailo_create_input_vstreams()` and `hailo_create_output_vstreams()`

Inference In this example, `p_thread` is used for inference, with a sub-thread for each output vstream receiving data and a sub-thread for each input vstream sending data.

Write to device The write thread, will firstly initialize a buffer used for send. Then the thread will send all frames in a loop.

Used APIs: `hailo_get_input_vstream_frame_size()` and `hailo_vstream_write_raw_buffer()`.

Read from device The read function is analogical to the write. Firstly initialize a receive buffer, then receive all frames in a loop.

Used APIs: `hailo_get_output_vstream_frame_size()` and `hailo_vstream_read_raw_buffer()`.

5.1.3. Multiple Device Inference, Using vstreams - *multi_device_example*

See: `hailort/libhailort/examples/c/multi_device_example.c`

Note: This example is not supported on Hailo-15.

Summary

- Demonstrates inference of a single network group over multiple devices.
- The program scans for Hailo devices.
- Creates a virtual device from all Hailo devices that are connected to the computer.
- The program infers HEFs with multiple virtual input streams and multiple virtual output streams.
- The program creates a thread for each virtual output stream and a thread for each virtual input stream.

Code Structure

The code is similar to `hailort/libhailort/examples/c/vstreams_example.c`, so be sure to read the *vstreams_example* first.

Device initialization First, scan for Hailo devices and then initialize the `hailo_vdevice_params_t` with the device count scanned before. The next step is to create the `VDevice`.

Used APIs: `hailo_create_vdevice()`, `hailo_init_vdevice_params()`
`hailo_scan_devices()`.

Pre infer stage

Configuring the vdevice from HEF (for each HEF) The next step is to create a `hailo_hef`, and to use it to configure the vdevice for inference. Initiate a `hailo_configure_params_t` with default values. Then, modify batch-size to 1 for each network group. This is the default batch-size value and will not have any effect (just to demonstrate API usage). Configure the vdevice to get a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params_by_vdevice()` and `hailo_configure_vdevice()`.

Build vstreams The next step is to create the input and output virtual streams for the configured network. First, make the default virtual stream params for each virtual stream. Second, create the virtual streams (inputs and outputs). Finally, extract the vstream's frame size and allocate the host input and output buffers and initialize the source with random data to send to inference.

Used APIs: `hailo_make_input_vstream_params()`, `hailo_make_output_vstream_params()`,
`hailo_create_input_vstreams()`, `hailo_create_output_vstreams()`,
`hailo_get_input_vstream_frame_size()` and `hailo_get_output_vstream_frame_size()`.

Infer stage

Inference In this example, create a new thread for each input vstream using the function `create_input_vstream_thread()`. Each created thread sends data to the vdevice using the function `write_to_vdevice()`.

Write to device The write thread gets as input the following:

- *input_vstream* – The virtual input stream.
- *src_data* – The actual data sent to the HW.
- *src_frame_size* – Size of the data to be sent to the HW.

The write thread will send all frames by writing the HW buffer to the vdevice.

Used APIs: `hailo_vstream_write_raw_buffer()`.

A new thread for each output vstream is also created using the function `create_output_vstream_thread`. Each created thread receives data from the vdevice using the function `read_from_vdevice`.

Read from device The read function gets as input the following:

- `output_vstream` – The virtual output stream.
- `dst_data` – The actual data sent by the HW back to the host.
- `dst_frame_size` – Size of the data expected to be received by the host.

The read function will receive all frames in the following flow:

- Reading data from the device.
- No post processing is made on the received data.

Used APIs: `hailo_vstream_read_raw_buffer()`.

In order to complete the inference on each network group, the main thread waits for all the input vstream threads to finish via `hailo_join_thread`, and afterwards waits for all the output vstreams threads to finish before deactivating the current activated network group and continuing onto the next.

5.1.4. Multi Network Inference Using Virtual Streams - `multi_network_vstream_example`

See: `hailort/libhailort/examples/c/multi_network_vstream_example.c`

Summary

- Demonstrates how to work with multiple networks compiled together into the same network group (co-compilation), using virtual streams.
- Working with multiple networks allows you to configure network-based parameters (such as different batch per network) and gives a more native approach to work with send/receive threads per network.
- The example works with an HEF containing one network group, and two networks in the network group.
- The example uses the first Hailo device found.
- Configure the network group and get the networks information to create the vstreams for each network.
- The data is sent to the device via input vstreams and received via output vstreams.
- The data is transformed before sent and after receiving in a different thread using the virtual stream pipeline.

Code Structure

Device initialization The first step is to initialize the vdevice, initialize the `hailo_vdevice_params_t` with default values, and then create the `VDevice`.

Used APIs: `hailo_create_vdevice()`, `hailo_init_vdevice_params()`

Configuring the vdevice from HEF The next step is to create a `hailo_hef`, and to use it to configure the vdevice for inference. Initialize a `hailo_configure_params_t` with default values, and change the batch size for each network in the `network_group`. Then, configure the vdevice and get a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params_by_vdevice()` and `hailo_configure_vdevice()`.

Get the networks information We use the `hailo_configured_network_group` object we got in the previous step, to get the networks information, `hailo_network_info_t` objects.

Used APIs: `hailo_get_network_infos()`.

Creating vstreams For each network in the network group, the vstream params need to be initialized (both input and output), and then they are ready to create the vstreams for each network. The vstreams will be ready for use only after activating the network group.

Note: To create the vstream for a specific network, one should pass the specific network's name when making the vstream's params. It is also possible to use the network group's name to create all vstreams for all networks together. For more information see the documentation for `hailo_make_input_vstream_params()` and `hailo_make_output_vstream_params()`.

Used APIs: `hailo_make_input_vstream_params()`, `hailo_make_output_vstream_params()`, `hailo_create_input_vstreams()` and `hailo_create_output_vstreams()`

Activating a network group Before starting inference, we need to activate the network group.

Used APIs: `hailo_activate_network_group()`.

Inference In this example, `p_thread` is used for inference, with a sub-thread for each output vstream receiving data and a sub-thread for each input vstream sending data.

Write to device The write thread, will firstly initialize a buffer used for send. Then the thread will send all frames in a loop.

Used APIs: `hailo_get_input_vstream_frame_size()` and `hailo_vstream_write_raw_buffer()`.

Read from device

The read function is analogical to the write. First, initialize a receive buffer, then receive all frames in a loop.

Used APIs: `hailo_get_output_vstream_frame_size()` and `hailo_vstream_read_raw_buffer()`.

Deactivating a network group After inference is done, we need to deactivate the network group.

Used APIs: `hailo_deactivate_network_group()`.

5.1.5. Automatic Switch Between Multiple Network Groups Using Virtual Streams - `switch_network_groups_example`

See: `hailort/libhailort/examples/c/switch_network_groups_example.c`

Summary

- Demonstrates inference of multiple network groups over a single device, using VDevice and Round-Robin scheduling algorithm.
- Multiple HEFs are configured into the VDevice, creating multiple network groups.
- For each network group, create virtual streams, send threads and receive threads.
- Wait for all threads to finish, and validate their results.

See also:

See the [comparison table](#) between context switching and network groups switching for more information about running multiple models.

Code Structure

The code is similar to `hailort/libhailort/examples/c/vstreams_example.c`, so be sure to read the `vstreams_example` first.

Device initialization Initialize the `hailo_vdevice_params_t` with default values, and then create the `VDevice`. The `hailo_vdevice_params_t::scheduling_algorithm` is set by default to `HAILO_SCHEDULING_ALGORITHM_ROUND_ROBIN`.

Used APIs: `hailo_create_vdevice()`.

Pre infer stage

Configuring the vdevice from HEF (for each HEF) The next step is to create a `hailo_hef`, and to use it to configure the vdevice for inference. Initialize a `hailo_configure_params_t` with default values, configure the vdevice to get a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params_by_vdevice()` and `hailo_configure_vdevice()`.

Set scheduler's timeout threshold, and priority Set a higher threshold and timeout for the first model scheduling. It will give priority to the scheduling of the second model. `HAILO_SCHEDULER_PRIORITY_NORMAL+1` is also set for a priority for the first. The practical meaning is that the first network will be ready to run only if `SCHEDULER_THRESHOLD` send requests have been accumulated, or more than `SCHEDULER_TIMEOUT_MS` time has passed and at least one send request has been accumulated. However when both the first and the second networks are ready to run, the first network will be preferred over the second network.

Used APIs: `hailo_set_scheduler_timeout()`, `hailo_set_scheduler_threshold()` and `hailo_set_scheduler_priority()`.

Build vstreams (for each network group) The next step is to create the input and output virtual streams for each configured network. First, make the default virtual stream params for each virtual stream. Second, create the virtual streams (inputs and outputs). Finally, extract the vstream's frame size and allocate the host input and output buffers and initialize the source with random data to send to inference.

Used APIs: `hailo_make_input_vstream_params()`, `hailo_make_output_vstream_params()`, `hailo_create_input_vstreams()`, `hailo_create_output_vstreams()`, `hailo_get_input_vstream_frame_size()` and `hailo_get_output_vstream_frame_size()`.

Inference In this example, create a new thread for each input vstream using the function `create_input_vstream_thread()`. Each created thread sends data to the device using the function `write_to_device()`.

Write to device The write thread gets as input the following:

- `input_vstream` – The virtual input stream.
- `src_data` – The actual data sent to the HW.
- `src_frame_size` – Size of the data to be sent to the HW.

The write thread will send all frames by writing the HW buffer to the device.

Used APIs: `hailo_vstream_write_raw_buffer()`.

A new thread for each output vstream is also created using the function `create_output_vstream_thread`. Each created thread receives data from the device using the function `read_from_device`.

Read from device The read function gets as input the following:

- `output_vstream` – The virtual output stream.
- `dst_data` – The actual data sent by the HW back to the host.
- `dst_frame_size` – Size of the data expected to be received by the host.

The read function will receive all frames in the following flow:

- Reading data from the device.
- No post processing is made on the received data.

Used APIs: `hailo_vstream_read_raw_buffer()`.

In order to complete the inference on each network group, the main thread waits for all the input vstream threads to finish via `hailo_join_thread`, and afterwards waits for all the output vstreams threads to finish.

5.1.6. User Controlled Switch Between Multiple Hefs Using Virtual Streams - `switch_network_groups_manually_example`

See: `hailort/examples/c/switch_network_groups_manually_example.c`

Summary

- Demonstrates inference of multiple network groups over a single device, using VDevice.
- For simplicity, single input / single output network groups are used.
- Multiple HEFs are configured into the VDevice, creating multiple network groups.
- Create send thread and recv thread. the vstreams for each network group will be created inside these threads.
- Wait for all threads to finish, and validate their results.

See also:

See the [comparison table](#) between context switching and network groups switching for more information about running multiple models.

Code Structure

Device initialization Initialize the `hailo_vdevice_params_t` with default values, and then the `hailo_vdevice_params_t::scheduling_algorithm` is set to `HAILO_SCHEDULING_ALGORITHM_NONE` in order to manually switch the network groups, and finally, create the `VDevice`.

Used APIs: `hailo_create_vdevice()`.

Pre infer stage The next step is to create a `hailo_hef`, and to use it to configure the device for inference. Initialize a `hailo_configure_params_t` with default values, configure the device and get a `hailo_configured_network_group` object. When complete, create `hailo_input_vstream_params_by_name_t` and `hailo_output_vstream_params_by_name_t` for the generated configured network group. Repeat this step `HEF_COUNT` times, and store all the configured network groups and vstream parameters in an array.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params_by_vdevice()`, `hailo_configure_vdevice()`, `hailo_make_input_vstream_params()`, and `hailo_make_output_vstream_params()`.

Creating VStream params and Vstream infos

The next step is to create the virtual stream params. The virtual stream params are created from the network group using the functions `hailo_make_input_vstream_params()` and `hailo_make_output_vstream_params()`. Then, all vstream infos are obtained from the HEF.

Used APIs: `hailo_make_input_vstream_params()`,
`hailo_make_output_vstream_params()` and `hailo_hef_get_all_vstream_infos()`

Inference In this example, one thread for inputs, and one thread for outputs are created, assuming we only have one input and one output per network group. The threads are responsible for creating the VStreams, allocating buffers for send / recv, and synchronizing the network group switches using `hailo_activate_network_group()` and `hailo_deactivate_network_group()` APIs. the first activation of the first network group is done in the main thread.

Used APIs: `hailo_activate_network_group()`.

Input vstream thread function The write thread gets as its input the following:

- `configured_networks` – The `hailo_configured_network_group` array.
- `input_vstream_params` – The `hailo_input_vstream_params_by_name_t` array.

Both arrays are in the length of HEF_COUNT.

The write thread will first create `hailo_input_vstream`, and allocate a buffer (`src_data`) representing the input buffers.

Used APIs: `hailo_create_input_vstreams()` and `hailo_get_input_vstream_frame_size()`.

Afterwards the write thread will loop over the configured network group and wait for activation (which will take place in the read thread). Once activated, we loop for INFER_FRAME_COUNT, generating random data over `src_data`, and write it to the device.

Used APIs: `hailo_wait_for_network_group_activation()` and `hailo_vstream_write_raw_buffer()`.

Once finished, we release the created `hailo_input_vstream` using `hailo_release_input_vstreams()`.

Output vstream thread function The read thread gets as input the following:

- `configured_networks` – The `hailo_configured_network_group` array.
- `activated_network_group` – a `hailo_activated_network_group`, holding the first activated network group.
- `output_vstream_params` – The `hailo_output_vstream_params_by_name_t` array.

Both arrays are in the length of HEF_COUNT.

The read thread will first create `hailo_output_vstream`, and allocate a buffer (`dst_data`) representing the output buffers.

Used APIs: `hailo_create_output_vstreams()` and `hailo_get_output_vstream_frame_size()`.

Afterwards the read thread will loop over the configured network group and wait for activation (which should return immediately, as the network group is already activated). Once activated, loop for INFER_FRAME_COUNT, reading from the device.

Used APIs: `hailo_wait_for_network_group_activation()` and `hailo_vstream_read_raw_buffer()`.

When the reading is completed, deactivate the current activated network group, and activate the next network group so that the input thread will be able to start sending again.

Used APIs: `hailo_deactivate_network_group()` and `hailo_activate_network_group()`.

Once finished, we release the created `hailo_output_vstream` using `hailo_release_output_vstreams()`.

In order to complete the inference on each network group, the main thread waits for all the threads to finish via `hailo_join_thread`, and release the resources.

Used APIs: `hailo_release_hef()` and `hailo_release_vdevice()`.

5.1.7. Quantization of Inputs/Outputs - *data_quantization_example*

See: `hailort/libhailort/examples/c/data_quantization_example.c`

Summary

Hailo devices require input data to be quantized/scaled before it is sent to the device via a `hailo_input_stream`. Similarly, data outputted from the device via a `hailo_output_stream` needs to be 'de-quantized'/rescaled as well.

When a neural network (NN from now on) is compiled by the Dataflow Compiler, each input/output layer in the NN is assigned two floating point values that are parameters to an input/output transformation: `qp_zp` (zero_point) and `qp_scale` (fields of the struct `hailo_quant_info_t`). These values are stored in the HEF.

- Input transformation: input data is divided by `qp_scale` and then `qp_zp` is added to the result.
- Output transformation: `qp_zp` is subtracted from output data and then the result is multiplied by `qp_scale`.

In the context of the HailoRT library, there are two options to quantize the data:

- Use a virtual stream, and mark it as non-quantized (`float32_t`) in the vstream params creation. The data sent to the virtual stream will be quantized as part as the transformation process. See the [vstreams example](#) for more info about virtual streams.
- Use a raw stream. In this case the data needs to be quantized using `hailo_input_transform_context` or `hailo_output_transform_context`. See the [raw streams example](#) for more info about streams and transformations.

In this tutorial we use virtual streams for the quantization process.

In this tutorial we use the following arguments:

- *in* – The input is quantized, hence HailoRT won't quantize the input.
- *out* – The output is to be left quantized, hence HailoRT won't de-quantize the output.
- *both* – The input is quantized and the output is to be left quantized, hence HailoRT won't do either.
- *none* – The input isn't quantized and the output is to be de-quantized, hence HailoRT will do both.

Code Structure

The code is similar to `hailort/libhailort/examples/c/vstreams_example.c`, so be sure to read the [vstreams example](#) first.

Device initialization and configuration Create a vdevice and configure it using HEF. The code is similar to the one in the [vstreams example](#)

Creating virtual streams In the function `create_vstreams` both input and output virtual stream are created. At this point we can set the desired quantization behavior for input streams and de-quantization behavior for output streams via the `format_type` argument.

- Passing `HAILO_FORMAT_TYPE_AUTO` in the function `hailo_make_input_vstream_params()`, means that the input data sent to the stream (via `hailo_vstream_write_raw_buffer()`) is in the same `format_type` as used by the device, and therefore quantized to begin with. This will result in an input stream that doesn't quantize the input data. Passing `HAILO_FORMAT_TYPE_FLOAT32` will lead to input data being quantized.
- Passing `HAILO_FORMAT_TYPE_AUTO` in the function `hailo_make_output_vstream_params()`, means that the output data received from the stream (via `hailo_vstream_read_raw_buffer()`) is in the same `format_type` as used by the device, and therefore to remain quantized (such as it is upon exiting the device). This will result in an output stream that doesn't de-quantize the output data. Passing `HAILO_FORMAT_TYPE_FLOAT32` will lead to output data being de-quantized.

Note: On some cases (such as NMS output) the output's *format_type* HAILO_FORMAT_TYPE_AUTO stands for HAILO_FORMAT_TYPE_FLOAT32, and therefore will result in output de-quantization.

The rest of the code After creating the virtual streams and activating the network group, get the stream handles and commence with the inference. This code is almost identical to the code used in `hailort/libhailort/examples/c/vstreams_example.c`. For an explanation, see the [vstreams example](#).

5.1.8. Inference Using Raw Streams – `raw_streams_example`

See: `hailort/libhailort/examples/c/raw_streams_example.c`

Summary

- Demonstrates basic inference of a shortcut network using raw stream API.
- The program uses the first identified Hailo device.
- The data sent/received does not undergo the transformation process as required on raw streams. For a complete raw stream example with transformation, see [raw_streams_example](#).

Code Structure

Device initialization Open the first enumerated Hailo device (by passing NULL as value for parameter `device_id` to `hailo_create_device_by_id()`)

Used APIs: `hailo_create_device_by_id()`.

Configuring the device from HEF The next step is to create a `hailo_hef`, and to use it to configure the device for inference. Initialize a `hailo_configure_params_t` with default values, configure the device to get a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params_by_device()` and `hailo_configure_device()`.

Activating a network group Before starting inference, we need to activate the network group.

Used APIs: `hailo_activate_network_group()`.

Inference In this example, `p_thread` is used for inference, with a sub-thread for each output vstream receiving data and a sub-thread for each input vstream sending data.

Write to device The write thread, will firstly initialize any necessarily resources, including:

- `host_data` – Will contain the actual user data (for example - frame).
- `hw_data` – The actual data sent to the hw.
- `transform_context` – a `hailo_input_transform_context` - used to transform the data from host format to HW.

Used APIs: `hailo_network_group_get_input_stream_infos()`, `hailo_get_input_stream()`, `hailo_create_input_transform_context()` and `hailo_get_host_frame_size()`.

After initialization, the write thread will send all frames:

- First, transform the buffer from host format to HW format
- Then writing the HW buffer to the device.

Used APIs: `hailo_transform_frame_by_input_transform_context()` and `hailo_stream_write_raw_buffer()`.

Read from device The read thread, will firstly initialize any necessarily resources, including:

- *host_data* - Will contain the actual user data (The network result).
- *hw_data* - The actual data received from the hw.
- *transform_context* a `hailo_output_transform_contexts` - used to transform the data from HW format to host.

Used APIs: `hailo_network_group_get_output_stream_infos()`,
`hailo_get_output_stream()`, `hailo_create_output_transform_context()`
and `hailo_get_host_frame_size()`.

After initialization, the write thread will send all frames:

- First, reading data from the device.
- Then transform is from HW format to host format.

Used APIs: `hailo_stream_read_raw_buffer()` and `hailo_transform_frame_by_output_transform()`

5.1.9. Async Inference Using Raw Streams - *raw_async_streams_single_thread_example*

See: `hailort/libhailort/examples/c/raw_async_streams_single_thread_example.c`

Summary

- Demonstrates basic async inference of a shortcut network using raw async stream API.
- Using single thread for the inference.
- The program uses the first Hailo device found.
- The data sent/received does not undergo the transformation process as required on raw streams. For a complete raw stream example with transformation, see [raw_streams_example](#).

Code Structure

Device Initialization Open the first enumerated Hailo device (by passing NULL as value for parameter *device_id* to `hailo_create_device_by_id()`)

Used APIs: `hailo_create_device_by_id()`.

Configuring The Device From HEF The next step is to create a `hailo_hef`, and use it to configure the device for inference. Initialize a `hailo_configure_params_t` with default values and to support async streams API and set the `HAILO_STREAM_FLAGS_ASYNC` flag inside `hailo_stream_parameters_t`. Configure the device using the `hailo_configure_device()` function which returns a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params_by_device()` and `hailo_configure_device()`.

Activating a network group Before starting inference, activate the network group.

Used APIs: `hailo_activate_network_group()`.

Prepare Async Inference Before launching async operations, some preparation is required.

Async streams are backed by limited-sized queue for pending async transfers. The max queue sizes are provided by `hailo_input_stream_get_async_max_queue_size()` and `hailo_output_stream_get_async_max_queue_size()`. In this example, the maximum number of ongoing transfers across all streams, denoted as *ongoing_transfers*, is calculated by taking the minimum value among the queues of all streams.

If a user attempts to perform more concurrent async operations than the value of *ongoing_transfers*, they may encounter an error with the code `HAILO_QUEUE_IS_FULL`.

Used APIs: `hailo_input_stream_get_async_max_queue_size()` and `hailo_output_stream_get_async_max_queue_size()`.

Async Inference

- For each output stream, *ongoing_transfers* async read requests are launched.
 1. For each transfer, allocate memory that is aligned to the system page size, as required by `hailo_stream_read_raw_buffer_async()`.
 2. The async read operation is initiated by calling `hailo_stream_read_raw_buffer_async()`, passing the allocated buffer.
 3. Once a read operation is finished, the associated async callback function is triggered. In this example, the callback function examines the `hailo_stream_read_async_completion_info_t::status` that is passed to it. If the status is `HAILO_SUCCESS`, indicating a successful transfer, a new async read request is initiated using the same buffer and callback.
- In a real application the results obtained by the read operations can be passed for post-processing or display.
- For each input stream, *ongoing_transfers* async writes requests are launched.
 1. For each transfer, allocate memory that is aligned to the system page size, as required by `hailo_stream_write_raw_buffer_async()`. In this example, the buffer initially contains garbage data.
 2. Map each buffer allocated to the device by explicitly calling `hailo_device_dma_map_buffer()`. This stage is not mandatory, but it can improve performance when the same buffer is used several times.
 3. The async write operation is initiated by calling `hailo_stream_write_raw_buffer_async()`, passing the allocated buffer.
 4. Once a write operation is finished, the associated async callback function is triggered. In this example, the callback function examines the `hailo_stream_write_async_completion_info_t::status` that is passed to it. If the status is `HAILO_SUCCESS`, indicating a successful transfer, a new async write request is initiated using the same buffer and callback.

Real applications may choose to fill the buffer with a new frame, free it, or return it to some buffer pool for later use.

- Since further async operations are launched within each callback, the inference is executed in parallel to the main thread. In this example, the main thread remains idle and sleeps for 5 seconds, allowing the inference to run in the background.
- To stop the inference process, we call `hailo_shutdown_network_group()`. Any transfers that have not been completed will be called with the status `HAILO_STREAM_ABORT`. After all callbacks have been called, the buffers can be safely released together with any other variable passed in the callback opaque.

Used APIs: `hailo_device_dma_map_buffer()`, `hailo_device_dma_unmap_buffer()`,
`hailo_stream_read_raw_buffer_async()` `hailo_stream_write_raw_buffer_async()`,

Note: It is important to ensure that the buffers passed to the inference and any variables passed in the async callback opaque remain valid until all callbacks are called. Failure to do so may result in the callback being called with invalid variables, leading to undefined behavior.

5.1.10. Inference Using Inference Pipeline – *infer_pipeline_example*

See: `hailort/examples/c/infer_pipeline_example.c`

Summary

- Demonstrates basic inference of a shortcut network (i.e inputs are sent through the device and right back out, without any changes made to the data).
- The program uses the Hailo virtual device with default values for initialization.
- The data is both sent to the device and received using the inference pipeline.

Code Structure

Device initialization Create the `VDevice` without passing any parameters, which means use `hailo_vdevice_params_t` with default values. Because it is enabled by default, the model scheduler is enabled (with its default scheduling scheme - Round Robin). For that reason, there is no need to activate/deactivate the network groups in the user application, since when the scheduler is enabled, the activation and deactivation is done automatically.

Used APIs: `hailo_create_vdevice()`

Configuring the vdevice from HEF The next step is to create a `hailo_hef`, and to use it to configure the vdevice for inference. Initialize a `hailo_configure_params_t` with default values, configure the vdevice to get a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params_by_vdevice()` and `hailo_configure_vdevice()`.

Creating VStream params and Vstream infos The next step is creating the virtual stream params. Create the virtual stream params from the network group using the functions `hailo_make_input_vstream_params()` and `hailo_make_output_vstream_params()`. Then, we get all vstream infos from the HEF.

Used APIs: `hailo_make_input_vstream_params()`, `hailo_make_output_vstream_params()` and `hailo_hef_get_all_vstream_infos()`

Preparing buffers before inference The input data buffers, and also the buffers for the inference output must be prepared.

Used APIs: `hailo_get_vstream_frame_size()`.

Inference Finally, run inference on the input data using the inference pipeline.

Used APIs: `hailo_infer()`.

5.1.11. Power Measurement – *power_measurement_example*

See: `hailort/libhailort/examples/c/power_measurement_example.c`

Note: This example is not supported on Hailo-15.

Summary

- Demonstrates how to perform a continuous power measurement on the chip.
- The program uses a VDevice created from all the Hailo devices that are connected to the computer.
- In this tutorial, the following arguments to control the measurement type will be used:
 - *power* – measure power consumption in W.
 - *current* – measure current in mA.

Note: For instantaneous power measurement, see `hailo_power_measurement()`.

For instantaneous temperature measurement, see `hailo_get_chip_temperature()`.

Code Structure

VDevice initialization First, scan for devices, then initialize the `hailo_vdevice_params_t` with the device count scanned before. The next step is to create the `VDevice`.

Used APIs: `hailo_create_vdevice()`, `hailo_init_vdevice_params()`
`hailo_scan_devices()`.

Power measurement

Initialization For sending controls to a `hailo_device`, one needs to get the underlying physical devices from the `hailo_vdevice`, using the function `hailo_get_physical_devices()`. For each physical device, first call the function `hailo_stop_power_measurement()` to make sure there aren't any former measurements currently running. The next step is to set power measurement arguments using the function `hailo_set_power_measurement()`, and to start the measurement using the function `hailo_start_power_measurement()`

Used APIs: `hailo_get_physical_devices()`, `hailo_stop_power_measurement()`,
`hailo_set_power_measurement()` and `hailo_start_power_measurement()`.

Stopping measurement and fetching the results After starting the measurement, pause for a constant period of time (5 seconds). After this time, review all the physical devices and do the following:

- Stop the current measurement using the function `hailo_stop_power_measurement()`.
- Obtain the measurement results using the function `hailo_get_power_measurement()`.
- Print measurement results.

Used APIs: `hailo_stop_power_measurement()`, `hailo_get_power_measurement()`.

5.1.12. Notification Callback - *notification_callback_example*

See: `hailort/libhailort/examples/c/notification_callback_example.c`

Summary

- Demonstrates the basic usage of notification callbacks.
- The program creates a device and then sets and removes a notification callback on it.

Code Structure

Device initialization

First, scan for a Hailo device using `hailo_scan_devices()`. Then, use the `hailo_device_id_t` obtained from the scan to create the device by calling `hailo_create_device_by_id()` with it.

Used APIs: `hailo_scan_devices()`, `hailo_create_device_by_id()`.

Creating the callback Create a `hailo_notification_callback` and assign it to the address of the callback function `void callback(hailo_device device, const hailo_notification_t *notification, void *opaque)`, that prints an overcurrent alarm message.

Set the callback notification

By calling `hailo_set_notification_callback()` with the following notification id: `hailo_notification_id_t::HAILO_NOTIFICATION_ID_HEALTH_MONITOR_OVERCURRENT_ALARM`, the callback function will be called when will get this notification.

Used APIs: `hailo_set_notification_callback()`.

Remove the callback notification

By calling `hailo_remove_notification_callback()` with the same `hailo_notification_id_t` as in the `hailo_set_notification_callback()`, remove this notification callback.

Used APIs: `hailo_remove_notification_callback()`.

5.2. C++ Inference Tutorial

In this section various examples of the HailoRT API are provided. The code mentioned below is included in the release under `hailort/libhailort/examples/cpp` in the GitHub repository. An installed HailoRT is a prerequisite for running the examples. See [documentation](#).

Note: (for C++ API users) In order to maintain ABI Integrity between libhailort and the examples code that uses the C++ API, you should [compile libhailort](#) instead of using the pre-compiled binaries.

All function calls are based on the headers provided in the directory `hailort/include/hailo/`. One can include `hailort/include/hailo/hailort.hpp` to access all functions. See the [API documentation](#) for more details.

5.2.1. Compilation and Execution

Compiling examples is done using the following commands from the examples directory:

```
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release  
cmake --build build --config release
```

In order to compile a specific example, either add the example name as a target with a `cpp` prefix, or run the `cmake` command from the target example directory:

```
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release  
cmake --build build --config release --target cpp_async_infer_basic_example
```

```
cd cpp/async_infer_basic_example  
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release  
cmake --build build --config release
```

Note: For debug builds of C++ examples, it is recommended to build the examples as part of *source compilation of HailoRT*.

Note: The examples use exceptions. See [Tutorials](#) for instructions on how to disable them.

Running examples:

Run the examples using the following command, from the examples directory:

```
build/cpp/<example_name>/cpp_<example_name> [params...]
```

Cross-Compile Tutorials

Add the following options while configuring the examples using CMake:

- `DCMAKE_TOOLCHAIN_FILE=<toolchain file>`
- `DCMAKE_FIND_ROOT_PATH=<target dir>`
- `DCMAKE_FIND_ROOT_PATH_MODE_PACKAGE=ONLY`

Note: For more details, see:

- [Cross Compiling With CMake](#)
 - [find_package — CMake 3.26.1 Documentation](#)
-

5.2.2. Power Measurement - `power_measurement_example`

See: `hailort/libhailort/examples/cpp/power_measurement_example.cpp`

Note: This example is not supported on Hailo-15.

Summary

- Demonstrates how to perform a continuous power measurement on the device.
- The program uses a VDevice created from all the Hailo devices that are connected to the computer.
- In this tutorial we use the following arguments to control the measurement type:
 - *power* – measure power consumption in W.
 - *current* – measure current in mA.

Note: For instantaneous power measurement, see `hailort::Device::power_measurement()`.

For instantaneous temperature measurement, see `hailort::Device::get_chip_temperature()`.

Code Structure

VDevice initialization First, scan for PCIe devices, then initialize the `hailo_vdevice_params_t` with the device count scanned before. The next step is to create the C++.

Used APIs: `hailort::VDevice::create()`, `hailo_init_vdevice_params()`
`hailort::Device::scan_pcie()`.

Power measurement

Initialization In order to send controls to a `hailort::Device`, it will be necessary to obtain the underlying physical devices from the `hailort::VDevice`, using the function `hailort::VDevice::get_physical_devices()`. For each physical device, first call the function `hailort::Device::stop_power_measurement()` to make sure there aren't any former measurements currently running. The next step is to set power measurement arguments using the function `hailort::Device::set_power_measurement()`, and to start the measurement using the function `hailort::Device::start_power_measurement()`

Used APIs: `hailort::VDevice::get_physical_devices()`, `hailort::Device::stop_power_measurement()`, `hailort::Device::set_power_measurement()` and `hailort::Device::start_power_measurement()`.

Stopping measurement and fetching the results After starting the measurement, pause for a constant period of time (5 seconds). After this time, review all the physical devices and do the following:

- Stop the current measurement using the function `hailort::Device::stop_power_measurement()`.
- Obtain the measurement results using the function `hailort::Device::get_power_measurement()`.
- Print measurement results.

Used APIs: `hailort::Device::stop_power_measurement()`, `hailort::Device::get_power_measurement()`.

5.2.3. Notification Callback - *notification_callback_example*

See: `hailort/libhailort/examples/cpp/notification_callback_example.cpp`

Summary

- Demonstrates the basic usage of notification callbacks.
- The program creates a device and then sets and removes a notification callback on it.

Code Structure

Device initialization

First, scan for Hailo devices using `hailort::Device::scan()`, then use the first device_id that was found for creating the device, using the function `hailort::Device::create()`.

Used APIs: `hailort::Device::create()`, `hailort::Device::scan()`.

Set the callback notification

By calling `Device::set_notification_callback()` with a lambda function that prints an overcurrent alarm message, and the following notification id `:hailo_notification_id_t::HAILO_NOTIFICATION_ID_HEALTH_MONITOR_OVERCURRENT_ALARM`, the callback function will be called when will get this notification.

Used APIs: `Device::set_notification_callback()`.

Remove the callback notification

By calling `Device::remove_notification_callback()` with the same `hailo_notification_id_t` as in the `Device::set_notification_callback()`, remove this notification callback.

Used APIs: `Device::remove_notification_callback()`.

5.2.4. Async Inference - *async_infer_basic_example*

See: `hailort/libhailort/examples/cpp/async_infer_basic_example.cpp`

Summary

- Demonstrates the basic usage of async inference.
- The program creates a virtual device and runs async inference on a model.

Code Structure

Virtual Device Initialization

The example creates a virtual device, using the function `hailort::VDevice::create()`.

Used APIs: `hailort::VDevice::create()`.

Inference Model Initialization and Configuration

The example creates an inference model based on the given HEF file, using the function `hailort::VDevice::create_infer_model()`. Then the example configures the inference model using the function `hailort::InferModel::configure()`.

Used APIs: `hailort::VDevice::create_infer_model()`, `hailort::InferModel::configure()`.

Input and Output Buffers Assignment

The example creates input and output buffers and sets them as `hailort::MemoryView` buffers for the inference model, using the function `hailort::ConfiguredInferModel::Bindings::InferStream::set_buffer()`

Used APIs: `hailort::ConfiguredInferModel::Bindings::InferStream::set_buffer()`.

Note: Buffer guards are used to ensure the buffers won't be deleted during inference.

Running Async Inference

The example runs async inference and waits for the job to finish using the functions `hailort::ConfiguredInferModel::run_async()`, and `hailort::AsyncInferJob::wait()`.

Used APIs: `hailort::ConfiguredInferModel::run_async()`, `hailort::AsyncInferJob::wait()`.

5.2.5. Async Inference - `async_infer_advanced_example`

See: `hailort/libhailort/examples/cpp/async_infer_advanced_example.cpp`

Summary

- Demonstrates advanced functionality for async inference.
- The program creates a virtual device and runs async inference on a multi-planar network.

Code Structure

Device Initialization

The example creates a virtual device, using the function `hailort::VDevice::create()`.

Used APIs: `hailort::VDevice::create()`.

Inference Model Initialization And Configuration

The example creates an inference model based on the given HEF file, using the function `hailort::VDevice::create_infer_model()`. The example then sets the output format type to HAILO_FORMAT_TYPE_FLOAT32 using `hailort::InferModel::InferStream::set_format_type()`, sets the batch size to BATCH_SIZE (defined at the top of the example) using `hailort::InferModel::InferStream::set_batch_size()` and following this the example configures the inference model using the function `hailort::InferModel::configure()`.

Used APIs: `hailort::VDevice::create_infer_model()`, `hailort::InferModel::InferStream::set_format_type()`, `hailort::InferModel::InferStream::set_size()`, `hailort::InferModel::configure()`.

Input And Output Buffers - Pre Mapping And Assignment

The example creates input and output buffers and sets them as `hailo_pix_buffer_t` input buffers and `hailort::MemoryView` output buffers for the inference model, using the

```
functions    hailort::ConfiguredInferModel::Bindings::InferStream::  
:set_pix_buffer() and hailort::ConfiguredInferModel::Bindings::  
InferStream::set_buffer()
```

Since the same buffers are used for inference, the buffers can be pre-mapped to the `hailort::VDevice` by constructing a `hailort::DmaMappedBuffer` object. Pre-mapping can increase performance and reduce the latency of each frame. The buffers will be mapped until the object is destroyed.

Used APIs: `hailort::DmaMappedBuffer::create()`, `hailort::ConfiguredInferModel::Bindings::InferStream::set_pix_buffer()`, `hailort::ConfiguredInferModel::Bindings::InferStream::set_buffer()`.

Note: Buffer-guards are used to ensure the buffers won't be deleted during inference, and buffer-map-guards are used to ensure the buffers won't be un-mapped after they are deleted.

Running Async Inference

The example runs async inference on all the frames and waits for the last async inference job to finish using the functions `hailort::ConfiguredInferModel::run_async()`, and `hailort::AsyncInferJob::wait()`.

Used APIs: `hailort::ConfiguredInferModel::run_async()`, `hailort::AsyncInferJob::wait()`.

5.2.6. Async Inference - *async_infer_dma_buffer_example*

See: `hailort/libhailort/examples/cpp/async_infer_dma_buffer_example.cpp`

Summary

- Demonstrates the usage of async inference using DMA buffers.
- The program creates a virtual device and runs async inference on a model using DMA buffers.

Code Structure

Virtual Device Initialization

The example creates a virtual device, using the function `hailort::VDevice::create()`.

Used APIs: `hailort::VDevice::create()`.

Inference Model Initialization and Configuration

The example creates an inference model based on the given HEF file, using the function `hailort::VDevice::create_infer_model()`. Then the example configures the inference model using the function `hailort::InferModel::configure()`.

Used APIs: `hailort::VDevice::create_infer_model()`, `hailort::InferModel::configure()`.

Input and Output Buffers Assignment

The example creates input and output DMA buffers and sets them for the inference model, using the function `hailort::ConfiguredInferModel::Bindings::InferStream::set_dma_buffer()`.

Used APIs: `hailort::ConfiguredInferModel::Bindings::InferStream::set_dma_buffer()`.

Note: Buffer guards are used to ensure the buffers won't be deleted during inference.

Running Async Inference

The example runs async inference and waits for the job to finish using the functions `hailort::ConfiguredInferModel::run_async()`, and `hailort::AsyncInferJob::wait()`.

Used APIs: `hailort::ConfiguredInferModel::run_async()`, `hailort::AsyncInferJob::wait()`.

5.2.7. Query for performance and health stats - *query_performance_and_health_stats_example*

See: `hailort/libhailort/examples/cpp/query_performance_and_health_stats_example/query_performance_and_health_stats_example.cpp`

Summary

- Demonstrates how to query the device for performance and health statistics.
- The program scans for all physical Hailo devices that are in the system, and loops through those devices, retrieving performance and health statistics for each.

Note: For instantaneous performance stats, see `hailort::Device::query_performance_stats()`.

For instantaneous health stats, see `hailort::Device::query_health_stats()`.

Code Structure

First the PCIe devices are scanned, then for each device id a `hailort::Device` object is created. For each Device object, call the functions `:cpp:func:hailort::Device::query_performance_stats()` to get the performance stats, and `:cpp:func:hailort::Device::query_health_stats()` to get the health stats and print the results.

Used APIs: `hailort::Device::scan()`, `hailort::Device::create()`,
`hailort::Device::query_performance_stats()`, `hailort::Device::query_health_stats()`.

5.2.8. Multi Network Async Inference - *multi_model_inference_example*

See: `hailort/libhailort/examples/cpp/multi_model_inference_example.cpp`

Summary

- Demonstrates the basic usage of multi-network inference.
- The program creates a virtual device and runs async inference on multiple models.

Code Structure

Virtual Device Initialization

The example creates a virtual device, using the function `hailort::VDevice::create()`.

Used APIs: `hailort::VDevice::create()`.

Inference Model Initialization and Configuration

The example creates two inference models, each based on the given HEF file, using the function `hailort::VDevice::create_infer_model()`. Then the example configures the inference model using the function `hailort::InferModel::configure()`.

Used APIs: `hailort::VDevice::create_infer_model()`, `hailort::InferModel::configure()`.

Input and Output Buffers Assignment

The example creates input and output buffers and sets them as `hailort::MemoryView` buffers for the inference model, using the function `hailort::ConfiguredInferModel::Bindings::InferStream::set_buffer()`

Used APIs: `hailort::ConfiguredInferModel::Bindings::InferStream::set_buffer()`.

Note: Buffer guards are used to ensure the buffers won't be deleted during inference.

Running Async Inference

The example runs async inference and waits for the job to finish using the functions `hailort::ConfiguredInferModel::run_async()`, and `hailort::AsyncInferJob::wait()`.

Used APIs: `hailort::ConfiguredInferModel::run_async()`, `hailort::AsyncInferJob::wait()`.

5.2.9. Multi Process Inference - *multi_process_example*

See: `hailort/libhailort/examples/cpp/multi_process_example.cpp` and `hailort/libhailort/examples/cpp/multi_process_example.sh -h`

Summary

- Demonstrates how to run multiprocess inference on a single Hailo-10H device, using the HailoRT C++ API.
- The program creates a virtual device with a shared Group ID and runs async inference on a model in multiple processes.

Running the Multi-Process Example

Options for running with multiple processes:

- 1) **From the examples directory, in each terminal run the following command:** ` ./build/cpp/cpp_multi_process_example`
- 2) **Using the script** `hailort/libhailort/examples/cpp/multi_process_example.sh`:
The script is running the example in multiple processes, its default is to run two instances of the inference example. In the script options, one can specify how many processes to run inference on (up to 8 processes). See `hailort/libhailort/examples/cpp/multi_process_example.sh -h` for more information.

Code Structure

Virtual Device Initialization

The example creates a virtual device with a shared Group ID, using the function `hailort::VDevice::create()`. The shared Group ID is used to share the virtual device between multiple processes. There is no need to set the `hailo_vdevice_params_t::multi_process_service` flag, as this functionality is supported by default on Hailo-10H devices.

Used APIs: `hailort::VDevice::create()`.

Inference Model Initialization and Configuration

The example creates an inference model, using the function `hailort::VDevice::create_infer_model()`. Then the example configures the inference model using the function `hailort::InferModel::configure()`.

Used APIs: `hailort::VDevice::create_infer_model()`, `hailort::InferModel::configure()`.

Input and Output Buffers Assignment

The example creates input and output buffers and sets them as `hailort::MemoryView` buffers for the inference model, using the function `hailort::ConfiguredInferModel::Bindings::InferStream::set_buffer()`

Used APIs: `hailort::ConfiguredInferModel::Bindings::InferStream::set_buffer()`.

Running Async Inference

The example runs async inference and waits for the job to finish using the functions `hailort::ConfiguredInferModel::run_async()` and `hailort::AsyncInferJob::wait()`.

Used APIs: `hailort::ConfiguredInferModel::run_async()`, `hailort::AsyncInferJob::wait()`.

5.3. GenAI Inference Tutorial

In this section various examples of the HailoRT GenAI API are provided. The code mentioned below is included in the release under `hailort/libhailort/examples/genai` in the GitHub repository. An installed HailoRT is a prerequisite for running the examples. See [documentation](#).

Note: Before running the HailoRT GenAI tutorials, ensure you first download the appropriate HEF files from the Hailo Model Zoo GenAI repository: https://github.com/hailo-ai/hailo_model_zoo_genai.git

Note: (for C++ API users) In order to maintain ABI Integrity between libhailort and the examples code that uses the C++ API, you should [compile libhailort](#) instead of using the pre-compiled binaries.

All function calls are based on the headers provided in the directory `hailort/include/hailo/`. One can include `hailort/include/hailo/hailort.hpp` to access all functions. See the [API documentation](#) for more details.

5.3.1. Compilation and Execution

Compiling examples is done using the following commands from the examples directory:

```
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release  
cmake --build build --config release
```

In order to compile a specific example, either add the example name as a target, or run the cmake command from the target example directory. For instance, to compile the VLM (vision-language model) example:

```
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release  
cmake --build build --config release --target vlm_example
```

```
cd genai/vlm_example  
cmake -S . -Bbuild -DCMAKE_BUILD_TYPE=Release  
cmake --build build --config release
```

Note: For debug builds of C++ examples, it is recommended to build the examples as part of *source compilation of HailoRT*.

Note: The examples use exceptions. See [Tutorials](#) for instructions on how to disable them.

Running examples:

Run the examples using the following command, from the examples directory:

```
build/cpp/<example_name>/<example_name> [params...]
```

Cross-Compile Tutorials

Add the following options while configuring the examples using CMake:

- DCMAKE_TOOLCHAIN_FILE=<toolchain file>
- DCMAKE_FIND_ROOT_PATH=<target dir>
- DCMAKE_FIND_ROOT_PATH_MODE_PACKAGE=ONLY

Note: For more details, see:

- [Cross Compiling With CMake](#)
 - [find_package — CMake 3.26.1 Documentation](#)
-

Memory Optimization and Compilation Requirements

Client-Side Tokenization for Memory Optimization

For large GenAI models that may exceed device memory, HailoRT provides an optional memory optimization feature that enables client-side tokenization. This feature moves tokenization processing from the device to the host, significantly reducing device memory usage and enabling large models to run on small-memory devices.

Enabling Memory Optimization:

1. **Compile libhailort with client tokenizer support:**

```
cmake -S . -Bbuild -DHAILO_BUILD_CLIENT_TOKENIZER=ON  
cmake --build build --config release
```

2. Use the `optimize_memory_on_device` parameter in your code:

```
// For LLM  
auto llm_params = hailort::genai::LLMParams(hef_path, lora_name, true);  
  
// For VLM  
auto vlm_params = hailort::genai::VLMPParams(hef_path, true);
```

Prerequisites:

- **Rust Toolchain:** The client tokenizer requires Rust to be installed on the build machine:
 - Install Rust: `curl -proto '=https' -LsSf https://sh.rustup.rs | sh`
 - Ensure `cargo` and `rustup` are available in your PATH
 - Restart your shell after installation
- **CMake Configuration:** Set `HAILO_BUILD_CLIENT_TOKENIZER=ON` when configuring the build

Benefits:

- **Reduced Device Memory Usage:** Tokenization artifacts are processed on the host instead of the device
- **Support for Larger Models:** Enables running models that would otherwise exceed device memory limits
- **Flexible Memory Management:** Choose between device-side (default) and host-side tokenization based on your requirements

Trade-offs:

- **Host Processing Overhead:** Tokenization is performed on the host, adding some latency
- **Build Complexity:** Requires Rust toolchain and additional compilation time
- **Binary Size:** Increases the final library size due to additional tokenizer components

5.3.2. LLM (Large Language Model) – *chat_example*

See: `hailort/libhailort/examples/genai/chat_example/chat.cpp`

Summary

- Implements an interactive chat interface using Hailo's LLM APIs.
- Creates a virtual device to host the LLM.
- Demonstrates LLM inference through an interactive loop using user-provided prompts.

Code Structure

Virtual Device Initialization The first step is to create a `C++`, using the function `hailort::VDevice::create_shared()`.

Used APIs: `hailort::VDevice::create_shared()`.

LLM Configuration The next step is to create `hailort::genai::LLMParams` with the appropriate HEF path, and create a `hailort::genai::LLM` instance using the virtual device and parameters.

Note: This step is computationally heavy and is expected to take a few seconds.

Memory Optimization: For large models that may exceed device memory, you can enable client-side tokenization by setting the `optimize_memory_on_device` parameter to `true` when creating LLMPParams. This moves tokenization to the host, reducing device memory usage and enabling large models to run on small-memory devices.

```
// Create LLMPParams with memory optimization enabled
auto llm_params = hailort::genai::LLMParams(hef_path, lora_name, true); // Optimization
    ↪ optimize_memory_on_device = true

// Create LLM instance
auto llm = hailort::genai::LLM::create(vdevice, llm_params).expect("Failed to
    ↪ create LLM");
```

Note: Memory optimization requires libhailort to be compiled with `HAILO_BUILD_CLIENT_TOKENIZER=ON` and requires Rust toolchain (cargo, rustup) to be installed on the build machine.

Used APIs: `hailort::genai::LLMParams`, `hailort::genai::LLM::create()`.

Interactive Chat Loop The example enters an interactive loop where users provide text prompts. For each interaction:

1. **Get user input:** The prompt is obtained from the user via standard input.
2. **Create JSON structured prompt:** The user input is formatted as a JSON message with role “user” and the content.
3. **Direct generation:** The `hailort::genai::LLM::generate()` function is called directly with the JSON prompt, returning a `hailort::genai::LLMGeneratorCompletion` object.
4. **Read and display tokens:** The example reads generated tokens using `hailort::genai::LLMGeneratorCompletion::read()` and displays them in real-time while checking the generation status.

```
while (true) {
    std::string user_input = get_user_prompt();
    std::vector<std::string> prompt = {
        R"({\"role\": \"user\", \"content\": \"")" + user_input + R"(\\"})"
    };

    auto generator_completion = llm.generate(prompt).expect("Failed to generate
    ↪ ");
    while (hailort::genai::LLMGeneratorCompletion::Status::GENERATING ==
    ↪ generator_completion.generation_status()) {
        auto output = generator_completion.read().expect("read failed!");
        std::cout << output << std::flush;
    }
    std::cout << std::endl;
}
```

The direct generation API handles all generator creation and management automatically, making the code simple and easy to understand.

Note: It is possible to pass custom generation parameters to the direct API by using `hailort::genai::LLM::generate()` with `hailort::genai::LLMGeneratorParams`:

```
auto params = llm.create_generator_params().expect("Failed to create params");
params.set_temperature(0.7);
auto completion = llm.generate(params, prompt).expect("Failed to generate");
```

Unlike LLM creation, this operation is light and fast and can be done for every generation.

Used APIs: `hailort::genai::LLM::generate()`, `hailort::genai::LLMGeneratorCompletion::generation_status()` and `hailort::genai::LLMGeneratorCompletion::read()`.

Advanced LLM Usage: Generator-Based API

For advanced scenarios requiring explicit control over generation parameters, conversation building, or performance optimization, one can use the generator-based API. This approach provides fine-grained control over the generation process.

When to Use Generators:

- **Consecutive writes:** When dealing with large input text, it is possible to write the text in multiple chunks and generate the response in a single call.
- **Performance optimization:** Avoiding repeated parameter setup for batch processing
- **Complex conversation management:** Explicit control over conversation state and history

Generator Creation and Parameter Control:

```
// Create generator with custom parameters
auto generator_params = llm.create_generator_params().expect("Failed to create generator");
generator_params.set_temperature(0.7);
generator_params.set_max_generated_tokens(150);
generator_params.set_top_p(0.9);
generator_params.set_top_k(40);
auto generator = llm.create_generator(generator_params).expect("Failed to create generator");
auto completion = generator.generate().expect("Failed to generate");
```

Consecutive Writes for Complex Conversations:

```
// Build conversation incrementally with consecutive writes
auto generator = llm.create_generator().expect("Failed to create generator");

// System message
generator.write({R"({"role": "system", "content": "You are a helpful programming assistant"})"});

// User question
generator.write({R"({"role": "user", "content": "How do I implement a binary search in C++?"})"});

// Assistant response (for conversation history)
generator.write({R"({"role": "assistant", "content": "Here's a binary search implementation: [previous response]"})"});

// Follow-up question
generator.write({R"({"role": "user", "content": "Can you explain the time complexity?"})"});

// Generate response to the complete conversation
auto completion = generator.generate();
```

Parameter Reuse for Batch Processing:

```
// Create generator once with optimized parameters
auto generator_params = llm.create_generator_params().expect("Failed to create generator");
// (continues on next page)
```

(continued from previous page)

```
generator_params.set_temperature(0.3); // Lower temperature for consistent responses
generator_params.set_max_generated_tokens(100);

auto generator = llm.create_generator(generator_params).expect("Failed to create
→generator");

// Process multiple prompts with same parameters
std::vector<std::string> prompts = {
    R"({{"role": "user", "content": "Summarize: Machine learning is..."}})",
    R"({{"role": "user", "content": "Summarize: Artificial intelligence is..."}})",
    R"({{"role": "user", "content": "Summarize: Deep learning is..."}})"
};

for (const auto& prompt : prompts) {
    generator.write({prompt});
    auto completion = generator.generate();
    // Process completion...
    llm.clear_context(); // Clear for next prompt
}
```

Advantages of Generator-Based Approach:

- **Fine-grained parameter control:** Set specific temperature, top_p, top_k values per generator
- **Conversation state management:** Build complex multi-turn conversations incrementally
- **Performance optimization:** Reuse generator configuration for multiple generations
- **Memory efficiency:** Explicit control over when to clear conversation context
- **Advanced features:** Access to additional generation parameters and settings

Used APIs: `hailort::genai::LLM::create_generator_params()`, `hailort::genai::LLM::create_generator()`, `hailort::genai::LLMGenerator::write()`, `hailort::genai::LLMGenerator::generate()`, `hailort::genai::LLM::clear_context()`.

5.3.3. VLM (Vision-Language Model) - *vlm_example*

See: `hailort/libhailort/examples/genai/vlm_example/vlm_example.cpp`

Note: This example requires a valid input frame (or "NONE" if no frame is used) and user-provided prompts.

Summary

- Implements an interactive interface combining vision and language inputs using Hailo's VLM APIs.
- Accepts an optional input frame file and a text prompt for inference.
- Creates a virtual device to host the VLM.
- Demonstrates VLM inference through an interactive loop using user-provided input.

Code Structure

Virtual Device Initialization The first step is to create a C++ using the function `hailort::VDevice::create_shared()`.

Used APIs: `hailort::VDevice::create_shared()`.

VLM Configuration The next step is to configure the model by creating `hailort::genai::VLMPParams` with the HEF path. The `hailort::genai::VLM` is then created using the virtual device and parameters.

Note: This step is computationally heavy and is expected to take a few seconds.

Memory Optimization: For large VLM models that may exceed device memory, you can enable client-side tokenization by setting the `optimize_memory_on_device` parameter to `true` when creating VLMPParams. This moves tokenization to the host, reducing device memory usage and enabling large models to run on small-memory devices.

```
// Create VLMPParams with memory optimization enabled
auto vlm_params = hailort::genai::VLMPParams(hef_path, true); // optimize_
↪memory_on_device = true

// Create VLM instance
auto vlm = hailort::genai::VLM::create(vdevice, vlm_params).expect("Failed to
↪create VLM");
```

Note: Memory optimization requires libhailort to be compiled with HAILO_BUILD_CLIENT_TOKENIZER=ON and requires Rust toolchain (cargo, rustup) to be installed on the build machine.

Used APIs: `hailort::genai::VLMPParams`, `hailort::genai::VLM::create()`.

Frame Buffer Creation After creating the VLM, fetch the input-frame-size using `hailort::genai::VLM::input_frame_size()` to pre-allocate a buffer with the correct size.

Used APIs: `hailort::genai::VLM::input_frame_size()`, `hailort::Buffer::create()`.

Interactive Inference Loop The example enters an interactive loop where users provide frame paths and text prompts. For each interaction:

1. **Get frame input:** Users specify an input frame path or "NONE" for text-only inference.
2. **Load frame if provided:** If a frame is specified, fetch the input-frame-size using `hailort::genai::VLM::input_frame_size()`, allocate a buffer with the correct size and load the frame into it.

Note: The frame-shape, format-order and format-type of this frame are fixed and pre-determined by the model, and can be obtained using `hailort::genai::VLM::input_frame_shape()`, `hailort::genai::VLM::input_frame_format_order()` and `hailort::genai::VLM::input_frame_format_type()`.

3. **Get text prompt:** Users provide a text prompt for the VLM.
4. **Create JSON structured messages:** Based on whether an image is provided, the appropriate JSON message structure is created:
 - With image: Content includes both image and text elements
 - Text-only: Content includes only text element
5. **Direct generation:** The `hailort::genai::VLM::generate()` function is called directly with the JSON messages and input frames, returning a `hailort::genai::LLMGeneratorCompletion` object.

6. **Read and display tokens:** Generated tokens are read and displayed in real-time.

The example uses structured prompts in JSON format, where each message contains a role ("user") and content array. For text-only prompts, the content contains only a text element. For prompts with images, the content includes both an image placeholder and text.

Note: Some VLM models support multiple input frames in a single generation. In that case, one should make sure that the number of frames matches the number of image entries in the structured messages.

```
// Process multiple images in a single generation call
std::vector<std::string> multi_image_messages = {
    R"({\"role\": \"user\", \"content\": [
        {\"type\": \"image\"},
        {\"type\": \"image\"},
        {\"type\": \"text\", \"text\": \"Compare these two images\"}
    ]})"
};

std::vector
```

The direct generation API automatically validates that the number of provided frames matches the number of image references in the JSON messages, and handles all generator creation and management internally.

Note: It is possible to pass custom generation parameters to the direct API by using `hailort::genai::VLM::generate()` with `hailort::genai::LLMGeneratorParams`:

```
auto params = vlm.create_generator_params().expect("Failed to create params");
params.set_temperature(0.5);
params.set_max_generated_tokens(200);
auto completion = vlm.generate(params, messages, input_frames).expect("Failed to generate");
```

Since the conversation context is not cleared between generations (can be done using the function `hailort::genai::VLM::clear_context()`), the conversation continues naturally without manual prompt management.

Used APIs: `hailort::genai::VLM::generate()`, `hailort::genai::LLMGeneratorCompletion::generation_status()`, and `hailort::genai::LLMGeneratorCompletion::read()`.

Advanced VLM Usage: Generator-Based API

For performance-critical applications, batch processing, or if explicit control over generator lifecycle and parameters is needed, the generator-based API provides advanced capabilities beyond the simplified direct generation approach.

When to Use VLM Generators:

- **Batch processing:** Process multiple images with the same conversation parameters
- **Parameter persistence:** Maintain consistent generation parameters across multiple inferences
- **Performance optimization:** Avoid repeated generator creation overhead
- **Multi-modal workflows:** Complex scenarios with varying image and text combinations
- **Resource management:** Explicit control over generator lifecycle and memory usage

Generator Creation with Custom Parameters:

```
// Create generator with custom parameters for consistent responses
auto generator_params = vlm.create_generator_params().expect("Failed to create
    ↪generator_params");
generator_params.set_temperature(0.5); // Lower temperature for more consistent
    ↪image descriptions
generator_params.set_max_generated_tokens(200);
generator_params.set_top_p(0.8);

auto generator = vlm.create_generator(generator_params).expect("Failed to create
    ↪generator");
```

Batch Image Processing with Parameter Reuse:

```
// Create generator once for processing multiple images
auto generator_params = vlm.create_generator_params().expect("Failed to create
    ↪generator_params");
generator_params.set_temperature(0.3); // Consistent analysis across images
auto generator = vlm.create_generator(generator_params).expect("Failed to create
    ↪generator");

// Prepare consistent analysis prompt
std::vector<std::string> analysis_messages = {
    R"({{"role": "system", "content": "You are an expert image analyzer. Provide
        ↪detailed, consistent analysis."}}",
    R"({{"role": "user", "content": [{"type": "image"}, {"type": "text", "text":
        ↪"Analyze this image in detail"}]}}"
};

// Process multiple images with same parameters and conversation context
std::vector image_batch = {image1_data, image2_data, image3_
    ↪data};

for (const auto& image_data : image_batch) {
    auto completion = generator.generate(analysis_messages, {image_data});

    // Process completion...
    while (hailort::genai::LLMGeneratorCompletion::Status::GENERATING == completion.
        ↪generation_status()) {
        auto output = completion.read().expect("read failed!");
        // Save or process output...
    }

    // Optionally clear context between images
    vlm.clear_context();
}
```

Multi-Modal Conversation Building:

```
// Build complex multi-modal conversations
auto generator = vlm.create_generator().expect("Failed to create generator");

// Start with system context and first image analysis
std::vector<std::string> first_image_message = {
    R"({{"role": "system", "content": "You are a visual assistant helping with image
        ↪comparison tasks."}}",
    R"({{"role": "user", "content": [{"type": "image"}, {"type": "text", "text":
        ↪"Describe this first image"}]}}"
};
auto completion1 = generator.generate(first_image_message, {first_image_data});

// Second image with comparison request
std::vector<std::string> comparison_message = {
```

(continues on next page)

(continued from previous page)

```
R"({\"role\": \"user\", \"content\": [{\"type\": \"image\"}, {"type\": \"text\", \"text\": \"Now\n→compare this second image with the first one\"}]} )\n};\nauto completion2 = generator.generate(comparison_message, {second_image_data});
```

Used APIs: `hailort::genai::VLM::create_generator_params()`, `hailort::genai::VLM::create_generator()`, `hailort::genai::VLMGenerator::generate()` and `hailort::genai::VLM::clear_context()`.

5.3.4. Structured Prompts

Both LLM and VLM APIs support structured prompts using JSON format, enabling more flexible and standardized interactions.

LLM Structured Prompts

The LLM API supports structured prompts through the `hailort::genai::LLMGenerator::write()` and `hailort::genai::LLM::generate()` methods with JSON messages:

```
std::vector<std::string> prompt_json_strings = {\n    R"({\"role\": \"system\", \"content\": \"You are a helpful assistant\"})",\n    R"({\"role\": \"user\", \"content\": \"Hello, how are you?\"})\"\n};\ngenerator.write(prompt_json_strings);\nauto completion = generator.generate();
```

VLM Structured Prompts

The VLM API supports structured prompts through the `hailort::genai::VLMGenerator::generate()` and `hailort::genai::VLM::generate()` methods with JSON messages:

```
// Text-only conversation\nstd::vector<std::string> messages = {\n    R"({\"role\": \"user\", \"content\": [{\"type\": \"text\", \"text\": \"Hello\"}]} )\"\n};\nstd::vector<hailort::MemoryView> frames; // Empty for text-only\nauto completion = generator.generate(messages, frames);
```

```
// With image content\nstd::vector<std::string> messages_with_image = {\n    R"({\"role\": \"user\", \"content\": [\n        {\"type\": \"image\"},\n        {\"type\": \"text\", \"text\": \"Describe this image\"}\n    ]})\"\n};\nstd::vector<hailort::MemoryView> image_frames = {image_data};\nauto completion = generator.generate(messages_with_image, image_frames);
```

Message Format

Structured messages follow this JSON format:

```
{  
  "role": "user|system|assistant",  
  "content": [  
    {  
      "type": "text",  
      "text": "text content"  
    },  
    {  
      "type": "image"  
    }  
  ]  
}
```

Supported Roles:

- **system**: System instructions and context
- **user**: User queries and inputs
- **assistant**: Assistant responses (for conversation history)

Content Types:

- **text**: Text content with `text` field
- **image**: Image placeholder (actual image data provided via `input_frames` for VLM)

Advanced Usage Patterns

Multi-Message Conversations:

```
std::vector<std::string> conversation = {  
  R"({\"role\": \"system\", \"content\": \"You are a helpful programming assistant\"})",  
  R"({\"role\": \"user\", \"content\": \"How do I sort an array in Python?\"})",  
  R"({\"role\": \"assistant\", \"content\": \"You can use the sorted() function\"})",  
  R"({\"role\": \"user\", \"content\": \"What's the difference between sorted() and .sort()?\"})"  
};
```

Multiple Images (VLM):

```
std::vector<std::string> messages = {  
  R"({\"role\": \"user\", \"content\": [  
    {\"type\": \"image\"},  
    {\"type\": \"image\"},  
    {\"type\": \"text\", \"text\": \"Compare these two images\"}  
  ]})"  
};  
std::vector<hailort::MemoryView> frames = {image1_data, image2_data};
```

Context Management:

```
// Clear conversation history when needed  
llm.clear_context(); // For LLM  
vlm.clear_context(); // For VLM
```

Image Count Validation (VLM)

The VLM API automatically validates that the number of provided image frames matches the number of image entries in messages. This prevents common integration errors:

```
// Valid: 2 images in message, 2 frames provided
std::vector<std::string> messages = {
    R"({\"role\": \"user\", \"content\": [
        {\"type\": \"image\"}, {\"type\": \"image\"},
        {\"type\": \"text\", \"text\": \"Compare these\"}
    ]})"
};

std::vector
```

Benefits:

- **Standardization:** Consistent message format across models
- **Flexibility:** Easy conversation management with role-based messages
- **Template Integration:** Automatic prompt template application
- **Multi-modal Support:** Seamless text and image content handling (VLM)

Best Practices

1. **Use structured prompts** for better organization and template integration
2. **Validate JSON format** before sending messages
3. **Use appropriate roles** (system, user, assistant) for different message types
4. **Handle errors gracefully** with proper exception handling
5. **Clear context periodically** for long conversations
6. **Validate image counts** before calling generate() (VLM)
7. **Follow JSON standards** for message formatting

5.3.5. Speech2Text (Whisper) - *speech2text_example*

See: [hailort/libhailort/examples/genai/speech2text_example/speech2text_example.cpp](https://github.com/hailo-ai/libhailort/tree/main/examples/genai/speech2text_example/speech2text_example.cpp)

Summary

- Demonstrates speech-to-text transcription using Hailo's Speech2Text API (Whisper model).
- Loads an audio file and performs speech-to-text transcription with configurable task and language settings.
- Shows how to generate transcription segments with timestamps.

Code Structure

Virtual Device Initialization As with other GenAI models, create a C++ using `hailort::VDevice::create_shared()`.

Used APIs: `hailort::VDevice::create_shared()`.

Speech2Text Configuration Create `hailort::genai::Speech2TextParams` with the appropriate HEF path, and instantiate a `hailort::genai::Speech2Text` object.

```
// Create Speech2TextParams
auto speech2text_params = hailort::genai::Speech2TextParams(speech2text_hef_
    ↪path);

// Create Speech2Text instance
auto speech2text = hailort::genai::Speech2Text::create(vdevice, speech2text_
    ↪params).expect("Failed to create Speech2Text");
```

Used APIs: `hailort::genai::Speech2TextParams`, `hailort::genai::Speech2Text::create()`.

Audio Input Format Requirements

The audio input must be in a specific format for proper processing:

- **Format:** PCM float32 normalized to [-1.0, 1.0]
- **Channels:** Mono (single channel)
- **Endianness:** Little-endian
- **Sample Rate:** 16 kHz

The example loads the audio file as a binary buffer:

```
// Load audio file as binary buffer
auto audio_chunk = get_input_audio_file(input_audio_file_path);
```

Generator Parameters Configuration

Create generator parameters and configure the task and language settings:

```
// Create generator parameters
auto speech2text_generator_params = speech2text.create_generator_
    ↪params().expect("Failed to create generator params");

// Set the task (TRANSCRIBE or TRANSLATE)
speech2text_generator_params.set_task(hailort::genai::Speech2TextTask::
    ↪TRANSCRIBE);

// Set the language (ISO-639-1 two-letter code)
speech2text_generator_params.set_language("en");
```

Task Options:

- `hailort::genai::Speech2TextTask::TRANSCRIBE` - Transcribe speech to text in the same language
- `hailort::genai::Speech2TextTask::TRANSLATE` - Translate speech to text in a different language

Language Format: Use ISO-639-1 two-letter codes (e.g., "en", "fr", "de", "es", "it"). An empty string means that the model will automatically detect the language.

Used APIs: `hailort::genai::Speech2Text::create_generator_params()`, `hailort::genai::Speech2TextGeneratorParams::set_task()`, `hailort::genai::Speech2TextGeneratorParams::set_language()`.

Generating Transcription Segments

The example uses `hailort::genai::Speech2Text::generate_all_segments()` to get detailed transcription with timestamps:

```
// Generate transcription segments with timestamps
auto segments = speech2text.generate_all_segments(hailort::
    ↪MemoryView(audio_chunk), speech2text_generator_params).expect("Failed
    ↪to generate");

// Print each segment with its timing information
for (const auto &segment : segments) {
    std::cout << "[" << segment.start_sec << ", " << segment.end_sec << "] " <<
    ↪segment.text << std::endl;
}
```

Each segment contains:

- **start_sec**: Start timestamp in seconds relative to audio beginning
- **end_sec**: End timestamp in seconds relative to audio beginning
- **text**: The transcribed text for this segment

Used APIs: `hailort::genai::Speech2Text::generate_all_segments()` and `hailort::genai::Speech2Text::SegmentInfo`.

Alternative: Generate Complete Text

For cases where only the complete transcription without timestamps is required, use `hailort::genai::Speech2Text::generate_all_text()`:

```
// Generate complete transcription as a single string
auto complete_text = speech2text.generate_all_text(hailort::
    ↪MemoryView(audio_chunk), speech2text_generator_params).expect("Failed
    ↪to generate");

std::cout << "Complete transcription: " << complete_text << std::endl;
```

Used APIs: `hailort::genai::Speech2Text::generate_all_text()`.

5.4. Python Async Inference Tutorial - Single Model

This tutorial describes how to run an inference process using `InferModel(Async)` API, which is the recommended option

Requirements:

- Run the notebook inside the Python virtual environment: `source hailo_virtualenv/bin/activate`

When inside the `virtualenv`, use the command `jupyter-notebook <tutorial-dir>` to open a Jupyter server that contains the tutorials (default folder on GitHub: `hailort/libhailort/bindings/python/platform/hailo_tutorials/notebooks/`).

5.4.1. HEF

An HEF is Hailo's binary format for neural networks. The HEF files contain:

- Target HW configuration
- Weights
- Metadata for HailoRT (e.g., input/output scaling)

```
[ ]: import numpy as np
from hailo_platform import VDevice

timeout_ms = 1000

# The vdevice is used as a context manager ("with" statement) to ensure it's released on
# time.
with VDevice() as vdevice:

    # Create an infer model from an HEF:
    infer_model = vdevice.create_infer_model('../hefs/resnet_v1_18.hef')

    # Configure the infer model and create bindings for it
    with infer_model.configure() as configured_infer_model:
        bindings = configured_infer_model.create_bindings()

        # Set input and output buffers
        buffer = np.empty(infer_model.input().shape).astype(np.uint8)
        bindings.input().set_buffer(buffer)

        buffer = np.empty(infer_model.output().shape).astype(np.uint8)
        bindings.output().set_buffer(buffer)

        # Run synchronous inference and access the output buffers
        configured_infer_model.run([bindings], timeout_ms)
        buffer = bindings.output().get_buffer()

    # Run asynchronous inference
    job = configured_infer_model.run_async([bindings])
    job.wait(timeout_ms)
```

5.5. Python Async Inference Tutorial - Multiple Models

This tutorial describes how to run an inference process with multiple models using `InferModel` (Async) API, which is the recommended option

Requirements:

- Run the notebook inside the Python virtual environment: `source hailo_virtualenv/bin/activate`

When inside the `virtualenv`, use the command `jupyter-notebook <tutorial-dir>` to open a Jupyter server that contains the tutorials (default folder on GitHub: `hailort/libhailort/bindings/python/platform/hailo_tutorials/notebooks/`).

```
[ ]: # Optional: define a callback function that will run after the inference job is done
      # The callback must have a keyword argument called "completion_info".
      # That argument will be passed by the framework.
def example_callback(completion_info, bindings):
    if completion_info.exception:
        # handle exception
```

(continues on next page)

(continued from previous page)

```
pass

_=bindings.output().get_buffer()
```

```
[ ]: import numpy as np
from functools import partial
from hailo_platform import VDevice, FormatType

number_of_frames = 4
timeout_ms = 10000

def infer():
    # Create a VDevice
    params = VDevice.create_params()
    params.group_id = "SHARED"
    with VDevice(params) as vdevice:

        # Create an infer model from an HEF:
        infer_model = vdevice.create_infer_model('../hefs/resnet_v1_18.hef')

        # Set optional infer model parameters
        infer_model.set_batch_size(2)

        # For a single input / output model, the input / output object
        # can be accessed with a name parameter ...
        infer_model.input("resnet_v1_18/input_layer1").set_format_type(FormatType.
→FLOAT32)
        # ... or without
        infer_model.output().set_format_type(FormatType.FLOAT32)

        # Once the infer model is set, configure the infer model
        with infer_model.configure() as configured_infer_model:
            for _ in range(number_of_frames):
                # Create bindings for it and set buffers
                bindings = configured_infer_model.create_bindings()
                bindings.input().set_buffer(np.empty(infer_model.input().shape).
→astype(np.float32))
                bindings.output().set_buffer(np.empty(infer_model.output().shape).
→astype(np.float32))

                # Wait for the async pipeline to be ready, and start an async inference job
                configured_infer_model.wait_for_async_ready(timeout_ms=10000)

                # Any callable can be passed as callback (lambda, function, functools.
→partial), as long
                # as it has a keyword argument "completion_info"
                job = configured_infer_model.run_async([bindings], partial(example_
→callback, bindings=bindings))

                # Wait for the last job
                job.wait(timeout_ms)
```

5.5.1. Running Multiple Models Concurrently

The models can be run concurrently using either multiple `Thread` objects or multiple `Process` objects.

```
[ ]: from threading import Thread

pool = [
    Thread(target=infer),
    Thread(target=infer)
]

print('Starting async inference on multiple models using threads')

for job in pool:
    job.start()
for job in pool:
    job.join()

print('Done inference')
```

5.6. Python Power Measurement Tutorial

This tutorial shows how to perform a power measurement on the device.

The Hailo device supports power measurement which is done via the control protocol, if an INA231 is assembled on the board.

Requirements:

- Run the notebook inside the Python virtual environment: `source hailo_virtualenv/bin/activate`

Attention:

- These examples should run in a different process than the one that performs the actual inference.

When inside the `virtualenv`, run `jupyter-notebook <tutorial-dir>` to open a Jupyter server that contains the tutorials (default folder on GitHub: `hailort/libhailort/bindings/python/platform/hailo_tutorials/notebooks/`).

5.6.1. Single Power Measurement

```
[ ]: import time

from hailo_platform import Device, DvmTypes, PowerMeasurementTypes, SamplingPeriod, AveragingFactor, MeasurementBufferIndex # noqa F401
```

Initialize the hardware object:

```
[ ]: target = Device()
```

When using the `power_measurement()` function with no parameters, the function tries to detect which Hailo-10 board is connected (evaluation board, M.2 or mPCIe) and determine the DVM accordingly (at the moment only the boards referred to in this section have INA231 assembled and are supported).

The parameter `dvm` (of type `DvmTypes`) defines which DVM will be measured. The user can choose a specific DVM or choose the default DVM. The meaning of the default DVM changes according to the board or module in use.

The default for the evaluation board is the sum of three DVMs: `DvmTypes.VDD_CORE`, `DvmTypes.MIPI_AVDD` and `DvmTypes.AVDD_H`. The sum of these three DVMs approximates the total power consumption

of the chip in PCIe setups. Only power can be measured using this default option, as voltage and current cannot be totaled (or calculated) this way.

The default for platforms supporting current monitoring, such as M.2 and mPCIe modules, is `DvmTypes.OVERCURRENT_PROTECTION`, which measures the power consumption of the whole module.

See the API documentation for further details about the supported DVMs and measurement types.

```
[ ]: single_power_measurement = target.control.power_measurement()
      print('Power from single measurement: {} W'.format(single_power_measurement))
```

5.6.2. Periodic Power Measurement

In the following example, a periodic power measurement is taken.

A measurement index is selected. It is a member of the enum class `MeasurementBufferIndex` (between 0 and 3). The DVM is not given so the default DVM will be used, as explained above.

```
[ ]: buffer_index = MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_0
      target.control.set_power_measurement(buffer_index=buffer_index)
```

The following call to `start_power_measurement()` allows you to configure the power sampling frequency. In this case we keep the default configuration. The API documentation provides more details.

```
[ ]: target.control.start_power_measurement()
```

Clear the old samples and statistics (min, max, average) each time the measurement is taken from the chip.

```
[ ]: should_clear = True
```

The power measurement is read every second, for 10 seconds.

The firmware calculates the min, max, and average of all the values from the sensor. Note that the average calculated by the firmware is the “average of averages”. This is the average of all values that have already been averaged by the sensor. The host then requests these values from the firmware every second by calling the `get_power_measurement()` function. The host can also read the average time interval between new sensor values.

```
[ ]: for _ in range(10):
      time.sleep(1)
      # Get saved power measurement values from the firmware.
      measurements = target.control.get_power_measurement(buffer_index=buffer_index, ↴
      ↪should_clear=should_clear)
      print('Average power is {} W. Min power is {} W. Max power is {} W.\nAverage time ↴
      ↪between power samples is {} ms\n'.format(measurements.average_value, measurements. ↪
      ↪min_value, measurements.max_value, measurements.average_time_value_
      ↪milliseconds))

      # Stop performing periodic power measurement
      target.control.stop_power_measurement()

      target.release()
```

5.7. Python LLM (Large Language Model) Tutorial

This tutorial demonstrates how to use the LLM Python API for running Large Language Models on Hailo hardware.

The LLM API provides both streaming and non-streaming text generation capabilities, context management, and various generation parameters for fine-tuned control over model outputs.

Best Practice: Structured Prompts This tutorial uses **structured prompts** (list of JSON messages) exclusively. Structured prompts provide better control, consistency, and leverage the model's chat template effectively.

Best Practice: context-manager This tutorial does not use context-manager, to share resources between different cells. Make sure to create VDevice and LLM using 'with' statements whenever possible. When not using 'with', use VDevice.release() and LLM.release() to clean up resources.

Requirements:

- Run the notebook inside the Python virtual environment: `source hailo_virtualenv/bin/activate`
- An LLM HEF file (Hailo Executable Format for Large Language Models)
- Optional: LoRA (Low-Rank Adaptation) name for fine-tuned models inside this HEF

Memory Optimization (Optional):

- For large models that may exceed device memory, enable client-side tokenization
- Requires libhailort to be compiled with `HAILO_BUILD_CLIENT_TOKENIZER=ON`
- Requires Rust toolchain (cargo, rustup) to be installed on the build machine
- Set `OPTIMIZE_MEMORY_ON_DEVICE = True` in the configuration section below

Tutorial Structure:

- Basic LLM initialization and simple generation, Streaming vs non-streaming
- Generation parameters (temperature, top_p, top_k, etc.)
- Context management for multi-turn conversations
- Advanced features: templates, tokenization, stop tokens

When inside the `virtualenv`, use the command `jupyter-notebook <tutorial-dir>` to open a Jupyter server that contains the tutorials (default folder on GitHub: `hailort/libhailort/bindings/python/platform/hailo_tutorials/notebooks/`).

```
[ ]: # LLM Tutorial: Setup and Configuration

from hailo_platform import VDevice
from hailo_platform.genai import LLM

# Configuration - Update these paths for your setup
MODEL_PATH = "/your/hef/path/llm.hef" # Update this path
LORA_NAME = "" # Optional: specify LoRA adapter name

print("Model path: {}".format(MODEL_PATH))
print("LoRA adapter: {}".format(LORA_NAME if LORA_NAME else 'None'))
vdevice = VDevice()
print("Initializing LLM... this may take a moment...")
llm = LLM(vdevice, MODEL_PATH, LORA_NAME)
print("LLM initialized successfully!")
```

5.7.1. Streaming Generation Example

```
[ ]: # Structured prompt (recommended approach)
structured_prompt = [
    {"role": "system", "content": "You are a helpful AI assistant."},
    {"role": "user", "content": "Explain machine learning in 2 sentences."}
]

with llm.generate(structured_prompt, max_generated_tokens=50, seed=31) as generation:
    for token in generation:
        print(token, end="", flush=True)
```

5.7.2. Non-Streaming Generation Example

```
[ ]: llm.clear_context()
print(llm.generate_all(structured_prompt, max_generated_tokens=50, seed=31))
```

5.7.3. Multi-Turn Conversations Example

```
[ ]: # Clear context
llm.clear_context()

# Turn 1: Introduction
conversation_1 = [
    {"role": "system", "content": "You are a helpful tutor."},
    {"role": "user", "content": "Hi! I'm learning Python. What's a list?"}
]

print(llm.generate_all(conversation_1, max_generated_tokens=60))
print()

# Turn 2: Follow-up (context maintained automatically)
followup = [
    {"role": "user", "content": "Can you show me an example?"}
]

print(llm.generate_all(followup, max_generated_tokens=60, seed=6))
```

5.7.4. Context Management Example

Context is maintained between generate() calls automatically. Use 'clear_context()' to start fresh conversations Use 'get_context_usage_size()' to see how many tokens are used in this conversation

```
[ ]: question = [
    {"role": "user", "content": "What is your profession?"}

print(llm.generate_all(question, max_generated_tokens=30))
new_question = [
    {"role": "user", "content": "What were we just discussing?"}
```

(continues on next page)

(continued from previous page)

```
]
print(llm.generate_all(new_question, max_generated_tokens=30))
print("Number of tokens in the context - {}".format(llm.get_context_usage_size()))

# Clear context and ask again
llm.clear_context()
print("Number of tokens in the context after 'llm.clear_context()' - {}".format(llm.
    get_context_usage_size()))

print(llm.generate_all(new_question, max_generated_tokens=30))
```

5.8. Function Calling / Tool Use Example

What are Tools?

Tools (also called “function calling”) allow LLMs to interact with external functions and APIs. Instead of just generating text, the model can request to call specific functions with appropriate parameters to retrieve real-time information, perform calculations, or interact with external services.

How to Use Tools:

1. **Define Tools:** Create tool definitions in JSON schema format (following OpenAI’s function calling schema). Each tool needs a name, description, and parameter specifications.
2. **Provide Tools:** Pass the `tools` list to the first `generate()` or `generate_all()` call. Tools can only be provided on a **fresh context** (first call or after `clear_context()`).
3. **Parse Model Response:** The model will output a tool call request (format depends on the model’s chat template). Parse the tool name and arguments from the response.
4. **Execute Function:** Run the actual function in your code (e.g., call a weather API) and get the result.
5. **Feed Result Back:** Continue the conversation by providing the tool result in your next message. The model will use this result to generate its final response.

Important Notes: - The model doesn’t execute functions - it only requests tool calls - Tools are registered once per context and persist throughout the conversation - To change tools, call `clear_context()` first

Learn More: [HuggingFace Tool Use Guide](#)

```
[ ]: # Clear context to start fresh (required for providing tools)
llm.clear_context()

# Step 1: Define tools (JSON schema format)
tools = [
    {
        "type": "function",
        "function": {
            "name": "get_current_temperature",
            "description": "Get current temperature at a location.",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "The location to get the temperature for, in the format \
                            \"City, State, Country\"."
                    },
                    "unit": {
                        "type": "string",
                    }
                }
            }
        }
]
```

(continues on next page)

(continued from previous page)

```
        "enum": ["celsius", "fahrenheit"],
        "description": "The unit to return the temperature in. Defaults to \
↪celsius\".
    }
},
"required": ["location"]
}
},
{
    "type": "function",
    "function": {
        "name": "get_temperature_date",
        "description": "Get temperature at a location and date.",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "The location to get the temperature for, in the format \
↪City, State, Country\".
                },
                "date": {
                    "type": "string",
                    "description": "The date to get the temperature for, in the format \"Year-\
↪Month-Day\"."
                },
                "unit": {
                    "type": "string",
                    "enum": ["celsius", "fahrenheit"],
                    "description": "The unit to return the temperature in. Defaults to \
↪celsius\".
                }
            },
            "required": ["location", "date"]
        }
    }
}
]

# Step 2: First turn - User asks about current temperature, tools are provided
user_question = [
    {"role": "user", "content": "What's the current temperature in Paris, France?"}
]

response = llm.generate_all(user_question, tools=tools, max_generated_tokens=60, \
↪timeout_ms=30000)
print("User: What's the current temperature in Paris, France?")
print("\nModel response:")
print(response)

# Step 3: Simulate tool execution
# Note: In a real application, you would:
# 1. Parse the tool call from the model's response
# 2. Execute the actual function (e.g. call a weather API - get_current_\
↪temperature(location='Paris, France', unit='celsius'))
# 3. Get the real result (e.g. 18C)

# Step 4: Feed the result back to the model
# In a real scenario, the tool result would be properly formatted
# Here we simulate the conversation continuing with the information
```

(continues on next page)

(continued from previous page)

```
follow_up = [
    {"role": "user", "content": "The current temperature is 18C."}
]

response = llm.generate_all(follow_up, max_generated_tokens=60, timeout_ms=30000)
print("\nModel response with tool result:")
print(response)

# Step 5: Conversation can continue (tools persist throughout context)
# Ask about historical temperature
historical_question = [
    {"role": "user", "content": "What was the temperature in Paris on 2024-01-15?"}
]

response = llm.generate_all(historical_question, max_generated_tokens=60, timeout_ms=30000)
print("\nUser: What was the temperature in Paris on 2024-01-15?")
print("Model response:")
print(response)

# Note: The conversation continues without re-specifying tools.
#       To use different tools, call llm.clear_context() first.
```

5.8.1. Advanced Context Management

The LLM API provides context management capabilities for monitoring and controlling conversation context. This is essential for:

- **Memory Management:** Monitor context usage to avoid exceeding `max_context_capacity`, which can affect model accuracy
- **Conversation Persistence:** Save and restore conversation states across sessions
- **Multi-Session Applications:** Maintain separate contexts for different users or topics

Key Functions:

- `get_context_usage_size()`: Returns current number of tokens in context
- `max_context_capacity()`: Returns maximum context capacity of the model
- `clear_context()`: Resets context usage to 0 and clears conversation history
- `save_context()`: Saves current context as binary data
- `load_context(context_data)`: Loads a previously saved context

```
[ ]: # Advanced Context Management Example
```

```
# Check initial state
llm.clear_context()
print("Max context capacity: {} tokens".format(llm.max_context_capacity()))
print("Initial context usage: {} tokens".format(llm.get_context_usage_size()))

# Build conversation context
conversation = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Explain machine learning briefly."}
]
response = llm.generate_all(conversation, max_generated_tokens=30)
print("Response:", response)
print("Context usage after generation: {} tokens".format(llm.get_context_usage_size()))
```

(continues on next page)

(continued from previous page)

```
# Save context
saved_context = llm.save_context()
print("Context saved ({} bytes)".format(len(saved_context)))

# Clear context and demonstrate it's empty
llm.clear_context()
print("After clear_context(): {} tokens".format(llm.get_context_usage_size()))

# Load saved context and continue conversation
llm.load_context(saved_context)
print("After load_context(): {} tokens".format(llm.get_context_usage_size()))
continuation = [{"role": "user", "content": "Give me an example."}]
print("Continuation:", llm.generate_all(continuation, max_generated_tokens=20))
```

5.8.2. Generation Parameters Example

seed - used to ensure reproducible results. more configurable parameters can be found in the API documentation.

```
[ ]: test_prompt = [
    {"role": "user", "content": "Tell me about AI."}
]
# Reproducible with seed
llm.clear_context()
response_seed1 = llm.generate_all(test_prompt, seed=42, temperature=0.8, max_
    ↪generated_tokens=25)
llm.clear_context()
response_seed2 = llm.generate_all(test_prompt, seed=42, temperature=0.8, max_
    ↪generated_tokens=25)
print("Seed=42 (run 1): {}".format(response_seed1))
print("Seed=42 (run 2): {}".format(response_seed2))
```

5.8.3. Raw Prompts vs Structured Prompts Example

Raw prompt is a single string, usually wrapped with special tokens that are different per model. Here we demonstrates the tokens for QWEN family. Special tokens and prompt structures can be obtained using 'llm.prompt_template()'

```
[ ]: llm.clear_context()
raw_prompt = "<|im_start|>system\nYou are a helpful assistant.<|im_end|>\n<|im_
    ↪start|>user\nWhat is machine learning?<|im_end|>\n<|im_start|>assistant\n"
with llm.generate(raw_prompt, max_generated_tokens=30, seed=100) as generation:
    print("".join(generation))
print()

llm.clear_context()

# Structured prompt (recommended)
structured_prompt = [
    {"role": "user", "content": "What is machine learning?"}
]
print(llm.generate_all(structured_prompt, max_generated_tokens=30, seed=100))
print()
print(llm.prompt_template())
```

5.8.4. Tokenization Example

The GenAI HEF comes with tokenization information, allowing the encoding of text into tokens.

```
[ ]: test_texts = [
    "Hello world",
    "Machine learning with Hailo",
    "The quick brown fox jumps!"
]

for text in test_texts:
    tokens = llm.tokenize(text)
    print(f'{text} tokens: {len(tokens)}')
```

5.8.5. Stop Tokens and Recovery Sequence Example

The generation will stop when one of these conditions is met:

Max Tokens Reached

- in this case, a custom recovery-sequence will be fed into the LLM
- most models comes with default recovery sequence, which is retrievable and configurable using 'llm.get_generation_recovery_sequence()' and 'llm.set_generation_recovery_sequence()'.
- the recovery-sequence tokens are not counted for 'max_generated_tokens'.

Logical End of Generation

- whenever hitting one of the stop_sequences, the model finishes its generation 'gracefully' (-> without any recovery-sequence).
- a stop-token can be any sequence of tokens (str). the generation will stop when the exact string is generated as a sequence.
- most models comes with default stop tokens, which are retrievable and configurable using 'llm.get_stop_tokens()' and 'llm.set_stop_tokens()'.
- all stop tokens are checked after each generated token. Setting too many can affect performance.

```
[ ]: # Get current stop tokens
original_stop_tokens = llm.get_stop_tokens()
print("Original stop tokens: {}".format(original_stop_tokens))

# Set custom stop tokens
custom_stop_tokens = [".", "END", "\n"]
llm.set_stop_tokens(custom_stop_tokens)
print("Custom stop tokens: {}".format(llm.get_stop_tokens()))
print()

# Set empty stop tokens - model will stop generation only when 'max_generated_tokens' is reached
llm.set_stop_tokens([])
test_prompt = [
    {"role": "user", "content": "Count to 5: 1, 2, 3, 4, 5. Done."}
]
print(llm.generate_all(test_prompt, max_generated_tokens=10))
print()

# Reset stop tokens
llm.set_stop_tokens(original_stop_tokens)
print("Reset stop tokens: {}".format(llm.get_stop_tokens()))
```

5.8.6. Cleanup and Resource Management

Properly clean up resources when done (best practice: use context managers when possible)

```
[ ]: l1m.release()
vdevice.release()
print("Resources released successfully")
print("LLM tutorial completed!")
```

5.9. Python VLM (Vision Language Model) Tutorial

This tutorial demonstrates how to use the VLM Python API for running Vision Language Models on Hailo hardware.

The VLM API provides multi-modal capabilities, combining text, image, and video inputs for comprehensive AI interactions. It supports both streaming and non-streaming text generation, context management, and various generation parameters.

Key Features:

- Multi-modal input processing (text + images + videos)
- Streaming and non-streaming text generation
- Support for multiple images and videos per prompt
- Automatic frame validation and conversion
- Context management for multi-turn conversations

Best Practice: Structured Prompts This tutorial uses **structured prompts** (list of JSON messages) exclusively. Structured prompts provide better control, consistency, and leverage the model's chat template effectively.

Best Practice: Context Manager This tutorial does not use context-manager to share resources between different cells. Make sure to create VDevice and VLM using 'with' statements whenever possible. When not using 'with', use VDevice.release() and VLM.release() to clean up resources.

Requirements

- Run the notebook inside the Python virtual environment: `source hailo_virtualenv/bin/activate`
- A VLM HEF file (Hailo Executable Format for Vision Language Models)
- Image files for testing (JPEG/PNG format)
- OpenCV for image processing: `pip install opencv-python`

Memory Optimization (Optional):

- For large models that may exceed device memory, enable client-side tokenization
- Requires libhailort to be compiled with `HAILO_BUILD_CLIENT_TOKENIZER=ON`
- Requires Rust toolchain (cargo, rustup) to be installed on the build machine
- Set `OPTIMIZE_MEMORY_ON_DEVICE = True` in the configuration section below

Tutorial Structure:

- Basic VLM initialization and image frame requirements
- Image conversion using OpenCV (JPEG/PNG to numpy arrays)
- Text-only generation (VLM without frames)
- Single image processing with structured prompts
- Multiple image processing

- Video processing (single video, combined image and video)
- Generation parameters and context management
- Advanced features: templates, tokenization, stop tokens

When inside the virtualenv, use the command `jupyter-notebook <tutorial-dir>` to open a Jupyter server that contains the tutorials (default folder on GitHub: `hailort/libhailort/bindings/python/platform/hailo_tutorials/notebooks/`).

5.9.1. VLM Tutorial: Setup and Configuration

```
[ ]: import numpy as np
import cv2
import os
from hailo_platform import VDevice
from hailo_platform.genai import VLM

# Configuration - Update these paths for your setup
MODEL_PATH = "/your/hef/path/vlm.hef" # Update this path
SAMPLE_IMAGE_PATH = "/path/to/your/image.jpg" # Update this path
# Memory Optimization: Enable client-side tokenization for large models
# This reduces device memory usage by moving tokenization to the host
# Requires libhailort to be compiled with HAILO_BUILD_CLIENT_TOKENIZER=ON
OPTIMIZE_MEMORY_ON_DEVICE = True # Set to False for device side memory optimization

print("Model path: {}".format(MODEL_PATH))
print("Sample image: {}".format(SAMPLE_IMAGE_PATH))
vdevice = VDevice()
print("Initializing VLM... this may take a moment...")
vlm = VLM(vdevice, MODEL_PATH, OPTIMIZE_MEMORY_ON_DEVICE)
print("VLM initialized successfully!")
```

5.9.2. Understanding VLM Frame Requirements

```
[ ]: # Get frame requirements from the model
frame_shape = vlm.input_frame_shape()
frame_dtype = vlm.input_frame_format_type()
frame_size = vlm.input_frame_size()
frame_order = vlm.input_frame_format_order()

print("Frame requirements:")
print(" Shape: {} (height, width, channels)".format(frame_shape))
print(" Data type: {}".format(frame_dtype))
print(" Size in bytes: {}".format(frame_size))
print(" Format order: {}".format(frame_order))

height, width, channels = frame_shape
print(" Expected frame format: {}x{}x{} {}".format(height, width, channels, frame_
    ↴dtype))
```

5.9.3. Image Conversion Functions Using OpenCV

```
[ ]: def convert_image_to_numpy(image_path, target_shape, target_dtype=np.uint8):
    """
    Convert JPEG/PNG image to numpy array with required shape and dtype.

    Args:
        image_path (str): Path to the image file
        target_shape (tuple): Target shape (height, width, channels)
        target_dtype: Target numpy data type

    Returns:
        numpy.ndarray: Converted image array
    """
    if not os.path.exists(image_path):
        raise FileNotFoundError("Image file not found: {}".format(image_path))

    target_height, target_width, target_channels = target_shape

    # Read image using OpenCV
    img = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if img is None:
        raise ValueError("Failed to load image: {}".format(image_path))

    # Convert BGR to RGB (OpenCV uses BGR by default)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Resize image if needed
    # Note: Using cv2.resize() may not preserve the aspect ratio of the original image.
    # Other types of resizing methods such as letterbox may produce better results.
    if img.shape[0] != target_height or img.shape[1] != target_width:
        img = cv2.resize(img, (target_width, target_height), interpolation=cv2.INTER_LINEAR)
        print("Image resized to: {}x{}".format(target_width, target_height))

    # Handle channel conversion if needed
    if target_channels == 1 and img.shape[2] == 3:
        img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        img = np.expand_dims(img, axis=2)
        print("Converted to grayscale")
    elif target_channels == 3 and len(img.shape) == 2:
        img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
        print("Converted to RGB")

    # Convert to target data type
    img = img.astype(target_dtype)

    print("Final image shape: {}, dtype: {}".format(img.shape, img.dtype))
    return img

def create_sample_image(shape, dtype=np.uint8):
    """
    Create a sample image for testing when no image file is available.

    Args:
        shape (tuple): Image shape (height, width, channels)
        dtype: Data type for the image

    Returns:
        numpy.ndarray: Generated sample image
    """
    height, width, channels = shape
```

(continues on next page)

(continued from previous page)

```
# Create a gradient pattern
img = np.zeros(shape, dtype=dtype)
for i in range(height):
    for j in range(width):
        if channels == 3:
            img[i, j, 0] = int((i / height) * 255) # Red gradient
            img[i, j, 1] = int((j / width) * 255) # Green gradient
            img[i, j, 2] = 128                      # Blue constant
        else:
            img[i, j] = int(((i + j) / (height + width)) * 255)

print("Created sample image with shape: {}, dtype: {}".format(img.shape, img.
→dtype))
return img

# Test the conversion function
print("Testing image conversion functions...")
if os.path.exists(SAMPLE_IMAGE_PATH):
    test_frame = convert_image_to_numpy(SAMPLE_IMAGE_PATH, frame_shape, frame_dtype)
else:
    print("Sample image not found, creating synthetic image...")
    test_frame = create_sample_image(frame_shape, frame_dtype)

print("Test frame ready for VLM processing!")
```

5.9.4. Single Image Processing with Streaming Generation

Important: The number of images in the ‘frames’ list must match the number of image entries in the structured prompt content.

```
[ ]: # Single image structured prompt
single_image_prompt = [
    {"role": "user", "content": [
        {"type": "image"}, # One image placeholder
        {"type": "text", "text": "Describe this image in detail."}
    ]}
]

# Streaming generation with one image
with vlm.generate(prompt=single_image_prompt, frames=[test_frame], max_generated_
→tokens=40) as generation:
    for token in generation:
        print(token, end="", flush=True)
```

5.9.5. Single Image Processing with Non-Streaming Generation

```
[ ]: # Clear context for fresh conversation
vlm.clear_context()

# Non-streaming generation with one image
analysis_prompt = [
    {"role": "user", "content": [
        {"type": "text", "text": "What colors do you see in this image? List the main objects."},
        {"type": "image"} # One image placeholder
    ]}
]

print(vlm.generate_all(prompt=analysis_prompt, frames=[test_frame], max_generated_tokens=40))
```

5.9.6. Text-Only Generation (VLM without Images)

VLM can work without any images by providing an empty frames list. In this case, it behaves like a regular LLM.

```
[ ]: # Text-only structured prompt (no images)
text_only_prompt = [
    {"role": "system", "content": [{"type": "text", "text": "You are a helpful AI assistant."}],
    {"role": "user", "content": [{"type": "text", "text": "Explain what a vision language model is in 2 sentences."}]}
]

# Generate with empty frames list
vlm.clear_context()
print(vlm.generate_all(prompt=text_only_prompt, frames=[], max_generated_tokens=50))
```

5.9.7. Multiple Image Processing

When using multiple images, ensure the frames list contains exactly the same number of images as image placeholders in the prompt.

```
[ ]: # Create a second test image (different pattern)
second_test_frame = create_sample_image(frame_shape, frame_dtype)
# Modify the second image to make it different
second_test_frame = second_test_frame * 0.7 # Darker version
second_test_frame = second_test_frame.astype(frame_dtype)

# Multiple images structured prompt
multi_image_prompt = [
    {"role": "user", "content": [
        {"type": "image"}, # First image placeholder
        {"type": "image"}, # Second image placeholder
        {"type": "text", "text": "Compare these two images. What are the main differences?"}
    ]}
]

# Clear context for fresh conversation
```

(continues on next page)

(continued from previous page)

```
vlm.clear_context()

# Generate with two images - IMPORTANT: frames list must match image count in prompt
with vlm.generate(prompt=multi_image_prompt,
    frames=[test_frame, second_test_frame], # Two frames for two image placeholders
    max_generated_tokens=100) as gen:
    print("".join(gen))
```

5.9.8. Video Processing

VLM supports video input in addition to standalone images. Videos are represented as lists of frames, where each frame is a numpy array matching the model's expected input format. Use {"type": "video"} in structured prompts as a video placeholder.

```
[ ]: # Create a video with 3 frames: original, brighter, even brighter
frame1 = test_frame.copy()
frame2 = np.clip(test_frame.astype(np.float32) * 1.3, 0, 255).astype(frame_dtype)
frame3 = np.clip(test_frame.astype(np.float32) * 1.6, 0, 255).astype(frame_dtype)
sample_video = [frame1, frame2, frame3]

# Video-only prompt: Ask about brightness change
vlm.clear_context()
video_prompt = [{"role": "user", "content": [
    {"type": "video"}, {"type": "text", "text": "In this video, are the frames getting brighter or darker?"}
]}]
print("Video-only generation:")
print(vlm.generate_all(prompt=video_prompt, frames=[], videos=[sample_video], max_
→generated_tokens=30))

# Combined Image + Video prompt
vlm.clear_context()
combined_prompt = [{"role": "user", "content": [
    {"type": "image"}, {"type": "video"}, {"type": "text", "text": "Do the objects in the first image appear somewhere in the video?"}
]}]
print("\nCombined image + video generation:")
print(vlm.generate_all(prompt=combined_prompt, frames=[test_frame], videos=[sample_video], max_
→generated_tokens=30))
```

5.9.9. Multi-Turn Conversations with Images

```
[ ]: # Clear context for fresh conversation
vlm.clear_context()

# Turn 1: Initial image analysis
turn1_prompt = [
    {"role": "system", "content": [{"type": "text", "text": "You are an expert image analyst."}]}, {"role": "user", "content": [
        {"type": "image"}, {"type": "text", "text": "What's the dominant color in this image?"}
    ]}]
```

(continues on next page)

(continued from previous page)

```
]

with vlm.generate(prompt=turn1_prompt, frames=[test_frame], max_generated_
    ↪tokens=50) as gen:
    print("".join(gen))

# Turn 2: Follow-up question (context maintained automatically)
turn2_prompt = [
    {"role": "user", "content": [{"type": "text", "text": "Can you suggest what type of red
    ↪scene this might be?"}]}
]

print(vlm.generate_all(prompt=turn2_prompt, frames=[], max_generated_tokens=50))
```

5.9.10. Advanced Context Management

Context is maintained between generate() calls automatically. The VLM API provides context management capabilities for monitoring and controlling conversation context, including image context. This is essential for:

- **Memory Management:** Monitor context usage to avoid exceeding `max_context_capacity`, which can affect model accuracy
- **Conversation Persistence:** Save and restore conversation states across sessions (including image context)
- **Multi-Session Applications:** Maintain separate contexts for different users or topics

Key Functions:

- `get_context_usage_size()`: Returns current number of tokens in context
- `max_context_capacity()`: Returns maximum context capacity of the model
- `clear_context()`: Resets context usage to 0 and clears conversation history
- `save_context()`: Saves current context as binary data (including image context)
- `load_context(context_data)`: Loads a previously saved context

```
[ ]: # Context Management Example

# Check initial state
vlm.clear_context()
print("Max context capacity: {} tokens".format(vlm.max_context_capacity()))
print("Initial context usage: {} tokens".format(vlm.get_context_usage_size()))

conversation = [
    {"role": "user", "content": [
        {"type": "image"},
        {"type": "text", "text": "What do you see in this image?"}
    ]}
]
response = vlm.generate_all(conversation, frames=[test_frame], max_generated_
    ↪tokens=30)
print("Response:", response)
print("Context usage after generation: {} tokens".format(vlm.get_context_usage_
    ↪size()))

# Save context (including image context)
saved_context = vlm.save_context()
print("Context saved ({} bytes)".format(len(saved_context)))

# Clear context and demonstrate it's empty
```

(continues on next page)

(continued from previous page)

```
vlm.clear_context()
print("After clear_context(): {} tokens".format(vlm.get_context_usage_size()))

# Load saved context and continue conversation
vlm.load_context(saved_context)
print("After load_context(): {} tokens".format(vlm.get_context_usage_size()))
continuation = [{"role": "user", "content": "Describe the colors."}]
print("Continuation:", vlm.generate_all(continuation, frames=[], max_generated_
→tokens=20))
```

5.9.11. Generation Parameters Example

Temperature - used to control the model's creativity. More configurable parameters can be found in the API documentation.

```
[ ]: test_prompt = [
    {"role": "user", "content": [
        {"type": "image"},
        {"type": "text", "text": "Describe this image creatively."}
    ]}
]

# Test different temperature settings

# Low temperature (more deterministic)
vlm.clear_context()
response_low = vlm.generate_all(
    prompt=test_prompt,
    frames=[test_frame],
    temperature=0.1,
    seed=42,
    max_generated_tokens=40
)
print("Low temperature (0.1): {}".format(response_low))

# High temperature (more creative)
vlm.clear_context()
response_high = vlm.generate_all(
    prompt=test_prompt,
    frames=[test_frame],
    temperature=0.9,
    seed=42,
    max_generated_tokens=40
)
print("High temperature (0.9): {}".format(response_high))
```

5.9.12. Raw Prompts vs Structured Prompts Example

VLM supports both structured prompts (recommended) and raw text prompts with special tokens. Here we demonstrate the tokens for QWEN family. Special tokens and prompts structures can be obtained using 'vlm.prompt_template()'

```
[ ]: vlm.clear_context()

# Raw prompt with vision tokens (model-specific format)
raw_prompt = "<|im_start|>system\nYou are a helpful assistant.<|im_end|>\n<|im_
→start|>user\nDescribe this image<|vision_start|><|image_pad|><|vision_end|>
→<|im_end|>\n<|im_start|>assistant\n"

print(vlm.generate_all(prompt=raw_prompt, frames=[test_frame], max_generated_
→tokens=40, seed=100))
print()

vlm.clear_context()

# Structured prompt (recommended)
structured_prompt = [
    {"role": "user", "content": [
        {"type": "text", "text": "Describe this image"}, 
        {"type": "image"} 
    ]} 
]

print(vlm.generate_all(prompt=structured_prompt, frames=[test_frame], max_
→generated_tokens=40, seed=100))
print()

print("Model prompt template:")
print(vlm.prompt_template())
```

5.9.13. Tokenization Example

```
[ ]: test_texts = [
    "Describe this image",
    "Vision language model with Hailo",
    "What colors do you see in this image?"
]

print("Tokenization examples:")
for text in test_texts:
    tokens = vlm.tokenize(text)
    print("{} -> {} tokens: {}".format(text, len(tokens), tokens))

print()
```

5.9.14. Stop Tokens and Recovery Sequence Example

```
[ ]: # Get current stop tokens
original_stop_tokens = vlm.get_stop_tokens()
print("Original stop tokens: {}".format(original_stop_tokens))

# Test with custom stop tokens
custom_stop_tokens = [".", "END"]
vlm.set_stop_tokens(custom_stop_tokens)
print("Custom stop tokens: {}".format(vlm.get_stop_tokens()))

test_prompt = [
    {"role": "user", "content": [
        {"type": "image"}, {"type": "text", "text": "List three things you see. 1. Color patterns 2. Shapes 3. Textures."}
    ]}
]

vlm.clear_context()
response = vlm.generate_all(prompt=test_prompt, frames=[test_frame], max_generated_tokens=50)
print("Response with custom stop tokens: {}".format(response))

# Reset stop tokens
vlm.set_stop_tokens(original_stop_tokens)
print("Reset stop tokens: {}".format(vlm.get_stop_tokens()))
```

5.9.15. Summary and Best Practices

Key Points to Remember:

1. **Frame Count Matching:** Always ensure the number of frames matches the number of image placeholders in your structured prompt
2. **Video Count Matching:** Similarly, ensure the number of videos matches the number of video placeholders
3. **Image Format:** Use `vlm.input_frame_shape()`, `vlm.input_frame_format_type()` to get correct format requirements
4. **OpenCV Conversion:** Use BGR→RGB conversion when loading images with OpenCV
5. **Text-Only Mode:** VLM can work without images by using empty frames list []
6. **Video Input:** Videos are lists of frames; each frame must match the model's expected input format
7. **Context Management:** Use `vlm.clear_context()` to start fresh conversations. Use `'vlm.get_context_usage_size()'` to see the number of tokens used in this context
8. **Resource Cleanup:** Always call `vlm.release()` and `vdevice.release()` when done

Structured Prompt Format for VLM:

```
prompt = [
    {"role": "user", "content": [
        {"type": "text", "text": "Your text here"}, {"type": "image"}, {"type": "video"}, {"type": "text", "text": "More text if needed"}
    ]}
]
```

5.9.16. Cleanup and Resource Management

Properly clean up resources when done (best practice: use context managers when possible)

```
[ ]: vlm.release()
vdevice.release()
print("Resources released successfully")
print("VLM tutorial completed!")
```

5.10. Python Speech2Text Tutorial

This tutorial demonstrates how to use the Speech2Text Python API for running Speech-to-Text models on Hailo hardware.

The Speech2Text API provides audio transcription capabilities using Whisper-based models, supporting both transcription and translation tasks with configurable language settings.

Key Features:

- Audio transcription with timestamped segments
- Language translation capabilities
- Support for multiple output formats (segments vs complete text)
- Configurable task types (transcribe/translate)
- Language-specific processing

Best Practice: Context Manager This tutorial does not use context-manager to share resources between different cells. Make sure to create VDevice and Speech2Text using ‘with’ statements whenever possible. When not using ‘with’, use VDevice.release() and Speech2Text.release() to clean up resources.

Requirements:

- Run the notebook inside the Python virtual environment: `source hailo_virtualenv/bin/activate`
- A Speech2Text HEF file (Hailo Executable Format for Speech-to-Text models)
- Audio files in PCM float32 format (normalized to [-1.0, 1.0], mono, little-endian, 16 kHz)
- NumPy for audio processing: `pip install numpy`

Audio Format Requirements:

The audio input must be in a specific format for proper processing:

- **Format:** PCM float32 normalized to [-1.0, 1.0]
- **Channels:** Mono (single channel)
- **Endianness:** Little-endian
- **Sample Rate:** 16 kHz

Tutorial Structure:

- Basic Speech2Text initialization and audio loading
- Transcription with timestamped segments
- Complete text transcription
- Language translation capabilities
- Task configuration (transcribe vs translate)

- Language-specific processing

When inside the virtualenv, use the command `jupyter-notebook <tutorial-dir>` to open a Jupyter server that contains the tutorials (default folder on GitHub: `hailort/libhailort/bindings/python/platform/hailo_tutorials/notebooks/`).

```
[ ]: # Speech2Text Tutorial: Setup and Configuration

from hailo_platform import VDevice
from hailo_platform.genai import Speech2Text, Speech2TextTask
import numpy as np
import os

# Configuration - Update these paths for your setup
MODEL_PATH = "/your/hef/path/speech2text.hef" # Update this path
AUDIO_FILE_PATH = "/your/audio/file/path/audio.bin" # Update this path

print("Model path: {}".format(MODEL_PATH))
print("Audio file path: {}".format(AUDIO_FILE_PATH))

vdevice = VDevice()
print("Initializing Speech2Text... this may take a moment...")
speech2text = Speech2Text(vdevice, MODEL_PATH)
print("Speech2Text initialized successfully!")
```

5.10.1. Audio Loading and Format Validation

Load and validate audio data in the required format.

```
[ ]: def load_audio_file(file_path):
    """
    Load audio file as binary data, and convert it into numpy array.
    Expected format: PCM float32 normalized to [-1.0, 1.0], mono, little-endian, 16 kHz
    """
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"Audio file not found: {file_path}")

    with open(file_path, 'rb') as f:
        audio_data = f.read()

    print(f"Loaded audio file: {file_path}")
    print(f"Audio data size: {len(audio_data)} bytes")

    audio_array = np.frombuffer(audio_data, dtype='<f4').copy()
    return audio_array

# Load audio file
audio_data = load_audio_file(AUDIO_FILE_PATH)
```

5.10.2. Basic Transcription with Segments

Generate transcription with timestamped segments for detailed analysis.

```
[ ]: task = Speech2TextTask.TRANSCRIBE # or "TRANSLATE"
language = "en" # ISO-639-1 language code

print("Generating transcription with segments...")
segments = speech2text.generate_all_segments(audio_data, task=task, ↴
language=language)

print("\nTranscription Results:")
print("=" * 50)
for i, segment in enumerate(segments):
    print(f"Segment {i+1}: {segment}")
    print("-" * 30)
```

5.10.3. Complete Text Transcription

Generate complete transcription as a single string without timestamps.

```
[ ]: # Generate complete transcription
print("Generating complete transcription...")
complete_text = speech2text.generate_all_text(audio_data, task=task, ↴
language=language)

print("\nComplete Transcription:")
print("=" * 50)
print(complete_text)
```

5.10.4. Language Translation Example

Demonstrate translation capabilities by transcribing speech in one language and translating to another.

```
[ ]: task = Speech2TextTask.TRANSLATE # Translate to English
language = "es" # Source language

print("Generating translation...")
translation_segments = speech2text.generate_all_segments(audio_data, task=task, ↴
language=language)

print("\nTranslation Results:")
print("=" * 50)
for i, segment in enumerate(translation_segments):
    print(f"Segment {i+1}: [{segment.start_sec:.2f}s - {segment.end_sec:.2f}s]")
    print(f"Translation: {segment.text}")
    print("-" * 30)
```

5.10.5. Language Detection Example

Demonstrates automatic language detection. Instead of specifying a language, the model detects it from the audio input, allowing seamless transcription or translation of audio in an unknown language.

```
[ ]: # Automatically detect language by omitting the 'language' parameter
print("Generating transcription with automatic language detection...")
auto_detected_segments = speech2text.generate_all_segments(audio_data, task=Speech2TextTask.TRANSCRIBE)

print("\nAuto-Detected Language Transcription Results:")
print("=" * 50)
for i, segment in enumerate(auto_detected_segments):
    print(f"Segment {i+1}: [{segment.start_sec:.2f}s - {segment.end_sec:.2f}s]")
    print(f"Text: {segment.text}")
    print("-" * 30)
```

5.10.6. Repetition penalty example

This section demonstrates the effect of applying a repetition penalty during speech-to-text generation. By adjusting the `repetition_penalty` parameter, you can control how much the model discourages repeated tokens. Higher values penalize repetition more strongly.

```
[ ]: print("Generating transcription with repetition penalty...")
complete_text_with_penalty = speech2text.generate_all_text(audio_data, task=task, language=language, repetition_penalty=1.2)

print("\nComplete Transcription with repetition penalty:")
print("=" * 50)
print(complete_text_with_penalty)
```

5.10.7. Tokenization Example

The GenAI HEF comes with tokenization information, allowing the encoding of text into tokens.

```
[ ]: tokens = speech2text.tokenize(complete_text)
print("The transcription has {} tokens: {}".format(len(tokens), tokens))
```

5.10.8. Cleanup and Resource Management

Properly clean up resources when done (best practice: use context managers when possible)

```
[ ]: # Clean up resources
print("Cleaning up resources...")
speech2text.release()
vdevice.release()
print("Resources cleaned up successfully!")
```

6. Running Inference

This section describes several use cases of the inference process. Inference refers to the stage where the host sends input data to the Hailo device (specifically to its NN Core) and receives the processed output.

The **InferModel API** is the new and recommended interface for running inference on Hailo devices. For the older vStream API (supported only on the Hailo-15 family), see [VStream API](#).

The InferModel API is designed for flexibility and ease of integration. It supports both **synchronous** and **asynchronous** inference modes. The **asynchronous (async)** API is optimized for performance and allows the inference of multiple frames without needing multiple threads, simplifying development while improving throughput.

Supported in both C++ and Python, the InferModel API enables zero-copy pipelines where possible, enhancing performance and optimizing memory usage.

See also:

The [C++ GenAI inference tutorial](#) demonstrates how to run inference in C++ on Generative AI models.

6.1. Inference Stages

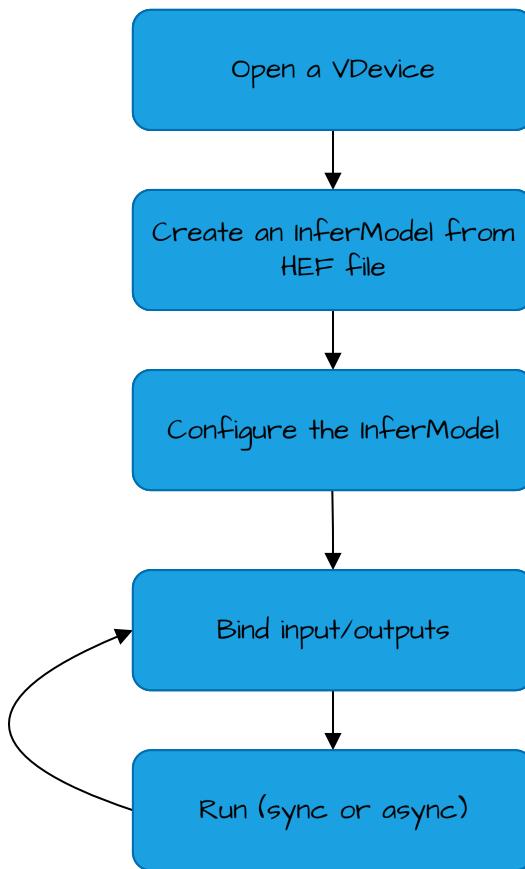


Figure 2. Illustration of an inference flow with the Hailo device.

6.1.1. Preparation

Open a VDevice

The first stage of the inference process is to open a **VDevice** ([C++](#), [Python](#)). A **VDevice** is a virtual object that represents a Hailo device.

The **VDevice** provides virtualization, allowing multiple models to run concurrently on the same physical device (see [model scheduler](#)).

The `hailo_vdevice_params_t` ([C++](#), [Python](#)) is an optional parameter that can be used to configure the **VDevice**. It allows you to specify various options, such as the device id to open, the group id or scheduling algorithm.

Successfully opening a **VDevice** confirms that the device is available and ready for inference.

Create an InferModel from a HEF file

Once the **VDevice** is opened, the next step is to call `VDevice::create_infer_model` ([C++](#), [Python](#)) to create an **InferModel** ([C++](#), [Python](#)) from a HEF file.

The HEF (Hailo Executable Format) file is a compiled model produced by the Hailo Dataflow Compiler (DFC).

Creating the **InferModel** validates that the HEF is compatible with the selected device.

Configure the InferModel

An **InferModel** instance holds all necessary information for configuring a model for inference, including methods to set and retrieve model parameters.

A stream is either input or output layer of the model. Each stream has `shape` ([C++](#), [Python](#)), `format_type` ([C++](#), [Python](#)) and `format_order` ([C++](#), [Python](#)). The `shape`, `format_type` and `format_order` as seen by the application side (host side) may be different from the `shape`, `format_type` and `format_order` as seen by the device side. On this case, a transformation operation will be added to the pipeline. Use the [Parse-Hef](#) tool to print the model's streams definitions.

To access a specific stream, call `InferModel::input(<name>)` ([C++](#), [Python](#)) or `InferModel::output(<name>)` ([C++](#), [Python](#)), which return an `InferModel::InferStream` ([C++](#), [Python](#)) object for querying and modifying parameters.

Once all required parameters are set, configure the model by calling `InferModel::configure` ([C++](#), [Python](#)). The `ConfiguredInferModel` ([C++](#), [Python](#)) returned represents a ready-to-run model.

6.1.2. Sending and Receiving Data

To begin running inference, input and output buffers must be prepared.

Create a `Bindings` instance ([C++](#), [Python](#)) and link it to the appropriate input and output buffers.

Since the API is optimized for zero-copy performance, the buffers must be aligned to the system page size (typically 4KB). It is recommended to allocate these buffers using system primitives such as `mmap` (on Linux) or `VirtualAlloc` (on Windows).

Synchronous Inference

Designed for simplicity, synchronous inference is performed by calling `ConfiguredInferModel::run()` ([C++](#), [Python](#)).

This method sends the input buffers to the device and fills the output buffers with the results, all within a single blocking call.

Asynchronous Inference

The asynchronous API provides better utilization of the NN Core and improved performance, as it allows multiple frames to be processed in parallel.

To perform asynchronous inference, use `ConfiguredInferModel::run_async()` ([C++](#), [Python](#)). This returns an `AsyncInferJob` ([C++](#), [Python](#)) that represents the running job.

An optional callback can be provided and will be called once inference completes. Alternatively, the job can be waited on manually.

In C++, the lifetime of the job is managed by the `AsyncInferJob` object. When the object is destroyed, it blocks until the job finishes — unless `AsyncInferJob::detach()` ([C++](#)) is called.

Important: Until the async job completes, the input and output buffers in the `Bindings` instance **must not** be freed or reused. Releasing the buffers before the job completes will result in undefined behavior.

6.1.3. C++ InferModel API

For tutorials, see [basic Async inference](#), [advanced Async inference](#) and [Async inference using DMA buffers](#).

For documentation - see [C++ InferModel API documentation](#).

6.1.4. Python InferModel API

For tutorials, see [Python single model inference tutorial](#) and [Python multi models inference tutorial](#).

For further documentation, refer the the class `InferModel` and below, under the [Python API documentation](#).

6.2. Data Formats

In this section we discuss the data formats used by the inference process. The data format is selected by the user to be used for input/output to/from HailoRT. This section covers popular data formats supported by HailoRT (for the full list of formats - see `hailo_format_order_t`). Input and output tensors are represented as `numpy` arrays in Python and as raw buffers in C/C++.

6.2.1. NHWC

The NHWC format is a common data format used in deep learning, for example, by [Tensorflow](#). It stands for **N** number of elements in the batch, with shape **H**eight, **W**idth and **C**hannels.

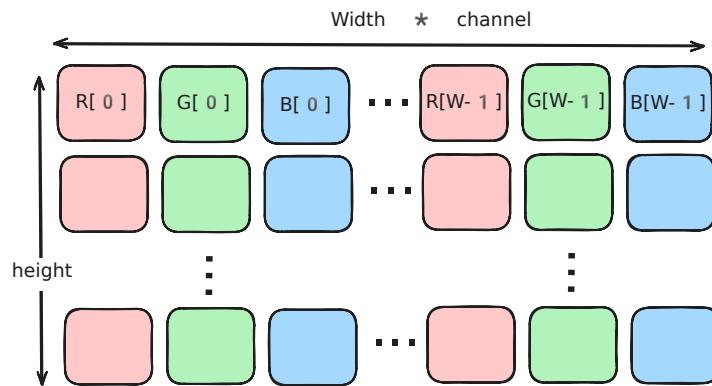


Figure 3. Data format NHWC

6.2.2. NCHW

The NCHW format is a common data format used in deep learning, for example, by [Pytorch](#). It stands for **N**umber of elements in the batch, with shape **C**hannels, **H**eight and **W**idth.

Note: It is not recommended to use this format with the Hailo hardware, as it requires a format conversion operation - so either NHWC or NHCW are preferred.

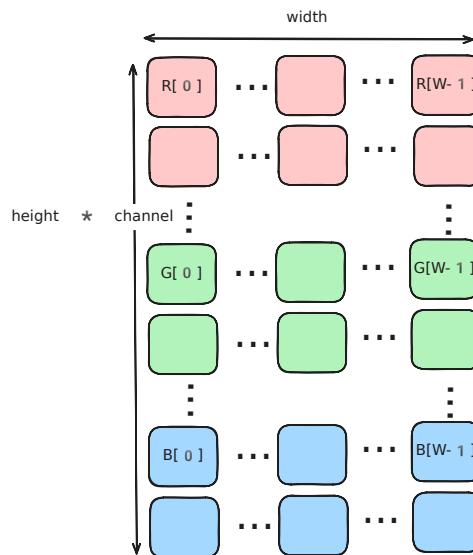


Figure 4. Data format NCHW

6.2.3. NHCW

Note: NHCW format is the default format used by the Hailo hardware.

The NHCW format is the default format used by the Hailo hardware (some exceptions include NMS, Argmax, or other special operations). Therefore, using any other format would necessitate a format conversion operation. The format conversion can be done either by HailoRT or by the NN Core, and therefore, can affect either the host or the NN Core utilization and the overall throughput. Additional format conversion may also be required to pad the data to 8 bytes. To verify which format conversions are required by the specific compiled model you are using, please refer to the [Parse-HEF](#) tool.

Note: Changing the conversions inside the NN Core requires a new compilation by the Hailo Dataflow Compiler.

The order of this format is **N** number of elements in the batch, with shape **Height**, **Channels** and **Width**.

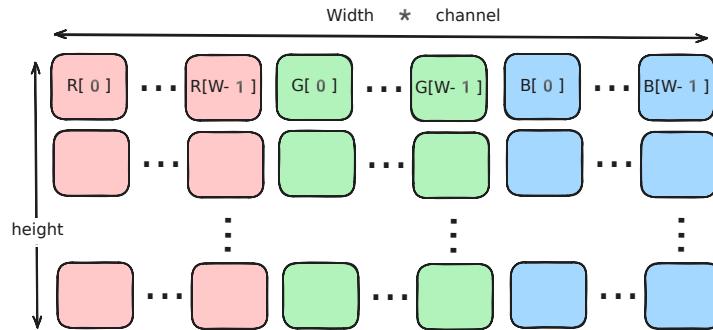


Figure 5. Data format NHCW

6.2.4. NV12

The NV12 format is a common format used in video processing. It is a planar format with the **Y** plane followed by the **UV** plane as described in the following figure. The two planes can be used in a single buffer or in two separate buffers for each plane.

Note: This format, with two separated planes, is used by default by Hailo-15 vision pipeline - and it is recommended by Hailo to use it for optimized performance.

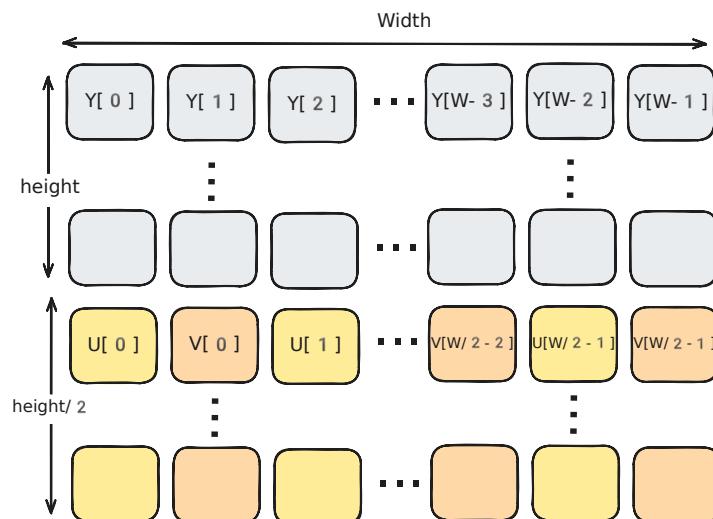


Figure 6. Data format NV12

6.2.5. NV21

The NV21 format is a common format used in video processing. It is a planar format with the **Y** plane followed by the **VU** plane as described in the following figure.

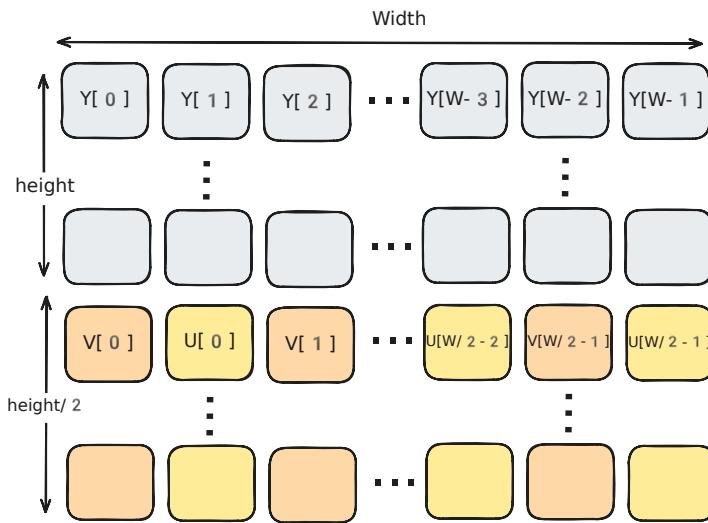


Figure 7. Data format NV21

6.2.6. YUY2

The YUY2 format is a common format used in video processing. It is a packed format with the **Y** plane followed by the **U** and **V** planes as described in the following figure.

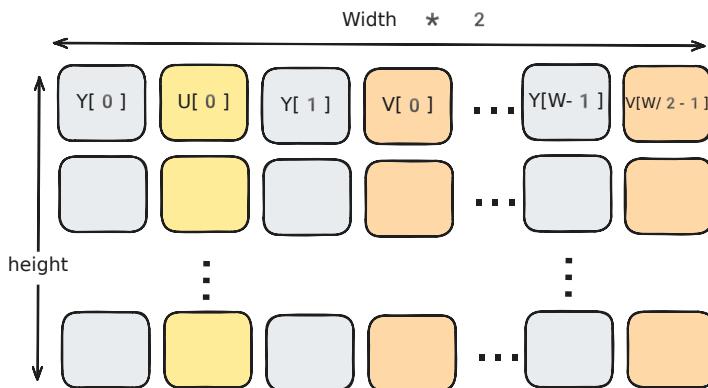


Figure 8. Data format YUY2

6.2.7. NMS_BY_SCORE

The NMS_BY_SCORE format is used when Non-Maximum Suppression (NMS) operation is performed. Data is arranged so that the first element is the number of detections, followed by the detections themselves ordered by their score. Each detection is represented by 6 elements: y_{min} , x_{min} , y_{max} , x_{max} , $class_id$, $score$.

6.2.8. NMS_BY_CLASS

The NMS_BY_CLASS format is used when Non-Maximum Suppression (NMS) operation is performed. Data is arranged so that for each class, the first element is the number of detections for that class, followed by the detections themselves ordered by their score. Each detection is represented by 5 elements: y_min, x_min, y_max, x_max and score.

6.3. Environment Variables

The HailoRT environment variables are used to enable/disable different features.

Environment Variable name	Default Value	Description
HAILORT_LOGGER_PATH	""	<ul style="list-style-type: none"> Not setting this variable, or setting it to an empty string, will set the log file path to the current directory Setting the value to NONE will disable the log file
HAILORT_CONSOLE_LOGGER_LEVEL	"warning"	<ul style="list-style-type: none"> Controlling the message level that will be prompted to the console Possible values: "info", "warning", "error" and "critical"
HAILO_MONITOR	0	see Monitor
HAILO_MONITOR_TIME_INTERVAL	""	see Monitor
HAILO_TRACE	""	see HailoRT profiler
HAILO_TRACE_PATH	0	see HailoRT profiler
HAILO_TRACE_TIME_IN_SECONDS_BOUNDED_DUMP	0	see HailoRT profiler
HAILO_TRACE_SIZE_IN_KB_BOUNDED_DUMP	0	see HailoRT profiler

6.4. Model Scheduler and Compiling Models Together

HailoRT provides several mechanisms that enable the running of multiple models:

- The **Network groups switching using HailoRT Model Scheduler** – The Model scheduler is a HailoRT component that comes to enhance and simplify the usage of the same Hailo device by multiple networks. The responsibility for activating/deactivating the network groups is now under HailoRT, and done **automatically** without user application intervention. In order to use the Model Scheduler, create the VDevice with scheduler enabled, configure all models to the device, and let HailoRT do the rest.
- The manual **user controlled switching** mechanism which allows the user to switch between several models manually. In order to achieve this, first both models are configured and the first one is activated. When the switch should take place, the first model is de-activated and the second model is activated.

The following table compares between these two methods:

	Automatic Model Scheduler	Manual Switching Between Models
Purpose	For most multiple models scenarios.	When the app needs full (low-level) control on when to run each model.
How to compile the model?	No special compilation flow. Any HEF can be used.	No special compilation flow. Any HEF can be used.
How to deploy the model?	Create VDevice with the scheduler enabled. Then the switching will be done automatically behind the scenes.	Use the activate() and deactivate() APIs to switch models.

- The **compiling models together** mechanism allows you to run multiple models by **automatically** switching between models that were compiled together using the [DFC join\(\) API](#), which constitute the full model.

6.5. Model Scheduler

The Model Scheduler is a HailoRT component that enhances and simplifies the usage of the same Hailo device by multiple networks. The responsibility for activating/deactivating the network groups is handled **automatically** without user application intervention. The **Model Scheduler**'s behavior and performance is controlled using the following parameters:

- Configured network group **batch-size** affects the size of the networks group's input and output queues. The larger the batch-size parameter, the larger the queues. For multi context models the actual infer batch will be up to the batch-size.

To modify this parameter, use [InferModel::set_batch_size \(C++, Python\)](#).

- Model scheduler **Scheduling Scheme** determines how the next network to be activated is selected. Currently, the only scheme is Round-Robin, which maintains fairness across different networks so each network in turn will be checked to be identified as ready for activation

- Model Scheduler **threshold** is the minimum required number of inference requests, before the network group is eligible to be scheduled (unless the timeout has passed). Default is 0. Once the scheduler switches to the new model, it will infer all available frames in its queues, even if they are more than the threshold.

To modify this parameter, use [ConfiguredInferModel::set_scheduler_threshold \(C++, Python\)](#).

- Model Scheduler **timeout** is the maximum time period [ms] that may pass before a configured network group is scheduled. This counter is reset when the network group is activated, and at least one frame is sent to it. Default is 0ms.

To modify this parameter, use [ConfiguredInferModel::set_scheduler_timeout \(C++, Python\)](#).

- Model Scheduler **priority** determines the relative importance of the network. When the scheduler comes to select the next network to run it first looks at the priority of the network. That means that, the scheduler will first evaluate all the networks that have the same highest priority, and if none can be selected then it will move to consider lower priority networks. By default, the priority level is set to HAILO_SCHEDULER_PRIORITY_NORMAL.

To modify this parameter, use [ConfiguredInferModel::set_scheduler_priority \(C++, Python\)](#).

Note: The Model Scheduler default behavior is using round-robin scheduling scheme with default values of threshold (0) and timeout (0ms). These defaults will schedule each network as soon as it receives a single frame.

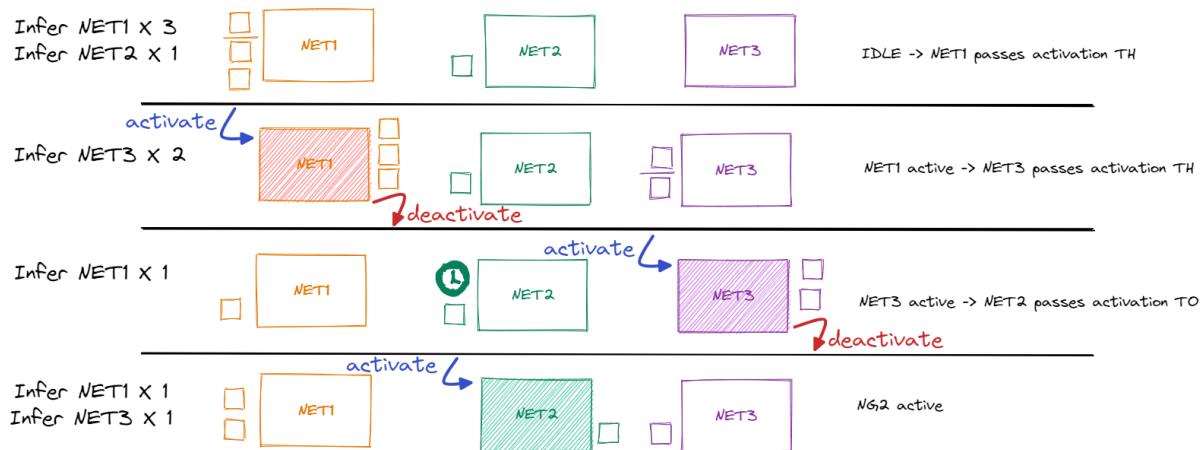


Figure 9. Model Scheduler Example

6.6. Model Scheduler Optimizations

When examining the scheduler behavior and performance we always need to keep in mind that it is all about tradeoffs: **Memory vs Performance** and **FPS vs Latency**.

The scheduler monitors the fullness of each network's low-level input and output queues (these queues are the buffer queues that reside just before and after the HW) and determine if the associated network needs and can be activated. The scheduler also monitors how much time has passed from the last time it was activated. The network is considered ready for activation if one of its activation criteria is reached. The network is considered able for activation if it has enough free space in its output queues.

Batch-size: The first step that will improve performance and make the other configurations more effective is to make the input and output queues larger by increasing the batch-size. Large queues improve performance by performing any of the following steps:

1. Pre and post processing can run in parallel to the actual HW inference.
2. Pre and post processing can run while the network is deactivated.
3. Once the network is activated more frames are ready to be inferred immediately, so the inference is more effective.
4. Large queues can flatten jitters in FPS rate.

The tradeoff of having larger queues is:

1. HailoRT consumes more host memory (in particular DMA-able memory which may be limited in some platforms).
2. Increase the overall latency.

The following parameters are activation parameters that take effect when the network needs to be activated:

Threshold (TH): The TH controls the number of frames that will be accumulated inside the input queue before the network should be activated. Since the TH is measured against the queue's fullness then the larger the input queue the more configuration options we have.

The TH is affecting the performance by:

1. A network with a lower TH will be activated more often.
2. The higher the TH the more frames will be accumulated so each time the network is activated the inference will be more effective.
3. The higher the TH the more frames will be accumulated so the latency will be higher.

Timeout (TO): The TO controls the maximum time from the last time the network was activated. Even if there are not enough frames in the input queue (a minimum of 1 frame is required) but enough time has passed the network will be activated in order to limit the latency, which may increase as a result of high TH.

Priority: The priority reflects the relative importance of each network. Networks with a higher priority will be given priority during selection, resulting in better performance and longer running time for these networks.

The priority is affecting the performance by:

1. Network with higher priority will get more running time.
2. Giving higher priority will enhance both the latency and the FPS rates.
3. Increasing the priority of a network will impact the performance of other networks.
4. When the priority of a network is increased, it may cause starvation to other networks if the higher-priority networks are always ready to run (reach their TH or TO).

The general rule of thumb should be:

1. For multi model scheduling we can set the TH to aggregate more frames before the network is scheduled, doing this will make the inference more effective but will raise the overall latency.
2. Using the TH we can perform “soft” prioritization, a smaller TH value will cause the network to be scheduled more often.
3. To limit the latency of the network inference the TO parameter can be set to ensure that a network will be scheduled for running if it has at least a single frame in its input queue and enough time has passed from its previous activation.
4. To enable the schedule to be more effective, we need to make the input and output queues larger so more frames can be accumulated and thus we can parallel the host’s pre and post processing to the actual inference. The tradeoff of this is consuming more memory.

6.7. Stream Multiplexer

The Model Scheduler enables **Stream Multiplexer** which automatically aggregates different streams of the same model, and saves the model’s activation/deactivation times.

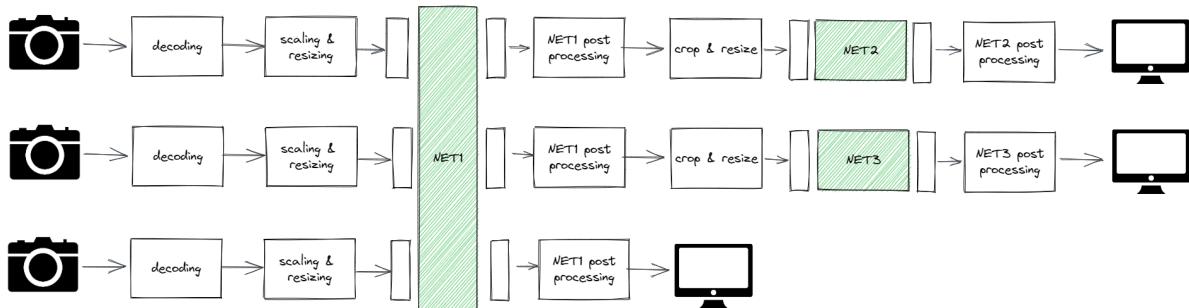


Figure 10. Stream Multiplexer

6.8. Multi-Process Service

On Hailo-15 devices, the **Multi-Process Service** enables the ability to manage and share a Hailo device between multiple processes, thus providing the ability to use multi-process inference.

Note: Multi-process inference is enabled by default on Hailo-10H devices, so this service is not required, and it is likewise unnecessary to use the corresponding flags when working with the various HailoRT APIs.

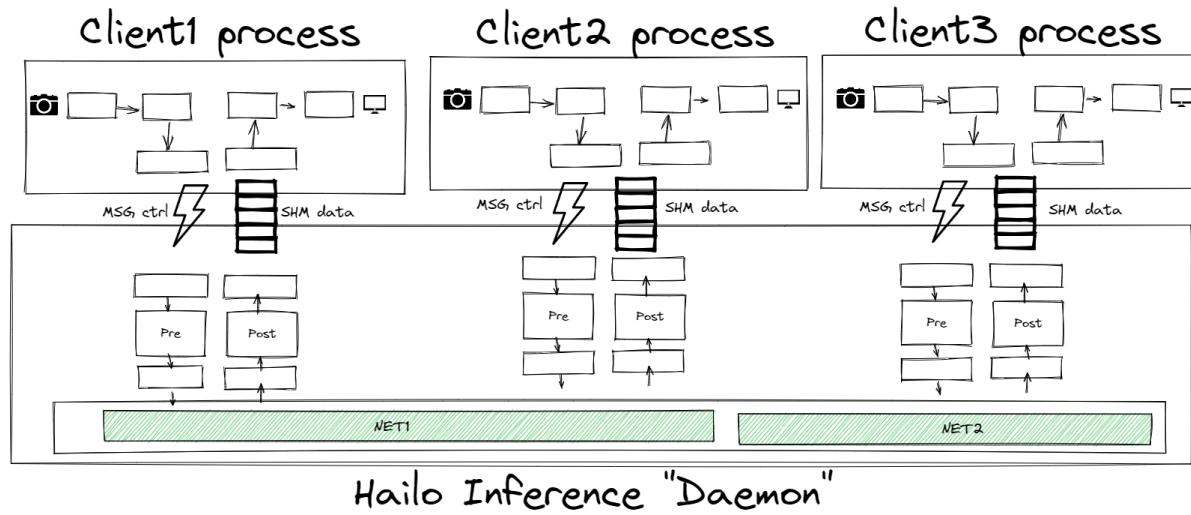


Figure 11. Multi-Process Service

6.8.1. Usage

- To create a shared VDevice with the service, use the same `hailo_vdevice_params_t::group_id` and enable `hailo_vdevice_params_t::multi_process_service`
- When using `hailortcli` run for inference with multi-process service, use the `--multi-process-service` flag
- When using HailoNet GStreamer element, use the `multi-process-service` property
- On Hailo-10H devices, multi-process inference is supported by default, so there is no need to enable any flag.
- In addition to the client's `hailort` log, the service writes to syslog.

Enable Service

For Hailo-15 devices, multi-process service is enabled by default. In order to disable it, run:

```
sudo systemctl disable hailort_server.service
```

The service can be enabled again by running (`--now` starts the service):

```
sudo systemctl enable --now hailort_server.service
```

6.9. VStream API

The vStream API is the old API used to run inference, and it is not recommended for new designs. It is supported only on Hailo-15 family devices.

7. Generative-AI

The HailoRT C++ API provides an efficient method to run GenAI models on Hailo devices. The following section covers the API that allows the loading of pre-compiled HEF files and running complete GenAI inference pipelines, such as LLMs, VLMs, Speech-to-Text or Text-to-Image models.

7.1. LLM / VLM C++ API

The LLM API includes the following key classes:

- `hailort::genai::LLM`: Main class that encapsulates `LLMParams` and instantiates new LLM models for inference on a specified `VDevice`. The `LLM` class controls the lifecycle of the model and must be maintained for any LLM interaction.
- `hailort::genai::LLMParams`: Responsible for configuring the LLM model at initialization. These parameters remain fixed during runtime and allow users to specify critical settings such as the HEF path and selection of LoRA adapter.
- `hailort::genai::LLMGenerator`: Created from the `LLM` class to facilitate model interaction. Using this class, users submit prompts and generate responses from the model. This class serves as the main interface provided to the user to communicate with the model.
- `hailort::genai::LLMGeneratorParams`: Allows runtime modification of generation parameters. These include sampling parameters such as temperature, frequency penalty, and KV-cache handling parameters. Initialized by the `LLM` with default values, these parameters can be modified at runtime to control the generation process.
- `hailort::genai::LLMGeneratorCompletion`: Represents the generation itself. This instance allows users to read the next token-completion, and to check the generation-status - holds the reason for the end of the generation.

Note: Although the creation of `LLMGenerator` is lightweight and does not require significant resources, `LLMGenerator` can be re-used for multiple generations. It is recommended to create a new `LLMGenerator` only when different generator-parameters are to be used.

Note: Only one `LLMGenerator` instance should be active at any time to ensure accurate parameter handling. Ensure the previous `LLMGenerator` is fully destructed before creating a new one.

Note: If the `LLMGenerator` is destructed before `hailort::genai::VLMGenerator::generate()` is called, all inputs written to it will be discarded.

Note: Only one `LLMGeneratorCompletion` instance should exist at a time. Creating multiple instances concurrently may lead to undefined behavior. Ensure the previous `LLMGeneratorCompletion` is fully destructed before creating a new one.

The workflow progresses from `VDevice` and `LLMParams` initialization, through `LLM` object construction, to `LLMGeneratorParams` configuration. The final steps involve Generator instantiation and utilization for prompt input (write API) and response generation (generate API). For the full example, please refer to [HailoRT C++ chat tutorial](#).

Note: The `hailort::genai::VLM` API follows the same structure as the LLM API, with the exception of ability to send images to the device using the `hailort::genai::VLMGenerator::generate()` API. More information can be found in the [HailoRT C++ VLM tutorial](#).

7.2. Speech-to-Text C++ API

The Speech-to-Text API includes the following core classes:

- * `hailort::genai::Speech2Text`: Main class that encapsulates `Speech2TextParams` and instantiates a new Speech-to-Text model for inference on a specified `VDevice`.
- * `hailort::genai::Speech2TextParams`: Responsible for configuring the Speech-to-Text model at initialization. These parameters remain fixed during runtime and allow users to specify critical settings, such as the HEF path.
- * `hailort::genai::Speech2TextGeneratorParams`: Enables runtime configuration of generation parameters that can be modified during execution. These include, settings such as the task type to perform (transcription or translation), the target language to use for transcription or translation (which can be automatically detected by the model if not specified), and the repetition penalty to use for the generation process.

Note: The format of the expected audio buffer is PCM float32 normalized to [-1.0, 1.0], mono, little-endian, at 16 kHz.

Note: The possible output formats are:

- `hailort::genai::Speech2Text::generate_all_text(): std::string`: A single string containing the full transcription.
 - `hailort::genai::Speech2Text::generate_all_segments(): std::vector<Speech2Text::SegmentInfo>`: A vector of `hailort::genai::Speech2Text::SegmentInfo` objects, each containing the start and end timestamps of the segment and the transcribed text.
-

For the full example, please refer to [HailoRT C++ speech-to-text tutorial](#).

8. Command Line Tools

The HailoRT package offers a command line tool related to the device and its firmware, providing measurement, configuration and update capabilities. They can be executed from the Linux shell. It is found under the `hailortcli` command – Native (C++ based) tools that don't depend on Python.

To list the available `hailortcli` tools, run:

```
hailortcli --help
```

Note: For additional information and parameters of the sub-commands, use `--help`.

8.1. Scan Tool

The `scan` tool is used to scan all the devices that are connected to the host. It scans the PCIe hierarchy for connected Hailo devices. If found, it will retrieve the bus/device/ function of the device.

```
hailortcli scan
```

Note: The `--interface-ip` flag of this command refers to the IP address of the *host's* interface that is connected to the Hailo device. It does not refer to the address of the Hailo device itself.

8.2. Parse-HEF Tool

The `parse-hef` tool is used for parsing an existing HEF file and getting information about its content, such as input/output formats and type, single/multi context, and so on.

```
hailortcli parse-hef example.hef
```

8.3. Inference

The `run` tool is used for inference, supported only on Hailo-15. It loads a given HEF to the device and then sends and receives data. While running, it performs several measurements such as FPS and power. It can also control the running mode:

- `streaming` – Streaming inference with random data.
- `hw-only` – Similar to the full streaming mode, but skips pre-infer and post-infer steps on host.

```
hailortcli run example.hef
```

The `--batch-size` option of this tool sets the batch size in case of context switch, which means after how many frames a context switch will take place. It should not be confused with the `--frames-count` option that sets the total frame count to be sent to the device.

Note: `--frames-count` sets the total frame count, and must be dividable by `--batch-size`.

Note: When using `--measure-latency`, the HW Latency value measured is the time period between when the frame started to be sent to device until the frame was received on the host.

When using `--measure-latency` `--measure-overall-latency`, the value measure is the time period between when the application started to send the frame, and when the application finished receiving the frame (including format reordering, quantization if needed, etc.).

For most use cases and benchmarks, it is recommended to use `--measure-latency` `--measure-overall-latency`.

8.3.1. Multiple HEF Inference

`run2` is a new subcommand used to run inference over multiple networks simultaneously using the model scheduler. The running modes of `run2` are: * `full_async` - Asynchronous mode, with transformation stages. * `raw_async` - Asynchronous mode, without transformation stages. * `full_sync` - Synchronous mode, with transformation stages. * `raw_sync` - Synchronous mode, without transformation stages. For Hailo-15 devices, all modes are supported. For Hailo-10H devices, only `full_async` is supported.

The advantage of `run2` over `run` is in its parameters - `run2` allows to individually control the parameters per network and per vStream within the network (see `hailortcli run2 --help`), for example:

```
hailortcli run2 set-net hefs/ssd_mobilenet_v1.hef --batch-size 60 set-net hefs/  
→yolov5l.hef --batch-size 10
```

8.3.2. Measure Firmware Actions for the Model Profiler

Note: This command replaces `collect-runtime-data` which is used with `hailortcli run` for collecting runtime data for the previous Model profiler version.

The `measure-fw-actions` option of the inference tool creates a JSON file, `runtime_data_<hef_name>.json`, which can be used by the Model Profiler, in order to augment it with runtime data (see Dataflow Compiler documentation).

for example:

```
hailortcli run2 --mode raw set-net example.hef measure-fw-actions
```

Notes:

- `measure-fw-actions` is a subcommand (same as `set-net`, for example), and therefore it cannot be set while another command is open. This command should appear last in the command line.
- In a case where no `--batch-size` was provided, the command will run inference and measure on the batch sizes 1,2,4,8,16 - all of them in a single json file
- Since the `measure-fw-actions` option includes also FPS measurement, it excludes the options `--measure-latency` and `--measure-overall-latency`

8.3.3. Pipeline Graph Visualization

The `.dot` option (DOT - graph description language), if set, prints the pipeline graph as a `.dot` file at the specified path, which can be used to analyze the current pipeline. This format is textual, and can be converted to an image (e.g. `.png/.svg`) using DOT, an executable provided by [Graphviz](#), which is available on Ubuntu.

```
hailortcli run example.hef --dot output_path.dot
```

To get the full-detailed pipeline graph, run an example HEF with the measurements flag:

```
hailortcli run example.hef --dot output_path.dot measure-stats --elem-fps
```

This runs inference on example.hef using VStreams (by default). After the inference is complete, a .csv file containing the gathered statistics collected from the VStream pipeline elements is created, and a .dot file containing the pipeline graph is written to output_path.dot.

An example output for a selected HEF:

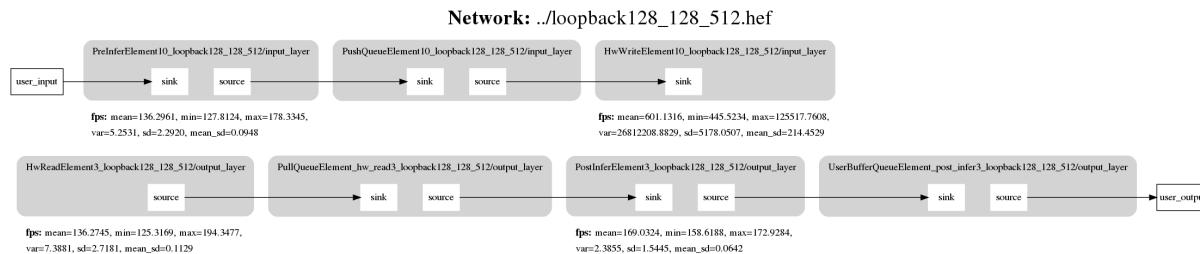


Figure 12. Pipeline Graph Visualization example

8.4. Benchmarking Tool

The `hailortcli benchmark` tool measures performance on a compiled network. This tool wraps the `hailortcli run2` tool and makes it easier to perform the full set of measurements for a given HEF. While running, it performs several measurements such as FPS, latency and power. The FPS measurement is performed twice: first, when taking only the Hailo device into account ("hardware only") and second, when taking the full system into account - including the host.

```
hailortcli benchmark example.hef
```

Note: Benchmark HEFs can be compiled using [Model Zoo](#).

- The `--batch-size` option of this tool sets the batch size in case of context switch, which means after how many frames a context switch will take place.
- In some cases, the benchmarking results might slightly vary depending on input data. To generate your own data, see [Using Datasets from the Hailo Model Zoo](#). The `--input-files` flag is used to provide the data-file path.

8.5. Monitor

The `monitor` command prints to the screen information about the models and devices which are currently running. The information is updated every second and is presented in three tables:

1. The first table shows information about the running devices:
 - Device Id – Identification string of the device, BDF for PCIe devices
 - Utilization (%) – Device utilization of the scheduler's runtime
 - Architecture – Device Architecture
2. The second table shows general information about the running models:
 - FPS – Frames per second
 - Utilization (%) – Model utilization of the scheduler's runtime
 - PID – Process id

3. The third table shows the frames state for each stream of each model:

- For host-to-device stream it will show the number of frames that can be queued, and the number of frames pending to be written to the device.
- For device-to-host stream it will show the number of frames that can be queued, and the number of frames that were read from the device and are pending to be read by the user.

Usage steps:

- 1) In the app's process, set environment variable: HAILO_MONITOR=1
- 2) Run the inference application
- 3) In a different process run:

```
hailortcli monitor
```

Note: Monitor will only work when the scheduler is active (i.e., when using [VDevice](#)).

Note: Monitor is running in a 1 second interval by default. If the user wants to change the interval, it can be done by setting the environment variable HAILO_MONITOR_TIME_INTERVAL to the desired interval in milliseconds.

Note: Monitor is currently not supported on Windows.

8.6. HailoRT Profiler

The HailoRT profiler is an html-based visualization tool, designed to allow users to see how the Model Scheduler behaves in their pipeline, and therefore allowing users to identify pipeline bottlenecks.

The intention is to allow better ease of use for HailoRT - with a quicker visual feedback when building and optimizing pipelines.

Note: In order to generate the HailoRT Profiler report - the Dataflow Compiler must be installed on the same machine. Therefore, it is recommended to generate the HailoRT profiler report with the Hailo AI Software Suite. It is not recommended to generate the HailoRT profiler report on weak or embedded hosts. However - it is entirely possible (and recommended), when using weak hosts - to gather the tracing data on the weaker host (including Hailo-15), and generate the report on a different (stronger) host after copying the trace file.

Usage steps:

- 1) Enable the HailoRT tracer to gather events: `export HAILO_TRACE=scheduler`. The tracer can be configured to generate the trace file either after a certain amount of KB or after a certain amount of time [seconds]. This can be configured using Environment Variables.

To generate the trace file after reaching N seconds use: `export HAILO_TRACE_TIME_IN_SECONDS_BOUNDED_DUMP=<N>`.

To generate the trace file after reaching N KB use: `export HAILO_TRACE_SIZE_IN_KB_BOUNDED_DUMP=<N>`.

These two options are mutually exclusive, attempting to set both will raise a warning and the profiler will ignore both options.

If not configured, the profiler will generate the trace file when the application exists.

Note: This might affect pipeline performance. Considering that a high number of events are being gathered, enabling tracing for short pipelines (less than a minute) is recommended.

After enabling the tracer and running a pipeline, a file of the type `.hrtt` will be created, e.g. `hailort_2024-07-08_16-58-30.hrtt`.

To change the location to which the trace file will be saved, use `HAILO_TRACE_PATH`, e.g. `export HAILO_TRACE_PATH=<path/to/directory/>`.

Note: When using " `HAILO_TRACE_PATH`", the defined directory must already exist.

- 2) With the Dataflow Compiler installed, run `hailo runtime-profiler <.hrtt file>` - and an html profiler report file will be opened.

Note: HailoRT Profiler is currently not supported on Windows.

8.7. Firmware Tools

8.7.1. Firmware Configuration Tool

The `fw-config` tool allows reading and writing the firmware configuration. For more information please see [User configuration documentation](#).

Run one of the following command in the Linux terminal to display the tool's help message:

```
hailortcli fw-config --help
```

Reading the Firmware Configuration

Reading the firmware configuration can be done by executing the following:

```
hailortcli fw-config read
```

This will return a JSON string that contains the current configuration. The output can be written to a file, for example:

```
hailortcli fw-config read --output-file config.json
```

Modifying the Firmware Configuration

In the case where a user config is already loaded, the easiest method is to change the configuration, that is to read it, modify it, and then write the updated JSON file. Use the following steps:

1. Read the configuration:

```
hailortcli fw-config read --output-file config.json
```

If the configs are not already loaded, the reading will fail. In that case, a sample user config at `hailort/hailortcli/example_config.json`, can be used as a reference. Further instructions will refer to the given json as `config.json` but are relevant for the `example_config.json` as well. For further information about the user config optional fields, please refer to the [User configuration documentation](#).

2. Modify `config.json` as needed.

3. Write the updated configuration:

```
hailortcli fw-config write config.json
```

The changes are only written to the Flash or EEPROM so a restart is required for the changes to take effect.

8.7.2. Firmware Control Tool

The `fw-control` tool provides useful firmware control operations.

Note: For running control operations on all existing devices over PCIe interface, use `-s *`.

Identify

The `identify` operation is used to present details about a device.

```
hailortcli fw-control identify
```

9. PCIe Driver

In order to work with a Hailo device over PCIe, a kernel driver must be installed. The PCIe driver for Linux is an external kernel module (also known as out-of-tree kernel module).

Normally, the driver is installed from the [*hailort-pcie-driver.deb*](#) or the [*Yocto recipes*](#) (for embedded platforms).

Note: By default, the PCIe driver is installed using the DKMS framework (if installed). This framework automatically rebuilds the driver when a new kernel is installed.

9.1. Manual Setup

Manual setup may be required on other platforms (for example, embedded environments without the Hailo Yocto layer).

9.1.1. Software Requirements

- wget (needed to download the firmware)
- build-essential (needed to compile the driver)
- linux-headers (can be installed using the package manager, e.g. `apt-get install linux-headers-$(uname -r)`)

9.1.2. Download

HailoRT PCIe driver sources can be cloned from GitHub using:

```
git clone https://github.com/hailo-ai/hailort-drivers.git
```

9.1.3. Compilation

Note: In order to compile the driver, a pre-built Linux kernel, including all configuration and headers, is required.

To compile the driver locally, run the following commands from the driver source path:

```
cd linux/pcie  
make all
```

The compiled driver binary will be created in the same directory as `hailo1x_pci.ko`.

9.1.4. Cross-Compilation

When cross-compiling the driver, it must be compiled with the exact kernel installed on the target host, including:

1. Same kernel commit or added patches.
2. Same compiler.
3. Same kernel configuration file. The configuration file is named `.config`, under the kernel source directory.

To ensure these conditions, it is recommended to enable the `CONFIG_MODVERSIONS` kernel configuration option. The option ensures that any driver loaded into the kernel is compiled with the exact same kernel version.

9.1.5. Installation

Before installation need to download the Hailo-10H firmware

```
./download_firmware_hailo10h.sh  
sudo mkdir -p /lib/firmware/hailo  
sudo mv hailo10h.<VERSION> /lib/firmware/hailo/  
sudo ln -s /lib/firmware/hailo/hailo10h.<VERSION> /lib/firmware/hailo/hailo10h
```

In order to install the driver, run the following:

```
# Install the driver in /lib/modules  
sudo make install  
  
# Load the driver (needs to be done once, after installation the driver will be loaded  
# on boot)  
sudo modprobe hailo1x_pci
```

Then update the device manager:

- Copy the udev rules

```
sudo cp ./linux/pcie/51-hailo-udev.rules /etc/udev/rules.d/
```

- To apply the changes, either reboot or run:

```
sudo udevadm control --reload-rules && sudo udevadm trigger
```

9.2. Parameters

The PCIe driver supports parameters that can change the driver's behavior:

- **no_power_mode**
 - Disables automatic PCIe D0->D3 transition.

There are two options for changing the parameters:

1. Edit/create the file `/etc/modprobe.d/hailo1x_pci.conf`
 - A template file (with the same name) is provided in the driver's source directory.
 - Reloading the driver is required.
2. Define the desired parameters upon loading the driver via modprobe. For example:

```
sudo modprobe hailo1x_pci no_power_mode=Y
```

10. SoC Features

10.1. Temperature Monitoring

During operation, the system continuously monitors the chip's internal temperature and verifies that the device is operating within the safe temperature range by executing the following mechanisms:

- Temperature throttling (may be disabled)
- Over-temperature protection (always on, kicks in when reaching the temperature AMR)

When the temperature throttling feature is enabled, the firmware monitors the chip's junction temperature ($T_{junction}$), performing frequency throttling as required to avoid exceeding the temperature AMR.

To that end, the chip's junction temperature range has been divided into three zones:

- Green Zone - Normal operation (no throttling). The Green Zone's temperature range is determined by `temperature_orange_threshold` and `temperature_orange_hysteresis_threshold`.
- Orange Zone - Throttling is active (if enabled) to supervise the chip's temperature. The Orange Zone's temperature range is determined by `temperature_orange_threshold`, `temperature_orange_hysteresis_threshold`, `temperature_red_threshold` and `temperature_red_hysteresis_threshold`.
- Red Zone - Not safe for operation. The Red Zone's temperature range is determined by `temperature_red_threshold` and `temperature_red_hysteresis_threshold`.

As long as $T_{junction}$ remains below the `temperature_orange_threshold`, the firmware remains in the Green Zone and continues normal operation. After exceeding the `temperature_orange_threshold`, the firmware enters the Orange Zone and decreases the internal clock frequency gradually, from a performance of 100% down to a performance of 50% and recovers as explained below. In case the `temperature_red_threshold` is exceeded, the chip enters the Red Zone, in which the over-temperature protection kicks in, and the input and output streams will be shut down. Any attempt to open new streams when the device is still on the Red Zone will fail.

In case the chip is cooling down, the firmware increases the internal clock frequency gradually, recovering the performance degradation caused by the thermal throttling. Performance recovery is based on a hysteresis threshold for each of the corresponding thresholds mentioned above; this is intended to stabilize the throttling process and prevent frequent clock and zone changes.

The chip changes from the Orange Zone to the Red Zone when $T_{junction}$ exceeds `temperature_red_threshold` and returns to the Orange Zone when $T_{junction}$ is lower than `temperature_red_hysteresis_threshold`. The same logic applies when returning from the Orange Zone to the Green Zone, with `temperature_orange_threshold` and `temperature_orange_hysteresis_threshold` respectively.

The parameters `temperature_orange_threshold`, `temperature_orange_hysteresis_threshold`, `temperature_red_threshold` and `temperature_red_hysteresis_threshold` can be overwritten by the [User configuration](#), although adjusting them is not recommended.

Table 2. Temperature Thresholds

Threshold Type	Description	Default value (in degree Celsius)
<code>temperature_orange_threshold</code>	Exceeding this threshold triggers entrance to the Orange Zone, which activates the throttling mechanism	104
<code>temperature_orange_hysteresis_threshold</code>	After being in the Orange Zone, moving back below this threshold returns the chip back to the Green Zone, where it is functioning normally	99

Continued on next page

Table 2 – continued from previous page

Threshold Type	Description	Default value (in degree Celsius)
temperature_red_threshold	Exceeding this threshold triggers entrance to the Red Zone, which is considered not safe for operation	120
temperature_red_hysteresis_threshold	After being in the Red Zone, moving back below this threshold returns the chip back to the Orange Zone	116

Note: Current is also monitored continuously, and overcurrent throttling is enabled by default and should not be disabled.

10.2. User Configuration

The firmware has various settings that can be managed by the user. This configuration is stored on the board's flash or EEPROM memory. It is read during boot time and contains several entries such as board name and max neural network core clock rate. The configuration consists of several sections, for example:

- **System** – includes fields related to power, frequency, and maintenance:
 - `name` – contains the board name. This name is used for identification.
 - `max_neural_network_core_clock_rate` – sets the clock rate of the neural network core. The supported values for this field are 100, 200 and 400 MHz, but it cannot raise the clock rate above the maximum supported value for each board or module.
 - `overcurrent_parameters_source` – can be set to `USER_CONFIG_VALUES` to let the values in the other over current configuration fields to take effect. Note that over current protection is only available on the mPCIe and M.2 modules.
 - `overcurrent_monitoring_red_threshold` – sets the maximum current threshold. This field supports any integer. It is given in mA units.
 - `overcurrent_conversion_time_microseconds` – sets the current sensor averaging period. It supports the following values: 140, 204, 332, 588, 1100, 2116, 4156, 8244. All values are given in microseconds.
 - `overcurrent_throttling_enable` – enables the overcurrent throttling option of the overcurrent monitoring component. Note that overcurrent throttling is not available on the mPCIe module.
 - `temperature_parameters_source` – can be set to `USER_CONFIG_VALUES` to let the values in the other temperature configuration fields to take effect.
 - `temperature_red_threshold` – sets the red threshold of the temperature monitoring component. When Tjunction reaches above this threshold, all PCIe streams are forcefully closed. All values are given in degree Celsius. Adjusting this parameter is not recommended.
 - `temperature_red_hysteresis_threshold` – sets the red hysteresis threshold of the temperature monitoring component. Tjunction has to decrease below this threshold (while being in the Red Zone) to allow the device to return to the Orange Zone. All values are given in degree Celsius. Adjusting this parameter is not recommended.
 - `temperature_orange_threshold` – sets the orange threshold of the temperature monitoring component. When Tjunction reaches above this threshold, the temperature throttling begins (in case all other conditions are met). All values are given in degree Celsius. Adjusting this parameter is not recommended.

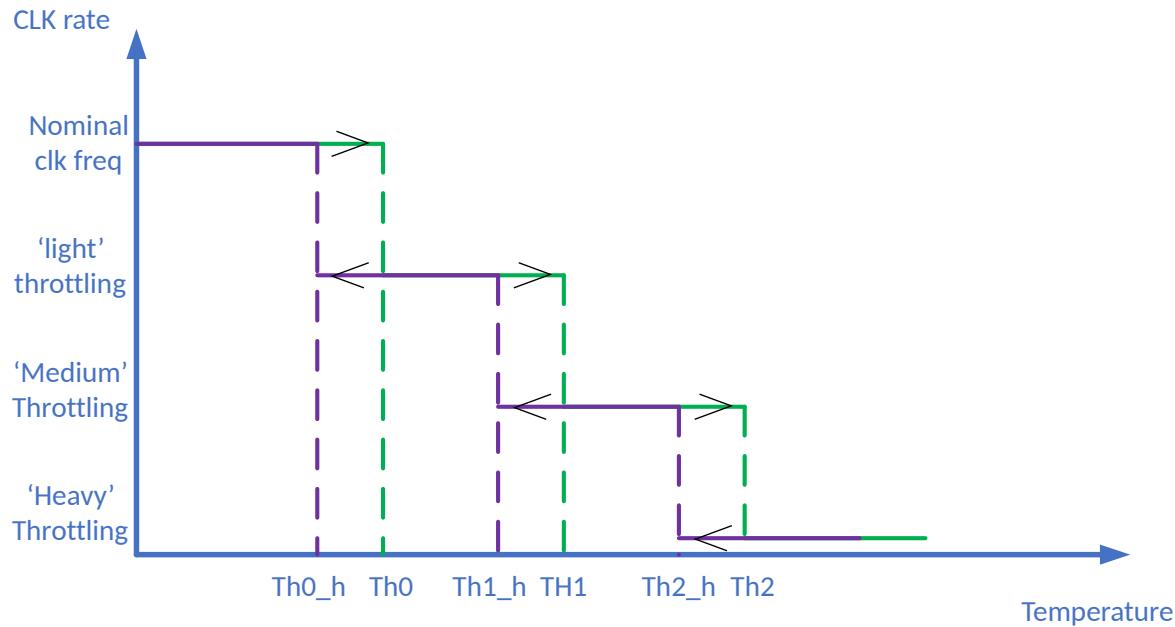


Figure 13. A qualitative frequency/temperature graph

- `temperature_orange_hysteresis_threshold` - sets the orange hysteresis threshold of the temperature monitoring component. Tjunction has to decrease below this threshold (while being in the Orange Zone) in order for the device to return back to the Green Zone. All values are given in degree Celsius. Adjusting this parameter is not recommended.
- `temperature_throttling_enable` - enables the temperature throttling option of the temperature monitoring component. Note that temperature throttling is not available on the mPCIe module.

Warning: Temperature and overcurrent throttling should remain enabled, disabling the features should occur only in special cases.

10.2.1. Example Configuration

The configuration uses the JSON format. For example:

```
{
  "system": {
    "name": "Hailo Test",
    "max_neural_network_core_clock_rate": "100MHZ",
    "overcurrent_parameters_source": "USER_CONFIG_VALUES",
    "overcurrent_monitoring_red_threshold": 1100,
    "overcurrent_conversion_time_microseconds": 140
  }
}
```

This configuration example reduces the maximum clock rate of the neural network core to 100 MHz in order to consume less power. This example also changes the parameters of the mPCIe and M.2 modules over current protection.

For additional information regarding reading or editing the user config please refer to the [Firmware configuration tool doc](#).

10.3. Power Modes

Hailo devices support two power modes (via `hailo_power_mode_t`):

- Performance power mode (default)
- Ultra-performance power mode

The difference between these two modes is the PCIe data-transfer size. In performance power mode - each PCIe data transfer is larger, allowing the PCIe data lanes to remain quiet for a longer duration and as a result switch to deeper PCIe power states in case the PCIe ASPM (Active-State Power Management) mode is enabled (leads to lower power consumption). This is the default working mode. In contrast to that, in ultra-performance power mode, each PCIe data transfer is smaller, and therefore might not be long enough to allow switching to a lower power state.

Note that in some cases, working in the ultra-performance power mode might lead to increased performance (at the cost of power) when compared to the default performance power mode. This might be the result of other system flows, such as the CPU entering a C-state low power mode due to the transfer size changes. To maximize performance it is possible to enable ultra-performance power mode.

11. Yocto

11.1. The Meta-Hailo Layers

This section will guide you through the integration of Hailo's Yocto layers into your own Yocto environment.

The layers are stored in the [Meta-Hailo GitHub repository](#), with a branch for each supported Yocto release:

- Kirkstone 4.0 (kernel 5.15)
- Mickledore 4.2 (kernel 6.1)

The instructions were written after being tested on the [DART-MX8M](#) embedded platform.

There are 3 exported layers:

11.1.1. HailoRT

- `meta-hailo-libhailort` - contains HailoRT usermode library, binaries, python, etc.
- `meta-hailo-accelerator` - contains HailoRT PCIe driver, and Hailo firmware.
- `meta-hailo-vpu` - contains HailoRT NNC driver, and Hailo-15 firmware.

11.1.2. TAPPAS

- `meta-hailo-tappas` - contains *TAPPAS* recipes including libgsthailotools. More details about these recipes, how to setup and integrate them, can be found in the *TAPPAS* documentation.

11.2. Recipes in Meta-Hailo-Libhailort

11.2.1. Libhailort

Hailo's API for running inference on the Hailo chip. The recipe compiles the libhailort shared object into the target device's root file system (/usr/bin)

11.2.2. Hailortcli

Hailortcli is a command line utility wrapper for libhailort operations, including inference, firmware controls, measurements, and more.

11.2.3. Pyhailort

Hailo's python API. The recipe compiles pyhailort shared object, unpacks the python directory, and uses `setup-tools` to install into `python/site-packages` on the target device's root file system.

The recipe depends on all the python recipes under `meta-hailo/recipes-python/`.

Note: pyhailort is not supported on Hardknott and later versions.

Python dependencies:

```
python3-argcomplete, python3-cppheaderparser, python3-netifaces, python3-tqdm, □
→python3-aspy.yaml,
python3-distlib, python3-pyzmq, python3-cfgv, python3-future,
python3-zipp.python3-contextlib2, python3-identify, python3-textutils, python3-zmq
```

11.2.4. Libgsthailo

Hailo's GStreamer plugin for running inference on the Hailo chip. Depends on `libhailort` and GStreamer. The recipe compiles and copies the `libgsthailo.so` file to `/usr/lib/gstreamer-1.0` on the target device's root file system, enabling it to be loaded by GStreamer as a plugin.

11.2.5. Hailort-Server

HailoRT-Server is a mechanism for communication between a client and a server, enabling remote execution of operations on Hailo devices. The recipe compiles and copies the `hailort_server` binary to `/usr/bin/` on the target device's root file system.

11.3. Recipes in Meta-Hailo-Accelerator

11.3.1. Hailo-Firmware

Hailo-10H firmware (`hailo10h/`). The recipe copies the firmware files to the `/lib/firmware/hailo/` directory on the target device's root file system.

11.3.2. Hailo Pcie Driver

Recipe name: `hailo-pci`

Compiles Hailo-10H's PCIe driver and installs it. The source files can be found on [GitHub](#).

11.4. Recipes in Meta-Hailo-Vpu

11.4.1. Hailo15-Nnc-Fw

Hailo-15 NNC (neural network core) firmware (`hailo15_nnc_fw.bin`). The recipe copies the file to the `/lib/firmware/hailo` directory on the target device's root file system.

11.4.2. Hailo-Integrated-Nnc

Compiles Hailo-15's NNC (neural network core) driver and installs it. The source files can be found at [GitHub](#).

11.5. Integrating with an Existing Yocto Environment

1. Meta-Hailo sources can be cloned from GitHub using:

```
git clone https://github.com/hailo-ai/meta-hailo.git
```

Note: It is recommended to clone meta-hailo into the sources directory.

2. After cloning meta_hailo, checkout a specific branch, for example (for Kirkstone)

```
cd meta-hailo  
git checkout kirkstone
```

3. The Yocto directory of all supported Yocto releases includes the meta-hailo layers. Open the directory of your required Yocto release and copy the meta-hailo layer directory to your sources directory (same directory as poky, meta-openembedded etc.). Then add it to your conf/bblayers.conf like so:

```
BBLAYERS += " ${BSPDIR}/sources/meta-hailo/meta-hailo-accelerator \  
           ${BSPDIR}/sources/meta-hailo/meta-hailo-libhailort"
```

4. Add the recipes to your image in conf/local.conf:

```
IMAGE_INSTALL_append = "libhailort hailortcli pyhailort libgsthailo hailo-  
                        ↪pci hailo-firmware"
```

An example of the expected directory structure:

```
/local  
  yocto-kirkstone/  
    build_xwayland/  
      conf/  
        local.conf  
        bblayers.conf  
      cache/  
      tmp/  
      workspace/  
    setup-environment  
    README  
    sources/  
      poky/  
      meta-hailo/  
        meta-hailo-libhailort/  
        meta-hailo-accelerator/  
      meta-openembedded/
```

11.6. Validating the Integration's Success

Make sure that the following conditions have been met on the target device:

Recipe	Details
libhailort	Libhailort exists at: <code>/usr/lib/libhailort.so</code>
hailortcli	<code>hailortcli</code> command works from the global environment. After connecting the hailo chip - <code>scan</code> and <code>run</code> commands should communicate with the board.
pyhailort	Python3 includes the <code>hailo_platform</code> module and all dependent modules. <code>hailo</code> command works from the global environment. After connecting the hailo chip - <code>fw-control identify</code> recognizes it and prints information.
libgsthalo	Running <code>gst-inspect-1.0 grep hailo</code> returns hailo elements: <pre>hailo: hailonet: hailonet element hailodevicesstats: hailodevicesstats element</pre>
hailort-server	HailoRT-Server exists at: <code>/usr/bin/hailort_server</code>
hailo-firmware	Hailo's firmware exists at: <code>/lib/firmware/hailo/</code>
hailo-pci	Running <code>modinfo hailo-pci</code> should return a successful result and info about the driver.

11.7. Offline Builds

Recipes in Yocto may be built offline, if the source code was previously collected on a machine that was connected to a network. The sources collected for the desired recipes will be found at `$DL_DIR` and they are suitable for mirroring (see [DL_DIR](#) and [source mirrors](#) for more information).

It is assumed that a Yocto environment exists on a machine with network access, which will be used to collect sources for the desired recipes. These sources will be copied to an offline machine with a Yocto environment, which will be used to build the aforementioned recipes. See [Usage Example](#) for detailed instructions.

11.7.1. Prepare HailoRT External Dependencies

The `prepare_hailort_external_dependencies` task will collect the recipe's sources and external dependencies, packaging them in a `.tar` file in `$DL_DIR`. The following recipes implement this task:

- libhailort
- libgsthalo
- hailortcli
- pyhailort-shared

11.7.2. Environment Variables

Two environment variables control the offline build flow of HailoRT recipes:

- `HAILORT_OFFLINE_BUILD_ENABLE` - Set to "1" to run the offline build flow, both on the machine with network access (that is used to collect sources) and the offline machine (that is used for building).

Note: If this variable is set to "1", the `$DL_DIR` isn't suitable for sharing between parallel builds of HailoRT recipes.

- HAILORT_OFFLINE_BUILD_USE_EXISTING_TAR - Set to "1" on the offline machine. The recipe will search for the .tar file previously created by `prepare_hailort_external_dependencies`.

These variables are set in `conf/local.conf`.

11.7.3. Usage Example

The steps outlined in [Integrating with an Existing Yocto Environment](#) are to be run prior to this example. It is assumed that we are operating with the [directory structure](#) described above.

- On a machine with network connection:

1. Add the following lines to `conf/local.conf`:

- `HAILORT_OFFLINE_BUILD_ENABLE="1"` - Required for collecting and packing sources and external dependencies for HailoRT recipes.
- `BB_GENERATE_MIRROR_TARBALLS = "1"` - This is required to make the `$DL_DIR` safe for mirroring or copying (see [BB_GENERATE_MIRROR_TARBALLS](#) for more information).

2. From the `sources` directory run:

```
source poky/oe-init-build-env
```

3. Collect sources for the the desired hailort recipes. In this example, sources for the `libhailort` and `hailortcli` recipes are collected.

1. Collect sources for the desired hailort recipes using `--runall=fetch`:

```
bitbake libhailort --runall=fetch  
bitbake hailortcli --runall=fetch
```

2. Run the `prepare_hailort_external_dependencies` task on the desired hailort recipes:

```
bitbake -c prepare_hailort_external_dependencies libhailort hailortcli
```

3. Collect any other sources needed for the image. Sources for `core-image-minimal` are collected in this example:

```
bitbake core-image-minimal --runall=fetch
```

- The contents of `$DL_DIR` can now be copied to a machine without network access, where they can be built. For example, building the `libhailort` and `hailortcli` recipes is done as follows on a machine without network access:

1. From the `sources` directory run:

```
source poky/oe-init-build-env
```

2. Clean artifacts from previous runs:

```
bitbake libhailort --runall=clean  
bitbake hailortcli --runall=clean
```

3. Copy the `$DL_DIR` created on the machine with network connection to the `$DL_DIR` used on the offline machine.

4. Add the following lines to `conf/local.conf`:

- `HAILORT_OFFLINE_BUILD_ENABLE = "1"` - Required for unpacking the HailoRT recipe's sources and external dependencies.

- `HAILORT_OFFLINE_BUILD_USE_EXISTING_TAR = "1"` - The recipe will search for the `.tar` file previously created by `prepare_hailort_external_dependencies`, that was copied to the offline machine's `$DL_DIR`.

Note: The following may also be added to `conf/local.conf`:

- `BB_NO_NETWORK = "1"` - In order to validate that no network access is made by bitbake (see [BB_NO_NETWORK](#)).
 - `BB_SRCREV_POLICY = "cache"` - Might be required in order to stop `git update` from being called during some tasks (see [BB_SRCREV_POLICY](#)).
-

5. Build the desired hailort recipes:

```
bitbake -c build libhailort hailortcli
```

12. Integration with Frameworks

12.1. GStreamer

[GStreamer multimedia framework](#) is a framework for creating streaming media applications. GStreamer's development framework makes it possible to write any type of streaming multimedia application. The GStreamer framework is designed to make it easy to write applications that handle audio or video or both. It isn't restricted to audio and video and can process any kind of data flow. The framework is based on plugins that will provide various codecs and other functionality. The plugins can be linked and arranged in a pipeline. This pipeline defines the flow of the data. The GStreamer core function is to provide a framework for plugins, data flow, and media type handling/negotiation. It also provides an API to write applications using the various plugins.

Hailo's application framework for optimized execution of video-processing pipelines ([TAPPAS](#)) is GStreamer based.

12.1.1. HailoNet

HailoNet is a GStreamer element which enables inference over a Hailo device.

The HailoNet element will read the output buffers from the device and attach them as metadata - to the source frame that inferred them. Therefore, HailoNet **has only one source**, even in cases where the HEF has more than one output layer. For the full element documentation, run `gst-inspect-1.0 hailonet`.

Important Parameters

- The data is inferred according to the selected HEF (`hef-path`).
- Selecting a specific device can be done with the `device-id` property.
- In order to share resources between multiple HailoNet elements, use the same `vdevice-group-id` on each HailoNet element.

Note: In order to share resources between different processes, `multi-process-service` needs to be enabled.

- If the pipeline contains a single HailoNet element, then that single HailoNet will be activated by default. Otherwise, each HailoNet element must explicitly specify if it is active or not by setting the `is_active` property. This property can also be used for switching the activated networks on a specific device during runtime.

Note: When `Model Scheduler` is enabled, the `is_active` property should not be passed.

- For multi-context networks, the `batch-size` property can be used to specify the batch size.

13. Development Guidelines

This section contains information that will help with setting up, working with, and debugging HailoRT.

13.1. Common Errors

When encountering an error, it can be initially challenging to understand the issue by relying only on the HailoRT log. The section below will provide detailed explanations of some common errors, their possible root causes, and recommended debug directions. For a full list of all possible HailoRT errors see [hailo_status](#).

HAILO_RPC_FAILED

- This error indicates a communication issue between the client and the server. Possible root cause - version mismatches between libhailort and hailort_server, the hailort_server process not being active or incorrectly initialized, or a previous system error that caused hailort_server to terminate unexpectedly.
- Start by verifying that the hailort_server process is running. Refer to [Multi-Process service](#) for instructions on starting or restarting the service. Additionally, review the hailort logs (both client and server logs) to identify the underlying cause of the communication failure.

HAILO_TIMEOUT

- This error occurs when a function exceeds its specified timeout duration. The hailort log provides details about the specific timeout that was triggered and its duration.
- Send/Write operation timeout:
 - If the system isn't receiving or reading data from the device quickly enough, back-pressure in the pipeline can cause the send operation to block and eventually timeout.
 - Consider extending the input's timeout duration or ensuring that the application processes the outputs faster.
- Receive/Read operation timeout:
 - A timeout can occur if the read and write operations are out of sync (e.g., writing 3 frames but attempting to read 4).
 - Implement a synchronization mechanism between read and write operations to prevent this issue.
- `wait_for_async_ready()` ([C++](#), [Python](#)) timeout:
 - This timeout suggests that the pipeline wasn't ready to handle new transfer requests.
 - This can happen if the timeout duration is too short relative to the pipeline's latency or if the `frames_count` passed to the function exceeds the `get_async_queue_size()` ([C++](#), [Python](#)).

HAILO_UNSUPPORTED_OPCODE

- This error indicates the device received a non-recognized opcode. Possible root cause - version mismatches between libhailort and the firmware. happen because of an unsuccessful update, try to re-install.

HAILO_DRIVER_OPERATION_FAILED

- This error indicates a failure running some driver operation (i.e IOCTL). For more info, read the driver log (On linux, check the dmesg log)

HAILO_INTERNAL_FAILURE

- This error indicates an unexpected internal failure. Please contact Hailo for further assistance.

13.2. Tips and Best Practices

The section below contains tips and best-practices when developing an application which utilizes HailoRT.

Reading hailort log

- The hailort log directory is determined by the environment variable `HAILORT_LOGGER_PATH` - see [Environment Variables](#).
- The hailort log provides valuable insights into the internal workings of the libhailort. By adjusting the trace levels (refer to [Environment Variables](#)), users can obtain more detailed logs, enabling easier troubleshooting and debugging.
- Errors in libhailort are typically interdependent, meaning that a single underlying issue can lead to multiple error messages in the log. When diagnosing a problem, it's crucial to identify the first error message in the sequence, as it usually points to the root cause of the issue.

Reading device logs

- The device logs are to be read from the 'hailortcli logs <LOG_TYPE>' command (`LOG_TYPE` is one of `hailo_log_type_t` enum values).
- Reading a log clears its content, so subsequent reads will return only new log entries added after the last read.
- Logs are printed to the console, and user can use the `--follow` option to continuously print new logs as they are generated. This is particularly useful for monitoring real-time events and debugging issues as they occur.
- In the case of NNC log type, the content is encrypted. Its output should be redirected to a file using the '`>`' terminal option (for instance, `hailortcli logs nnc > nnc.log`), and sent to Hailo support for further analysis. Also, `--follow` option is not allowed on this log type.

Debugging Performance

- Debugging performance can be complex. In this section, we will provide initial guidance on how to better understand and address performance issues.
- Single Model Performance:
 - Identifying performance bottlenecks is crucial. These bottlenecks can originate either from the host CPU or from the Hailo device's NN-core.
 - Review the model pipeline description in the HailoRT log to ensure it aligns with your expectations. Mis-configuration may introduce unnecessary elements, leading to degraded performance and increased CPU utilization.
 - Use `hailortcli` to evaluate the maximum performance of your model on the system, testing both full and raw modes:

- * `full_async` mode runs the entire inference pipeline, including pre-inference and post-inference steps (e.g., data preparation like quantization or frame re-ordering). These stages typically execute on the host CPU.

```
hailortcli run2 --mode full_async set-net your_model.hef
```

- * `raw_async` mode only runs the core inference on the Hailo device, bypassing the pre-inference and post-inference stages.

```
hailortcli run2 --mode raw_async set-net your_model.hef
```

- Multiple Models Pipeline:

- When debugging performance of a multiple-models pipeline with [Model Scheduler](#), it is possible to follow the scheduler's decision-making using the [HailoRT profiler](#).

Creation and Destruction Order

- Whenever creating HailoRT instances in your app (e.g. `VDevice`, `InferModel`, `ConfiguredInferModel`), it is important to release them in the reverse order of creation (release `VDevice` last) - those instances have an internal connection between them, and releasing them in any other order may lead to undefined behavior

Note: When working with PyHailoRT, it is recommended to use `context manager` when possible to ensure the order of destruction.

Compiling from sources

- The CMake-based build system will try to use cached values from previous runs. When changing build configurations, such as target architecture, or build type, delete `CMakeCache.txt`

PyHailoRT and multi-process best practices

- On Hailo-10H device , multi process support is enabled by default, and it enables resource-sharing, such as the `VDevice` and [Model Scheduler](#), across multiple processes. Note that a `VDevice` is only accessible within the process in which it was created. To share the `VDevice` between processes, assign the same `VDeviceParams.group_id` across all relevant processes We recommend creating a separate instance of `VDevice` in each process.
 - [InferModel API](#): See the [Python Async Inference Tutorial for Multiple Models](#).

Note: When using Hailo-10H device with multiple processes, each model must be managed within a single process. The recommended practice is to use the asynchronous approach with the [InferModel API](#). Using multiple processes to handle the same model (e.g., by utilizing `InputVStream` and `OutputVStream`) can lead to undefined behavior.

Part II

API Reference

14. HailoRT C API Reference

HailoRT is Hailo's runtime library. The C API is used for running inference over compiled models (HEFs) in a C/C++ program.

14.1. Device and Control API Functions

`hailo_status` `hailo_scan_devices`(`hailo_scan_devices_params_t` *params, `hailo_device_id_t` *device_ids, ...)

Returns information on all available devices in the system.

Parameters

- `params` - **[in]** Scan params, used for future compatibility, only NULL is allowed.
- `device_ids` - **[out]** Array of `hailo_device_id_t` that were scanned.
- `device_ids_length` - **[inout]** As input - the size of `device_ids` array. As output - the number of devices scanned.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_create_device_by_id`(const `hailo_device_id_t` *device_id, `hailo_device` *device)

Creates a device by the given device id.

Note: To release a device, call the `hailo_release_device` function with the returned `hailo_device`.

Parameters

- `device_id` - **[in]** Device id, can represent several device types: [-] for pcie devices - pcie bdf (XXXX:XX:XX.X) [-] for ethernet devices - ip address (xxx.xxx.xxx.xxx) If NULL is given, uses an arbitrary device found on the system.
- `device` - **[out]** A pointer to a `hailo_device` that receives the allocated PCIe device.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_scan_PCIE_devices`(`hailo_PCIE_device_info_t` *pcie_device_infos, `size_t` ...)

Returns information on all available pcie devices in the system.

Parameters

- `pcie_device_infos` - **[out]** A pointer to a buffer of `hailo_PCIE_device_info_t` that receives the information.
- `pcie_device_infos_length` - **[in]** The number of `hailo_PCIE_device_info_t` elements in the buffer pointed to by `pcie_device_infos`.
- `number_of_devices` - **[out]** This variable will be filled with the number of devices. If the buffer is insufficient to hold the information a `HAILO_INSUFFICIENT_BUFFER` error is returned.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_parse_PCIE_device_info`(const char *device_info_str, `hailo_PCIE_device_info_t` ...)

Parse PCIe device BDF string into hailo device info structure.

Note: Call `hailo_scan_PCIE_devices` to get all available hailo pcie devices.

Parameters

- `device_info_str` - **[in]** BDF device info, format <domain>.<bus>.<device>.<func>.
- `device_info` - **[out]** A pointer to a `hailo_pcie_device_info_t` that receives the parsed device info.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_create_pcie_device(hailo_pcie_device_info_t *device_info, hailo_device *device)`

Creates a PCIe device.

Note: To release a device, call the `hailo_release_device` function with the returned `hailo_device`.

Parameters

- `device_info` - **[in]** Information about the device to open. If NULL is given, uses an arbitrary device found on the system.
- `device` - **[out]** A pointer to a `hailo_device` that receives the allocated PCIe device.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_release_device(hailo_device device)`

Releases an open device.

Parameters `device` - **[in]** A `hailo_device` object to be released.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_device_get_type_by_device_id(const hailo_device_id_t *device_id, ...)`

Returns the device type of the given device id string.

Parameters

- `device_id` - **[in]** A `:hailo_device_id_t` device id to check.
- `device_type` - **[out]** A `:hailo_device_type_t` returned device type.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_identify(hailo_device device, hailo_device_identity_t *device_identity)`

Sends identify control to a Hailo device.

Parameters

- `device` - **[in]** A `hailo_device` to be identified.
- `device_identity` - **[out]** Information about the device.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_core_identify(hailo_device device, hailo_core_information_t *core_information)`

Receive information about the core cpu.

Parameters

- `device` - **[in]** A `hailo_device` object.
- `core_information` - **[out]** Information about the device.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_get_extended_device_information(hailo_device device, ...)`

Get extended device information from a Hailo device.

Parameters

- `device` - **[in]** A `hailo_device` to get extended device info from.

- **extended_device_information-[out]** Extended information about the device.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_set_fw_logger(hailo_device device, hailo_fw_logger_level_t level, uint32_t interface_mask)`

Configure fw logger level and interface of sending.

Parameters

- `device - [in]` A `hailo_device` object.
- `level - [in]` The minimum logger level.
- `interface_mask - [in]` Output interfaces (mix of `hailo_fw_logger_interface_t`).

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_set_throttling_state(hailo_device device, bool should_activate)`

Change throttling state of temperature protection and overcurrent protection components. In case that change throttling state of temperature protection didn't succeed, the change throttling state of overcurrent protection is executed.

Parameters

- `device - [in]` A `hailo_device` object.
- `should_activate - [in]` Should be true to enable or false to disable.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_get_throttling_state(hailo_device device, bool *is_active)`

Get current throttling state of temperature protection and overcurrent protection components. If any throttling is enabled, the function return true.

Parameters

- `device - [in]` A `hailo_device` object.
- `is_active - [out]` A pointer to the temperature protection or overcurrent protection components throttling state: true if active, false otherwise.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_wd_enable(hailo_device device, hailo_cpu_id_t cpu_id)`

Enable firmware watchdog.

Note: Advanced API. Please use with care.

Parameters

- `device - [in]` A `hailo_device` object.
- `cpu_id - [in]` A `hailo_cpu_id_t` indicating which CPU WD to enable.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_wd_disable(hailo_device device, hailo_cpu_id_t cpu_id)`

Disable firmware watchdog.

Note: Advanced API. Please use with care.

Parameters

- `device - [in]` A `hailo_device` object.

- **cpu_id - [in]** A *hailo_cpu_id_t* indicating which CPU WD to disable.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status *hailo_wd_config*(*hailo_device* device, *hailo_cpu_id_t* cpu_id, *uint32_t* wd_cycles, ...)

Configure firmware watchdog.

Note: Advanced API. Please use with care.

Parameters

- **device - [in]** A *hailo_device* object.
- **cpu_id - [in]** A *hailo_cpu_id_t* indicating which CPU WD to configure.
- **wd_cycles - [in]** Number of cycles until watchdog is triggered.
- **wd_mode - [in]** A *hailo_watchdog_mode_t* indicating which WD mode to configure.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status *hailo_get_previous_system_state*(*hailo_device* device, *hailo_cpu_id_t* cpu_id, *uint32_t* ...)

Read the FW previous system state.

Note: Advanced API. Please use with care.

Parameters

- **device - [in]** A *hailo_device* object.
- **cpu_id - [in]** A *hailo_cpu_id_t* indicating which CPU to state to read.
- **previous_system_state - [out]** A *uint32_t* to be filled with the previous system state. 0 indicating external reset, 1 indicating WD HW reset, 2 indicating WD SW reset, 3 indicating SW control reset.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status *hailo_set_pause_frames*(*hailo_device* device, *bool* rx_pause_frames_enable)

Enable/Disable Pause frames.

Parameters

- **device - [in]** A *hailo_device* object.
- **rx_pause_frames_enable - [in]** Indicating whether to enable or disable pause frames.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status *hailo_get_device_id*(*hailo_device* device, *hailo_device_id_t* *id)

Get device id which is the identification string of the device (i.e bdf for pcie devices)

Parameters

- **device - [in]** A *hailo_device* object.
- **id - [out]** The returned device id.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

`hailo_status hailo_get_chip_temperature(hailo_device device, hailo_chip_temperature_info_t *temp_info)`

Get temperature information on the device

Note: Temperature in Celsius of the two internal temperature sensors (TS).

Parameters

- `device - [in]` A `hailo_device` object.
- `temp_info - [out]` A `hailo_chip_temperature_info_t` to be filled.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_reset_device(hailo_device device, hailo_reset_device_mode_t mode)`

Reset device

Parameters

- `device - [in]` A `hailo_device` object.
- `mode - [in]` The mode of the reset.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_update_firmware(hailo_device device, void *firmware_buffer, uint32_t ...)`

Updates firmware to device flash.

Note: Check `hailo_extended_device_information_t.boot_source` returned from `hailo_get_extended_device_information` to verify if the fw is booted from flash.

Parameters

- `device - [in]` A `hailo_device` object.
- `firmware_buffer - [in]` A pointer to a buffer that contains the firmware to be updated on the `device`.
- `firmware_buffer_size - [in]` The size in bytes of the buffer pointed by `firmware_buffer`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_update_second_stage(hailo_device device, void *second_stage_buffer, uint32_t ...)`

Updates second stage to device flash.

Note: Check `hailo_extended_device_information_t.boot_source` returned from `hailo_get_extended_device_information` to verify if the fw is booted from flash.

Parameters

- `device - [in]` A `hailo_device` object.
- `second_stage_buffer - [in]` A pointer to a buffer that contains the second_stage to be updated on the `device`.
- `second_stage_buffer_size - [in]` The size in bytes of the buffer pointed by `second_stage_buffer`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_set_notification_callback(hailo_device device, hailo_notification_callback ...)`

Sets a callback to be called when a notification with ID `notification_id` will be received

Parameters

- `device` - **[in]** A `hailo_device` to register the callback to.
- `callback` - **[in]** The callback function to be called.
- `notification_id` - **[in]** The ID of the notification.
- `opaque` - **[in]** User specific data.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_remove_notification_callback(hailo_device device, hailo_notification_id_t ...)`

Removes a previously set callback with ID `notification_id`

Parameters

- `device` - **[in]** A `hailo_device` to register the callback to.
- `notification_id` - **[in]** The ID of the notification to remove.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_reset_sensor(hailo_device device, uint8_t section_index)`

Reset the sensor that is related to the section index config.

Parameters

- `device` - **[in]** A `hailo_device` object.
- `section_index` - **[in]** Flash section index to load config from. [0-6]

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_set_sensor_i2c_bus_index(hailo_device device, hailo_sensor_types_t ...)`

Set the I2C bus to which the sensor of the specified type is connected.

Parameters

- `device` - **[in]** A `hailo_device` object.
- `sensor_type` - **[in]** The sensor type.
- `bus_index` - **[in]** The I2C bus index of the sensor.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_load_and_start_sensor(hailo_device device, uint8_t section_index)`

Load the configuration with I2C in the section index.

Parameters

- `device` - **[in]** A `hailo_device` object.
- `section_index` - **[in]** Flash section index to load config from. [0-6]

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_i2c_read(hailo_device device, const hailo_i2c_slave_config_t *slave_config, uint32_t ...)`

Read data from an I2C slave over a hailo device.

Parameters

- `device` - **[in]** A `hailo_device` object.
- `slave_config` - **[in]** The `hailo_i2c_slave_config_t` configuration of the slave.
- `register_address` - **[in]** The address of the register from which the data will be read.

- **data** - **[in]** Pointer to a buffer that would store the read data.
- **length** - **[in]** The number of bytes to read into the buffer pointed by *data*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_i2c_write(hailo_device device, const hailo_i2c_slave_config_t *slave_config, uint32_t ...)`

Write data to an I2C slave over a hailo device.

Parameters

- **device** - **[in]** A *hailo_device* object.
- **slave_config** - **[in]** The *hailo_i2c_slave_config_t* configuration of the slave.
- **register_address** - **[in]** The address of the register to which the data will be written.
- **data** - **[in]** A pointer to a buffer that contains the data to be written to the slave.
- **length** - **[in]** The size of *data* in bytes.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_dump_sensor_config(hailo_device device, uint8_t section_index, const char ...)`

Dump config of given section index into a csv file.

Parameters

- **device** - **[in]** A *hailo_device* object.
- **section_index** - **[in]** Flash section index to load config from. [0-7]
- **config_file_path** - **[in]** File path to dump section configuration into.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_store_sensor_config(hailo_device device, uint32_t section_index, ...)`

Store sensor configuration to Hailo chip flash memory.

Parameters

- **device** - **[in]** A *hailo_device* object.
- **section_index** - **[in]** Flash section index to write to. [0-6]
- **sensor_type** - **[in]** Sensor type.
- **reset_config_size** - **[in]** Size of reset configuration.
- **config_height** - **[in]** Configuration resolution height.
- **config_width** - **[in]** Configuration resolution width.
- **config_fps** - **[in]** Configuration FPS.
- **config_file_path** - **[in]** Sensor configuration file path.
- **config_name** - **[in]** Sensor configuration name.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_store_isp_config(hailo_device device, uint32_t reset_config_size, uint16_t ...)`

Store sensor ISP configuration to Hailo chip flash memory.

Parameters

- **device** - **[in]** A *hailo_device* object.
- **reset_config_size** - **[in]** Size of reset configuration.
- **config_height** - **[in]** Configuration resolution height.
- **config_width** - **[in]** Configuration resolution width.
- **config_fps** - **[in]** Configuration FPS.

- `isp_static_config_file_path` - **[in]** ISP static configuration file path.
- `isp_runtime_config_file_path` - **[in]** ISP runtime configuration file path.
- `config_name` - **[in]** Sensor configuration name.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_test_chip_memories(hailo_device device)`

Test chip memories using smart BIST mechanism.

Parameters `device` - **[in]** - A `hailo_device` object.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

14.2. VDevice API Functions

`hailo_status` `hailo_init_vdevice_params(hailo_vdevice_params_t *params)`

Init vdevice params with default values.

Parameters `params` - **[out]** A `hailo_vdevice_params_t` to be filled.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_create_vdevice(hailo_vdevice_params_t *params, hailo_vdevice *vdevice)`

Creates a vdevice.

Note: To release a vdevice, call the `hailo_release_vdevice` function with the returned `hailo_vdevice`.

Parameters

- `params` - **[in]** A `hailo_vdevice_params_t` (may be NULL). Can be initialized to default values using `hailo_init_vdevice_params`.
- `vdevice` - **[out]** A pointer to a `hailo_vdevice` that receives the allocated vdevice.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_configure_vdevice(hailo_vdevice vdevice, hailo_hef hef, hailo_configure_params_t ...)`

Configure the vdevice from an hef.

Parameters

- `vdevice` - **[in]** A `hailo_vdevice` object to be configured.
- `hef` - **[in]** A `hailo_hef` object to configure the `vdevice` by.
- `params` - **[in]** A `hailo_configure_params_t` (may be NULL). Can be initialized to default values using `hailo_init_configure_params_by_vdevice`.
- `network_groups` - **[out]** Array of network_groups that were loaded from the HEF file.
- `number_of_network_groups` - **[inout]** As input - the size of `network_groups` array. As output - the number of `network_groups` loaded.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_get_physical_devices(hailo_vdevice vdevice, hailo_device *devices, size_t ...)`

Gets the underlying physical devices from a vdevice.

Note: The returned physical devices are held in the scope of `vdevice`.

Parameters

- `vdevice` - **[in]** A `hailo_vdevice` object to fetch physical devices from.
- `devices` - **[out]** Array of `hailo_device` to be fetched from `vdevice`.
- `number_of_devices` - **[inout]** As input - the size of `devices` array. As output - the number of physical devices under `vdevice`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_vdevice_get_physical_devices_ids`(`hailo_vdevice` `vdevice`, `hailo_device_id_t` ...)
Gets the physical devices' ids from a `vdevice`.

Note: The returned physical devices are held in the scope of `vdevice`.

Parameters

- `vdevice` - **[in]** A `hailo_vdevice` object to fetch physical devices from.
- `devices_ids` - **[out]** Array of `hailo_device_id_t` to be fetched from `vdevice`.
- `number_of_devices` - **[inout]** As input - the size of `devices_ids` array. As output - the number of physical devices under `vdevice`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_release_vdevice`(`hailo_vdevice` `vdevice`)

Release an open `vdevice`.

Parameters `vdevice` - **[in]** A `hailo_vdevice` object to be released.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

14.3. HEF API Functions

`hailo_status` `hailo_create_hef_file`(`hailo_hef` *`hef`, const char *`file_name`)

Creates an HEF from file.

Note: To release an HEF, call the `hailo_release_hef` function with the returned `hef`.

Parameters

- `hef` - **[out]** A pointer to a `hailo_hef` that receives the allocated HEF.
- `file_name` - **[in]** The name of the hef file.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_create_hef_buffer`(`hailo_hef` *`hef`, const void *`buffer`, `size_t` `size`)

Creates an HEF from buffer.

Note: To release an HEF, call the `hailo_release_hef` function with the returned `hef`.

Parameters

- `hef` - **[out]** A pointer to a `hailo_hef` that receives the allocated HEF.

- **buffer** - **[in]** A pointer to a buffer that contains the HEF content.
- **size** - **[in]** The size in bytes of the HEF content pointed by *buffer*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status *hailo_release_hef*(*hailo_hef* *hef*)

Release an open HEF.

Parameters *hef* - **[in]** A *hailo_hef* object to be released.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status *hailo_hef_get_all_stream_infos*(*hailo_hef* *hef*, const char **name*, *hailo_stream_info_t* ...)

Gets all stream infos.

Parameters

- **hef** - **[in]** A *hailo_hef* object that contains the information.
- **name** - **[in]** The name of the network or network_group which contains the stream_infos. In case network group name is given, the function returns all stream infos of all the networks of the given network group. In case network name is given (provided by *hailo_hef_get_network_infos*), the function returns all stream infos of the given network. If NULL is passed, the function returns all the stream infos of all the networks of the first network group.
- **stream_infos** - **[out]** A pointer to a buffer of *hailo_stream_info_t* that receives the informations.
- **stream_infos_length** - **[in]** The number of *hailo_stream_info_t* elements in the buffer pointed by *stream_infos*
- **number_of_streams** - **[out]** This variable will be filled with the number of stream_infos if the function returns with *HAILO_SUCCESS* or *HAILO_INSUFFICIENT_BUFFER*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, if the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* would be returned. In any other case, returns a *hailo_status* error.

hailo_status *hailo_hef_get_stream_info_by_name*(*hailo_hef* *hef*, const char ...)

Gets stream info by name.

Parameters

- **hef** - **[in]** A *hailo_hef* object that contains the information.
- **network_group_name** - **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- **stream_name** - **[in]** The name of the stream as presented in the hef.
- **stream_direction** - **[in]** Indicates the stream direction.
- **stream_info** - **[out]** A pointer to a *hailo_stream_info_t* that receives the stream information.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status *hailo_hef_get_all_vstream_infos*(*hailo_hef* *hef*, const char **name*, ...)

Gets all virtual stream infos.

Parameters

- **hef** - **[in]** A *hailo_hef* object that contains the information.

- **name** - **[in]** The name of the network or network_group which contains the virtual stream infos. In case network group name is given, the function returns all virtual stream infos of all the networks of the given network group. In case network name is given (provided by *hailo_hef_get_network_infos*), the function returns all stream infos of the given network. If NULL is passed, the function returns all the stream infos of all the networks of the first network group.
- **vstream_infos** - **[out]** A pointer to a buffer of *hailo_stream_info_t* that receives the informations.
- **vstream_infos_count** - **[inout]** As input - the maximum amount of entries in *vstream_infos* array. As output - the actual amount of entries written if the function returns with *HAILO_SUCCESS* or the amount of entries needed if the function returns *HAILO_INSUFFICIENT_BUFFER*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status *hailo_hef_get_vstream_name_from_original_name*(*hailo_hef* hef, const char ...)

Gets vstream name from original layer name.

Parameters

- **hef** - **[in]** A *hailo_hef* object that contains the information.
- **network_group_name** - **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- **original_name** - **[in]** The original layer name as presented in the hef.
- **vstream_name** - **[out]** The name of the vstream for the provided original name, ends with NULL terminator.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status *hailo_hef_get_original_names_from_vstream_name*(*hailo_hef* hef, const char ...)

Gets all original layer names from vstream name.

Parameters

- **hef** - **[in]** A *hailo_hef* object that contains the information.
- **network_group_name** - **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- **vstream_name** - **[in]** The name of the stream as presented in the hef.
- **original_names** - **[out]** Array of *hailo_layer_name_t*, all original names linked to the provided stream (each name ends with NULL terminator).
- **original_names_length** - **[inout]** As input - the size of original_names array. As output - the number of original_layers names.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status *hailo_hef_get_vstream_names_from_stream_name*(*hailo_hef* hef, const char ...)

Gets all vstream names from stream name.

Parameters

- **hef** - **[in]** A *hailo_hef* object that contains the information.
- **network_group_name** - **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- **stream_name** - **[in]** The name of the stream as presented in the hef.
- **vstream_names** - **[out]** Array of *hailo_layer_name_t*, all vstream names linked to the provided stream (each name ends with NULL terminator).

- `vstream_names_length - [inout]` As input - the size of `vstream_names` array. As output - the number of `vstream` names.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_hef_get_stream_names_from_vstream_name(hailo_hef hef, const char ...)`

Gets all stream names from `vstream` name.

Parameters

- `hef - [in]` A `hailo_hef` object that contains the information.
- `network_group_name - [in]` The name of the `network_group` which contains the `stream_infos`. If `NULL` is passed, the first `network_group` in the HEF will be addressed.
- `vstream_name - [in]` The name of the `vstream` as presented in the `hef`.
- `stream_names - [out]` Array of `hailo_layer_name_t`, all stream names linked to the provided `vstream` (each name ends with `NULL` terminator).
- `stream_names_length - [inout]` As input - the size of `stream_names` array. As output - the number of stream names.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_hef_get_sorted_output_names(hailo_hef hef, const char ...)`

Gets sorted output names from network group name.

Parameters

- `hef - [in]` A `hailo_hef` object that contains the information.
- `network_group_name - [in]` The name of the `network_group`. If `NULL` is passed, the first `network_group` in the HEF will be addressed.
- `sorted_output_names - [out]` List of sorted outputs names.
- `sorted_output_names_count - [inout]` As input - the expected size of `sorted_output_names` list. As output - the number of `sorted_output_names`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_hef_get_bottleneck_fps(hailo_hef hef, const char *network_group_name, ...)`

Gets bottleneck fps from network group name.

Parameters

- `hef - [in]` A `hailo_hef` object that contains the information.
- `network_group_name - [in]` The name of the `network_group`. If `NULL` is passed, the first `network_group` in the HEF will be addressed.
- `bottleneck_fps - [out]` Bottleneck FPS. Note: This is not relevant in the case of multi context.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

14.4. Network Group API Functions

`hailo_status hailo_init_configure_params(hailo_hef hef, hailo_stream_interface_t stream_interface, ...)`

Init configure params with default values for a given hef.

Parameters

- `hef` - **[in]** A `hailo_hef` object to configure the `device` by.
- `stream_interface` - **[in]** A `hailo_stream_interface_t` indicating which `hailo_stream_parameters_t` to create.
- `params` - **[out]** A `hailo_configure_params_t` to be filled.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_init_configure_params_by_vdevice(hailo_hef hef, hailo_vdevice vdevice, ...)`

Init configure params with default values for a given hef by virtual device.

Parameters

- `hef` - **[in]** A `hailo_hef` object to configure the `device` by.
- `vdevice` - **[in]** A `hailo_vdevice` for which we init the params for.
- `params` - **[out]** A `hailo_configure_params_t` to be filled.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_init_configure_params_by_device(hailo_hef hef, hailo_device device, ...)`

Init configure params with default values for a given hef by device.

Parameters

- `hef` - **[in]** A `hailo_hef` object to configure the `device` by.
- `device` - **[in]** A `hailo_device` for which we init the params for.
- `params` - **[out]** A `hailo_configure_params_t` to be filled.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_init_configure_network_group_params(hailo_hef hef, ...)`

Init configure params with default values for a given hef.

Parameters

- `hef` - **[in]** A `hailo_hef` object to configure the `device` by.
- `stream_interface` - **[in]** A `hailo_stream_interface_t` indicating which `hailo_stream_parameters_t` to create.
- `network_group_name` - **[in]** The name of the `network_group` to make configure params for. If NULL is passed, the first `network_group` in the HEF will be addressed.
- `params` - **[out]** A `hailo_configure_params_t` to be filled.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_configure_device(hailo_device device, hailo_hef hef, hailo_configure_params_t ...)`

Configure the device from an hef.

Parameters

- `device` - **[in]** A `hailo_device` object to be configured.
- `hef` - **[in]** A `hailo_hef` object to configure the `device` by.
- `params` - **[in]** A `hailo_configure_params_t` (may be NULL). Can be initialized to default values using `hailo_init_configure_params_by_device`.

- **network_groups - [out]** Array of network_groups that were loaded from the HEF file.
- **number_of_network_groups - [inout]** As input - the size of network_groups array. As output - the number of network_groups loaded.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

[hailo_status](#) `hailo_wait_for_network_group_activation(hailo_configured_network_group ...)`

Block until network_group is activated, or until timeout_ms is passed.

Parameters

- **network_group - [in]** A [hailo_configured_network_group](#) to wait for activation.
- **timeout_ms - [in]** The timeout in milliseconds. If `timeout_ms` is zero, the function returns immediately. If `timeout_ms` is [HAILO_INFINITE](#), the function returns only when the event is signaled.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

[hailo_status](#) `hailo_get_network_groups_infos(hailo_hef hef, hailo_network_group_info_t *infos, ...)`

Get network group infos from an hef.

Parameters

- **hef - [in]** A [hailo_hef](#) object.
- **infos - [out]** Array of [hailo_network_group_info_t](#) to be filled.
- **number_of_infos - [inout]** As input - the size of infos array. As output - the number of infos loaded.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

[hailo_status](#) `hailo_network_group_get_all_stream_infos(hailo_configured_network_group ...)`

Gets all stream infos from a configured network group

Parameters

- **network_group - [in]** A [hailo_configured_network_group](#) object.
- **stream_infos - [out]** A pointer to a buffer of [hailo_stream_info_t](#) that receives the informations.
- **stream_infos_length - [in]** The number of [hailo_stream_info_t](#) elements in the buffer pointed by `stream_infos`
- **number_of_streams - [out]** This variable will be filled with the number of `stream_infos` if the function returns with [HAILO_SUCCESS](#) or [HAILO_INSUFFICIENT_BUFFER](#).

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, if the buffer is insufficient to hold the information a [HAILO_INSUFFICIENT_BUFFER](#) would be returned. In any other case, returns a [hailo_status](#) error.

[hailo_status](#) `hailo_network_group_get_input_stream_infos(hailo_configured_network_group ...)`

Gets all input stream infos from a configured network group

Parameters

- **network_group - [in]** A [hailo_configured_network_group](#) object.
- **stream_infos - [out]** A pointer to a buffer of [hailo_stream_info_t](#) that receives the informations.
- **stream_infos_length - [in]** The number of [hailo_stream_info_t](#) elements in the buffer pointed by `stream_infos`

- `number_of_streams` - **[out]** This variable will be filled with the number of stream_infos if the function returns with HAILO_SUCCESS or HAILO_INSUFFICIENT_BUFFER.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, if the buffer is insufficient to hold the information a `HAILO_INSUFFICIENT_BUFFER` would be returned. In any other case, returns a `hailo_status` error.

`hailo_status` `hailo_network_group_get_output_stream_infos(hailo_configured_network_group ...)`

Gets all output stream infos from a configured network group

Parameters

- `network_group` - **[in]** A `hailo_configured_network_group` object.
- `stream_infos` - **[out]** A pointer to a buffer of `hailo_stream_info_t` that receives the informations.
- `stream_infos_length` - **[in]** The number of `hailo_stream_info_t` elements in the buffer pointed by `stream_infos`
- `number_of_streams` - **[out]** This variable will be filled with the number of stream_infos if the function returns with HAILO_SUCCESS or HAILO_INSUFFICIENT_BUFFER.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, if the buffer is insufficient to hold the information a `HAILO_INSUFFICIENT_BUFFER` would be returned. In any other case, returns a `hailo_status` error.

`hailo_status` `hailo_shutdown_network_group(hailo_configured_network_group network_group)`

Shutdown a given network group. Makes sure all ongoing async operations are canceled. All sync callbacks of transfers that have not been completed will be called with status `HAILO_STREAM_ABORT`. Any resources attached to the network group may be released after function returns.

Note: Calling this function is optional, and it is used to shutdown network group while there is still ongoing inference.

Parameters `network_group` - **[in]** NetworkGroup to be shutdown.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_activate_network_group(hailo_configured_network_group network_group, ...)`

Activates hailo_device inner-resources for inference.

Parameters

- `network_group` - **[in]** NetworkGroup to be activated.
- `activation_params` - **[in]** Optional parameters for the activation (may be NULL).
- `activated_network_group_out` - **[out]** Handle for the activated network_group.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_deactivate_network_group(hailo_activated_network_group ...)`

De-activates hailo_device inner-resources for inference.

Parameters `activated_network_group` - **[in]** NetworkGroup to deactivate.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

hailo_status `hailo_get_input_stream(hailo_configured_network_group configured_network_group, ...)`

Return input stream from configured network_group by stream name.

Parameters

- `configured_network_group` - **[in]** NetworkGroup to get stream from.
- `stream_name` - **[in]** The name of the input stream to retrieve.
- `stream` - **[out]** The returned input stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_output_stream(hailo_configured_network_group configured_network_group, ...)`

Return output stream from configured network_group by stream name.

Parameters

- `configured_network_group` - **[in]** NetworkGroup to get stream from.
- `stream_name` - **[in]** The name of the output stream to retrieve.
- `stream` - **[out]** The returned output stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_latency_measurement(hailo_configured_network_group ...)`

Returns the network latency (only available if latency measurement was enabled).

Parameters

- `configured_network_group` - **[in]** NetworkGroup to get the latency measurement from.
- `network_name` - **[in]** Network name of the requested latency measurement. If NULL is passed, all the networks in the network group will be addressed, and the resulted measurement is avarage latency of all networks.
- `result` - **[out]** Output latency result.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_set_scheduler_timeout(hailo_configured_network_group ...)`

Sets the maximum time period that may pass before receiving run time from the scheduler. This will occur providing at least one send request has been sent, there is no minimum requirement for send requests, (e.g. threshold - see `set_scheduler_threshold()`).

Note: The time period is measured from the time the first frame enters the queue. It resets when a switch occurs and there are still frames in the queue.

Note: Using this function is only allowed when `scheduling_algorithm` is not *HAILO_SCHEDULING_ALGORITHM_NONE*.

Note: The default timeout is 0ms.

Note: Currently, setting the timeout for a specific network is not supported.

Note: The timeout may be ignored to prevent idle time from the device.

Parameters

- `configured_network_group` - **[in]** NetworkGroup for which to set the scheduler timeout.
- `timeout_ms` - **[in]** Timeout in milliseconds.
- `network_name` - **[in]** Network name for which to set the timeout. If NULL is passed, the timeout will be set for all the networks in the network group.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_set_scheduler_threshold(hailo_configured_network_group ...)`

Sets the scheduler threshold. This threshold sets the minimum number of send requests required before the network is considered ready to get run time from the scheduler. If at least one send request has been sent, but the threshold is not reached within a set time period (e.g. timeout - see `hailo_set_scheduler_timeout()`), the scheduler will consider the network ready regardless.

Note: Using this function is only allowed when `scheduling_algorithm` is not `HAILO_SCHEDULING_ALGORITHM_NONE`.

Note: The default threshold is 1, which means HailoRT will apply an automatic heuristic to choose the threshold.

Note: Currently, setting the threshold for a specific network is not supported.

Note: The threshold may be ignored to prevent idle time from the device.

Parameters

- `configured_network_group` - **[in]** NetworkGroup for which to set the scheduler threshold.
- `threshold` - **[in]** Threshold in number of frames.
- `network_name` - **[in]** Network name for which to set the threshold. If NULL is passed, the threshold will be set for all the networks in the network group.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_set_scheduler_priority(hailo_configured_network_group ...)`

Sets the priority of the network. When the network group scheduler will choose the next network, networks with higher priority will be prioritized in the selection. Larger number represents higher priority

Note: Using this function is only allowed when `scheduling_algorithm` is not `HAILO_SCHEDULING_ALGORITHM_NONE`.

Note: The default priority is `HAILO_SCHEDULER_PRIORITY_NORMAL`.

Note: Currently, setting the priority for a specific network is not supported.

Parameters

- `configured_network_group` - **[in]** NetworkGroup for which to set the scheduler priority.
- `priority` - **[in]** Priority as a number between HAILO_SCHEDULER_PRIORITY_MIN - HAILO_SCHEDULER_PRIORITY_MAX.
- `network_name` - **[in]** Network name for which to set the priority. If NULL is passed, the priority will be set for all the networks in the network group.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

14.5. Virtual Stream API Functions

`hailo_status` `hailo_hef_make_input_vstream_params`(`hailo_hef` hef, const char *name, bool ...)

Creates input virtual stream params linked to a network or a network group.

Parameters

- `hef` - **[in]** A `hailo_hef` object that contains the information.
- `name` - **[in]** The name of the network group or network which contains the input virtual streams. In case network group name is given, the function returns input virtual stream params of all the networks of the given network group. In case network name is given (provided by `hailo_hef_get_network_infos`), the function returns input virtual stream params of the given network. If NULL is passed, the function returns the input virtual stream params of all the networks of the first network group.
- `unused` - **[in]** Unused.
- `format_type` - **[in]** The default format type for all input virtual streams.
- `input_params` - **[out]** List of params for input virtual streams.
- `input_params_count` - **[inout]** On input: Amount of `input_params` array. On output: Will be filled with the detected amount of input vstreams on the `network` or `network_group`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_hef_make_output_vstream_params`(`hailo_hef` hef, const char *name, bool ...)

Creates output virtual stream params linked to a network or a network group.

Parameters

- `hef` - **[in]** A `hailo_hef` object that contains the information.
- `name` - **[in]** The name of the network group or network which contains the output virtual streams. In case network group name is given, the function returns output virtual stream params of all the networks of the given network group. In case network name is given (provided by `hailo_hef_get_network_infos`), the function returns output virtual stream params of the given network. If NULL is passed, the function returns the output virtual stream params of all the networks of the first network group.
- `unused` - **[in]** Unused.
- `format_type` - **[in]** The default format type for all output virtual streams.
- `output_params` - **[out]** List of params for output virtual streams.
- `output_params_count` - **[inout]** On input: Amount of `output_params` array. On output: Will be filled with the detected amount of output vstreams on the `network` or `network_group`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_make_input_vstream_params(hailo_configured_network_group network_group, ...)`

Creates input virtual stream params for a given network_group.

Parameters

- `network_group` - **[in]** Network group that owns the streams.
- `unused` - **[in]** Unused.
- `format_type` - **[in]** The default format type for all input virtual streams.
- `input_params` - **[out]** List of params for input virtual streams.
- `input_params_count` - **[inout]** On input: Amount of `input_params` array. On output: Will be filled with the detected amount of input vstreams on the `network_group`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_make_output_vstream_params(hailo_configured_network_group network_group, ...)`

Creates output virtual stream params for given network_group.

Parameters

- `network_group` - **[in]** Network group that owns the streams.
- `unused` - **[in]** Unused.
- `format_type` - **[in]** The default format type for all output virtual streams.
- `output_params` - **[out]** List of params for output virtual streams.
- `output_params_count` - **[inout]** On input: Amount of `output_params` array. On output: Will be filled with the detected amount of output vstreams on the `network_group`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_get_output_vstream_groups(hailo_configured_network_group network_group, ...)`

Gets output virtual stream groups for given network_group. The groups are split with respect to their low-level streams.

Parameters

- `network_group` - **[in]** Network group that owns the streams.
- `output_name_by_group` - **[out]** List of params for output virtual streams.
- `output_name_by_group_count` - **[inout]** On input: Amount of `output_name_by_group` array. On output: Will be filled with the detected amount of output vstreams on the `network_group`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_create_input_vstreams(hailo_configured_network_group ...)`

Creates input virtual streams.

Note: To release input virtual streams, call the `hailo_release_input_vstreams` function with the returned `input_vstreams` and `inputs_count`.

Parameters

- `configured_network_group` - **[in]** Network group that owns the streams.
- `inputs_params` - **[in]** List of input virtual stream params to create input virtual streams from.
- `inputs_count` - **[in]** How many members in `input_params`.
- `input_vstreams` - **[out]** List of input virtual streams.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_create_output_vstreams(`*hailo_configured_network_group* ...`)`

Creates output virtual streams.

Note: If not creating all output vstreams together, one should make sure all vstreams from the same group are created together. See `hailo_get_output_vstream_groups`

Note: To release output virtual streams, call the `hailo_release_output_vstreams` function with the returned `output_vstreams` and `outputs_count`.

Parameters

- `configured_network_group` - **[in]** Network group that owns the streams.
- `outputs_params` - **[in]** List of output virtual stream params to create output virtual streams from.
- `outputs_count` - **[in]** How many members in `outputs_params`.
- `output_vstreams` - **[out]** List of output virtual streams.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_get_input_vstream_frame_size(`*hailo_input_vstream* `input_vstream,` `size_t` ...`)`

Gets the size of a virtual stream's frame on the host side in bytes (the size could be affected by the format type - for example using `UINT16`, or by the data having not yet been quantized)

Parameters

- `input_vstream` - **[in]** A `hailo_input_vstream` object.
- `frame_size` - **[out]** The size of the frame on the host side in bytes.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_get_input_vstream_info(`*hailo_input_vstream* `input_vstream,` `hailo_vstream_info_t` ...`)`

Gets the `hailo_vstream_info_t` struct for the given vstream.

Parameters

- `input_vstream` - **[in]** A `hailo_input_vstream` object.
- `vstream_info` - **[out]** Will be filled with `hailo_vstream_info_t`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_get_input_vstream_user_format(`*hailo_input_vstream* `input_vstream,` ...`)`

Gets the user buffer format struct for the given vstream.

Parameters

- `input_vstream` - **[in]** A `hailo_input_vstream` object.
- `user_buffer_format` - **[out]** Will be filled with `hailo_format_t`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_get_input_vstream_quant_infos(`*hailo_input_vstream* `vstream,` ...`)`

Gets quant infos for a given input vstream.

Note: In case a single qp is present - the returned list will be of size 1. Otherwise - the returned list will be of the same length as the number of the frame's features.

Parameters

- `vstream` - **[in]** A `hailo_input_vstream` object.
- `quant_infos` - **[out]** A pointer to a buffer of `hailo_quant_info_t` that will be filled with quant infos.
- `quant_infos_count` - **[inout]** As input - the maximum amount of entries in `quant_infos` array. As output - the actual amount of entries written if the function returns with `HAILO_SUCCESS` or the amount of entries needed if the function returns `HAILO_INSUFFICIENT_BUFFER`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_get_output_vstream_quant_infos`(`hailo_output_vstream` `vstream`, ...)

Gets quant infos for a given output vstream.

Note: In case a single qp is present - the returned list will be of size 1. Otherwise - the returned list will be of the same length as the number of the frame's features.

Parameters

- `vstream` - **[in]** A `hailo_output_vstream` object.
- `quant_infos` - **[out]** A pointer to a buffer of `hailo_quant_info_t` that will be filled with quant infos.
- `quant_infos_count` - **[inout]** As input - the maximum amount of entries in `quant_infos` array. As output - the actual amount of entries written if the function returns with `HAILO_SUCCESS` or the amount of entries needed if the function returns `HAILO_INSUFFICIENT_BUFFER`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_get_output_vstream_frame_size`(`hailo_output_vstream` `output_vstream`, ...)

Gets the size of a virtual stream's frame on the host side in bytes (the size could be affected by the format type - for example using `UINT16`, or by the data having not yet been quantized)

Parameters

- `output_vstream` - **[in]** A `hailo_output_vstream` object.
- `frame_size` - **[out]** The size of the frame on the host side in bytes.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_get_output_vstream_info`(`hailo_output_vstream` `output_vstream`, ...)

Gets the `hailo_vstream_info_t` struct for the given vstream.

Parameters

- `output_vstream` - **[in]** A `hailo_output_vstream` object.
- `vstream_info` - **[out]** Will be filled with `hailo_vstream_info_t`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_get_output_vstream_user_format`(`hailo_output_vstream` `output_vstream`, ...)

Gets the user buffer format struct for the given vstream.

Parameters

- `output_vstream` - **[in]** A `hailo_output_vstream` object.
- `user_buffer_format` - **[out]** Will be filled with `hailo_format_t`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_get_vstream_frame_size(hailo_vstream_info_t *vstream_info, hailo_format_t ...)`

Gets the size of a virtual stream's frame in bytes (the size could be affected by the format type - for example using `UINT16`, or by the data having not yet been quantized)

Parameters

- `vstream_info` - **[in]** A `hailo_vstream_info_t` object.
- `user_buffer_format` - **[in]** A `hailo_format_t` object.
- `frame_size` - **[out]** The size of the frame on the host side in bytes.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_vstream_write_raw_buffer(hailo_input_vstream input_vstream, const void ...)`

Writes buffer to hailo device via input virtual stream `input_vstream`.

Parameters

- `input_vstream` - **[in]** A `hailo_input_vstream` object.
- `buffer` - **[in]** A pointer to a buffer to be sent. The buffer format comes from the vstream's `format` (Can be obtained using `hailo_get_input_vstream_user_format`) and the shape comes from `shape` inside `hailo_vstream_info_t` (Can be obtained using `hailo_get_input_vstream_info`).
- `buffer_size` - **[in]** `buffer` buffer size in bytes. The size is expected to be the size returned from `hailo_get_input_vstream_frame_size`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_vstream_write_pix_buffer(hailo_input_vstream input_vstream, const ...)`

Writes the buffer to hailo device via input virtual stream `input_vstream`.

Note: Currently only support memory_type field of buffer to be `HAILO_PIX_BUFFER_MEMORY_TYPE_USERPTR`.

Parameters

- `input_vstream` - **[in]** A `hailo_input_vstream` object.
- `buffer` - **[in]** A pointer to the buffer containing pointers to the planes to where the data to be sent to the device is stored.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_flush_input_vstream(hailo_input_vstream input_vstream)`

Blocks until the pipeline buffers of `input_vstream` are flushed.

Parameters `input_vstream` - **[in]** The input vstream to flush.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_vstream_read_raw_buffer(hailo_output_vstream output_vstream, void *buffer, ...)`

Reads data from hailo device via `output_vstream` into `dst`.

Parameters

- `output_vstream` - **[in]** A `hailo_output_vstream` object.
- `buffer` - **[in]** A pointer to the received buffer. The buffer format comes from the vstream's `format` (Can be obtained using `hailo_get_output_vstream_user_format`) and the shape comes from `shape` or `nms_shape` inside `hailo_vstream_info_t` (Can be obtained using `hailo_get_output_vstream_info`).
- `buffer_size` - **[in]** `dst` buffer size in bytes. The size is expected to be the size returned from `hailo_get_output_vstream_frame_size`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_vstream_set_nms_score_threshold`(`hailo_output_vstream` `output_vstream`, ...)

Set NMS score threshold, used for filtering out candidates. Any box with score<TH is suppressed.

Note: This function will fail in cases where the output vstream has no NMS operations on the CPU.

Parameters

- `output_vstream` - **[in]** A `hailo_output_vstream` object.
- `threshold` - **[in]** NMS score threshold to set.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_vstream_set_nms_iou_threshold`(`hailo_output_vstream` `output_vstream`, ...)

Set NMS intersection over union overlap Threshold, used in the NMS iterative elimination process where potential duplicates of detected items are suppressed.

Note: This function will fail in cases where the output vstream has no NMS operations on the CPU.

Parameters

- `output_vstream` - **[in]** A `hailo_output_vstream` object.
- `threshold` - **[in]** NMS IoU threshold to set.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_vstream_set_nms_max_proposals_per_class`(`hailo_output_vstream` ...)

Set a limit for the maximum number of boxes per class.

Note: This function will fail in cases where the output vstream has no NMS operations on the CPU.

Parameters

- `output_vstream` - **[in]** A `hailo_output_vstream` object.
- `max_proposals_per_class` - **[in]** NMS max proposals per class to set.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_release_input_vstreams`(`const hailo_input_vstream` *`input_vstreams`, `size_t` ...)

Release input virtual streams.

Parameters

- `input_vstreams` - **[in]** List of input virtual streams to be released.
- `inputs_count` - **[in]** The amount of elements in `input_params`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_release_output_vstreams`(`const hailo_output_vstream` *`output_vstreams`, `size_t` ...)

Release output virtual streams.

Parameters

- `output_vstreams` - **[in]** List of output virtual streams to be released.
- `outputs_count` - **[in]** The amount of elements in `output_vstreams`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_clear_input_vstreams`(const `hailo_input_vstream` **input_vstreams*, `size_t` ...)

Clears the pipeline buffers of each vstream in *input_vstreams*.

Parameters

- `input_vstreams` - **[in]** List of input virtual streams to be cleared.
- `inputs_count` - **[in]** The amount of elements in *input_params*.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_clear_output_vstreams`(const `hailo_output_vstream` **output_vstreams*, `size_t` ...)

Clears the pipeline buffers of each vstream in *output_vstreams*.

Note: If not all output vstreams from the same group are passed together, it will cause an **undefined behavior**. See `hailo_get_output_vstream_groups`, to get the output vstreams' groups.

Parameters

- `output_vstreams` - **[in]** List of output virtual streams to be cleared.
- `outputs_count` - **[in]** The amount of elements in *output_vstreams*.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_infer`(`hailo_configured_network_group` *configured_network_group*, ...)

Run simple inference using vstreams pipelines.

Note: *configured_network_group* should be activated before calling this function.

Note: the size of each element in *input_buffers* and *output_buffers* should match the product of *frames_count* and the frame size of the matching `hailo_input_vstream` / `hailo_output_vstream`.

Parameters

- `configured_network_group` - **[in]** A `hailo_configured_network_group` to run the inference on.
- `inputs_params` - **[in]** Array of input virtual stream params, indicates *input_buffers* format.
- `input_buffers` - **[in]** Array of `hailo_stream_raw_buffer_by_name_t`. Ths input dataset of the inference.
- `inputs_count` - **[in]** The amount of elements in *inputs_params* and *input_buffers*.
- `outputs_params` - **[in]** Array of output virtual stream params, indicates *output_buffers* format.
- `output_buffers` - **[out]** Array of `hailo_stream_raw_buffer_by_name_t`. Ths results of the inference.
- `outputs_count` - **[in]** The amount of elements in *outputs_params* and *output_buffers*.
- `frames_count` - **[in]** The amount of inferred frames.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

14.6. Stream API Functions

`hailo_status` `hailo_set_input_stream_timeout`(`hailo_input_stream` stream, `uint32_t` timeout_ms)

Set new timeout value to an input stream

Parameters

- `stream` - **[in]** A `hailo_input_stream` object to get the new timeout value.
- `timeout_ms` - **[in]** the new timeout value to be set.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_set_output_stream_timeout`(`hailo_output_stream` stream, `uint32_t` timeout_ms)

Set new timeout value to an output stream

Parameters

- `stream` - **[in]** A `hailo_output_stream` object to get the new timeout value.
- `timeout_ms` - **[in]** the new timeout value to be set.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`size_t` `hailo_get_input_stream_frame_size`(`hailo_input_stream` stream)

Gets the size of a stream's frame on the host side in bytes

Parameters `stream` - **[in]** A `hailo_input_stream` object.

Returns The size of the frame on the host side in bytes

`size_t` `hailo_get_output_stream_frame_size`(`hailo_output_stream` stream)

Gets the size of a stream's frame on the host side in bytes

Parameters `stream` - **[in]** A `hailo_output_stream` object.

Returns The size of the frame on the host side in bytes

`hailo_status` `hailo_get_input_stream_info`(`hailo_input_stream` stream, `hailo_stream_info_t` ...)

Gets stream info from the given input stream

Parameters

- `stream` - **[in]** A `hailo_input_stream` object.
- `stream_info` - **[out]** An output `hailo_stream_info_t`.

Returns The size of the frame on the host side in bytes

`hailo_status` `hailo_get_output_stream_info`(`hailo_output_stream` stream, `hailo_stream_info_t` ...)

Gets stream info from the given output stream

Parameters

- `stream` - **[in]** A `hailo_input_stream` object.
- `stream_info` - **[out]** An output `hailo_stream_info_t`.

Returns The size of the frame on the host side in bytes

`hailo_status` `hailo_get_input_stream_quant_infos`(`hailo_input_stream` stream, `hailo_quant_info_t` ...)

Gets quant infos for a given input stream.

Note: In case a single qp is present - the returned list will be of size 1. Otherwise - the returned list will be of the same length as the number of the frame's features.

Parameters

- **stream - [in]** A *hailo_input_stream* object.
- **quant_infos - [out]** A pointer to a buffer of *hailo_quant_info_t* that will be filled with quant infos.
- **quant_infos_count - [inout]** As input - the maximum amount of entries in *quant_infos* array. As output - the actual amount of entries written if the function returns with *HAILO_SUCCESS* or the amount of entries needed if the function returns *HAILO_INSUFFICIENT_BUFFER*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status *hailo_get_output_stream_quant_infos*(*hailo_output_stream* stream, ...)

Gets quant infos for a given output stream.

Note: In case a single qp is present - the returned list will be of size 1. Otherwise - the returned list will be of the same length as the number of the frame's features.

Parameters

- **stream - [in]** A *hailo_output_stream* object.
- **quant_infos - [out]** A pointer to a buffer of *hailo_quant_info_t* that will be filled with quant infos.
- **quant_infos_count - [inout]** As input - the maximum amount of entries in *quant_infos* array. As output - the actual amount of entries written if the function returns with *HAILO_SUCCESS* or the amount of entries needed if the function returns *HAILO_INSUFFICIENT_BUFFER*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status *hailo_stream_read_raw_buffer*(*hailo_output_stream* stream, void *buffer, *size_t* size)

Synchronously reads data from a stream.

Note: The output buffer format comes from the *format* field inside *hailo_stream_info_t* and the shape comes from the *hw_shape* field inside *hailo_stream_info_t*.

Note: *size* is expected to be *stream_info.hw_frame_size*.

Parameters

- **stream - [in]** A *hailo_output_stream* object.
- **buffer - [in]** A pointer to a buffer that receives the data read from *stream*.
- **size - [in]** The amount of bytes to read, should be the frame size.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status *hailo_stream_write_raw_buffer*(*hailo_input_stream* stream, const void *buffer, *size_t* ...)

Synchronously writes all data to a stream.

Note: The input buffer format comes from the *format* field inside *hailo_stream_info_t* and the shape comes from the *hw_shape* field inside *hailo_stream_info_t*.

Note: *size* is expected to be `stream_info.hw_frame_size`.

Parameters

- `stream` - **[in]** A `hailo_input_stream` object.
- `buffer` - **[in]** A pointer to a buffer that contains the data to be written to `stream`.
- `size` - **[in]** The amount of bytes to write.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_stream_wait_for_async_output_ready(hailo_output_stream stream, size_t ...)`

Waits until the stream is ready to launch a new `hailo_stream_read_raw_buffer_async` operation. Each stream has a limited-size queue for ongoing transfers. You can retrieve the queue size for the given stream by calling `hailo_output_stream_get_async_max_queue_size`.

Parameters

- `stream` - **[in]** A `hailo_output_stream` object.
- `transfer_size` - **[in]** Must be the result of `hailo_get_output_stream_frame_size` for the given stream.
- `timeout_ms` - **[in]** Amount of time to wait until the stream is ready in milliseconds.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise:

- If `timeout_ms` has passed and the stream is not ready, returns `HAILO_TIMEOUT`.
- In any other error case, returns `hailo_status` error.

`hailo_status` `hailo_stream_wait_for_async_input_ready(hailo_input_stream stream, size_t ...)`

Waits until the stream is ready to launch a new `hailo_stream_write_raw_buffer_async` operation. Each stream has a limited-size queue for ongoing transfers. You can retrieve the queue size for the given stream by calling `hailo_input_stream_get_async_max_queue_size`.

Parameters

- `stream` - **[in]** A `hailo_input_stream` object.
- `transfer_size` - **[in]** Must be the result of `hailo_get_input_stream_frame_size` for the given stream.
- `timeout_ms` - **[in]** Amount of time to wait until the stream is ready in milliseconds.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise:

- If `timeout_ms` has passed and the stream is not ready, returns `HAILO_TIMEOUT`.
- In any other error case, returns `hailo_status` error.

`hailo_status` `hailo_output_stream_get_async_max_queue_size(hailo_output_stream stream, ...)`

Returns the maximum amount of frames that can be simultaneously read from the stream (by `hailo_stream_read_raw_buffer_async` calls) before any one of the read operations is complete, as signified by `user_callback` being called.

Parameters

- `stream` - **[in]** A `hailo_output_stream` object.
- `queue_size` - **[out]** Returns value of the queue

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_input_stream_get_async_max_queue_size(hailo_input_stream stream, ...)`

Returns the maximum amount of frames that can be simultaneously written to the stream (by `hailo_stream_write_raw_buffer_async` calls) before any one of the write operations is complete, as signified by `user_callback` being called.

Parameters

- `stream` - **[in]** A `hailo_input_stream` object.
- `queue_size` - **[out]** Returns value of the queue

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_stream_read_raw_buffer_async(hailo_output_stream stream, void *buffer, ...)`

Reads into `buffer` from the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., `hailo_stream_read_raw_buffer_async` returns `HAILO_SUCCESS`), the deferred operation has been initiated. Until `user_callback` is called, the user cannot change or delete `buffer`.
- If the function call fails (i.e., `hailo_stream_read_raw_buffer_async` returns a status other than `HAILO_SUCCESS`), the deferred operation will not be initiated and `user_callback` will not be invoked. The user is free to change or delete `buffer`.
- `user_callback` is triggered upon successful completion or failure of the deferred operation. The callback receives a `hailo_stream_read_async_completion_info_t` object containing a pointer to the transferred buffer (`buffer_addr`) and the transfer status (`status`). If the operation has completed successfully, the contents of `buffer` will have been updated by the read operation.

Note: `user_callback` should execute as quickly as possible.

Note: The output buffer format comes from the `format` field inside `hailo_stream_info_t` and the shape comes from the `hw_shape` field inside `hailo_stream_info_t`.

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Parameters

- `stream` - **[in]** A `hailo_output_stream` object.
- `buffer` - **[in]** The buffer to be read into. The buffer must be aligned to the system page size.
- `size` - **[in]** The size of the given buffer, expected to be the result of `hailo_get_output_stream_frame_size`.
- `user_callback` - **[in]** The callback that will be called when the transfer is complete or has failed.
- `opaque` - **[in]** Optional pointer to user-defined context (may be NULL if not desired).

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise:

- If the stream queue is full, returns `HAILO_QUEUE_IS_FULL`. In this case, please wait until `user_callback` is called on previous reads, or call `hailo_stream_wait_for_async_output_ready`. The size of the queue can be determined by calling `hailo_output_stream_get_async_max_queue_size`.
- In any other error case, returns a `hailo_status` error.

`hailo_status hailo_stream_write_raw_buffer_async(hailo_input_stream stream, const void ...)`

Writes the contents of *buffer* to the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., `hailo_stream_write_raw_buffer_async` returns `HAILO_SUCCESS`), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., `hailo_stream_write_raw_buffer_async` returns a status other than `HAILO_SUCCESS`), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a `hailo_stream_write_async_completion_info_t` object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*).

Note: *user_callback* should run as quickly as possible.

Note: The input buffer format comes from the *format* field inside `hailo_stream_info_t` and the shape comes from the *hw_shape* field inside `hailo_stream_info_t`.

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Parameters

- *stream* - **[in]** A `hailo_input_stream` object.
- *buffer* - **[in]** The buffer to be written. The buffer must be aligned to the system page size.
- *size* - **[in]** The size of the given buffer, expected to be the result of `hailo_get_input_stream_frame_size`.
- *user_callback* - **[in]** The callback that will be called when the transfer is complete or has failed.
- *opaque* - **[in]** Optional pointer to user-defined context (may be NULL if not desired).

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise:

- If the stream queue is full, returns `HAILO_QUEUE_IS_FULL`. In this case please wait until *user_callback* is called on previous writes, or call `hailo_stream_wait_for_async_input_ready`. The size of the queue can be determined by calling `hailo_input_stream_get_async_max_queue_size`.
- In any other error case, returns a `hailo_status` error.

`size_t hailo_get_host_frame_size(const hailo_stream_info_t *stream_info, const ...)`

Gets the size of a stream's frame on the host side in bytes (the size could be affected by the format type - for example using `UINT16`, or by the data having not yet been quantized)

Parameters

- *stream_info* - **[in]** The stream's info represented by `hailo_stream_info_t`
- *transform_params* - **[in]** Host side transform parameters

Returns The size of the frame on the host side in bytes

14.7. Transformation API Functions

`hailo_status` `hailo_create_input_transform_context`(const `hailo_stream_info_t` *`stream_info`, ...)

Creates an input transform_context object. Allocates all necessary buffers used for the transformation (pre-process).

Note: To release the transform_context, call the `hailo_release_input_transform_context` function with the returned `hailo_input_transform_context`.

Parameters

- `stream_info` - **[in]** - A `hailo_stream_info_t` object
- `transform_params` - **[in]** - A `hailo_transform_params_t` user transformation parameters.
- `transform_context` - **[out]** - A `hailo_input_transform_context`

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_create_input_transform_context_by_stream`(`hailo_input_stream` ...)

Creates an input transform_context object. Allocates all necessary buffers used for the transformation (pre-process).

Note: To release the transform_context, call the `hailo_release_input_transform_context` function with the returned `hailo_input_transform_context`.

Parameters

- `stream` - **[in]** A `hailo_input_stream` object
- `transform_params` - **[in]** A `hailo_transform_params_t` user transformation parameters.
- `transform_context` - **[out]** A `hailo_input_transform_context`

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_release_input_transform_context`(`hailo_input_transform_context` ...)

Releases a transform_context object including all allocated buffers.

Parameters `transform_context` - **[in]** - A `hailo_input_transform_context` object.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_is_input_transformation_required2`(const `hailo_3d_image_shape_t` ...)

Check whether or not a transformation is needed - for quant_info per feature case.

Note: In case `transformation_required` is false, the src frame is ready to be sent to HW without any transformation.

Parameters

- `src_image_shape` - **[in]** The shape of the src buffer (host shape).
- `src_format` - **[in]** The format of the src buffer (host format).
- `dst_image_shape` - **[in]** The shape of the dst buffer (hw shape).

- `dst_format` - **[in]** The format of the dst buffer (hw format).
- `quant_infos` - **[in]** A pointer to an array of `hailo_quant_info_t` object containing quantization information.
- `quant_infos_count` - **[in]** The number of `hailo_quant_info_t` elements pointed by `quant_infos`. `quant_infos_count` should be equals to either 1, or `src_image_shape.features`
- `transformation_required` - **[out]** Indicates whether or not a transformation is needed.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status` `hailo_transform_frame_by_input_transform_context`(`hailo_input_transform_context ...`)

Transforms an input frame pointed to by `src` directly to the buffer pointed to by `dst`.

Warning: The buffers must not overlap.

Parameters

- `transform_context` - **[in]** A `hailo_input_transform_context`.
- `src` - **[in]** A pointer to a buffer to be transformed.
- `src_size` - **[in]** The number of bytes to transform. This number must be equal to the input `host_frame_size`, and less than or equal to the size of `src` buffer.
- `dst` - **[out]** A pointer to a buffer that receives the transformed data.
- `dst_size` - **[in]** The number of bytes in `dst` buffer. This number must be equal to the input `hw_frame_size`, and less than or equal to the size of `dst` buffer.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_is_output_transformation_required2`(const `hailo_3d_image_shape_t ...`)

Check whether or not a transformation is needed - for `quant_info` per feature case.

Note: In case `transformation_required` is false, the `src` frame is already in the required format without any transformation.

Parameters

- `src_image_shape` - **[in]** The shape of the src buffer (hw shape).
- `src_format` - **[in]** The format of the src buffer (hw format).
- `dst_image_shape` - **[in]** The shape of the dst buffer (host shape).
- `dst_format` - **[in]** The format of the dst buffer (host format).
- `quant_infos` - **[in]** A pointer to an array of `hailo_quant_info_t` object containing quantization information.
- `quant_infos_count` - **[in]** The number of `hailo_quant_info_t` elements pointed by `quant_infos`. `quant_infos_count` should be equals to either 1, or `dst_image_shape.features`.
- `transformation_required` - **[out]** Indicates whether or not a transformation is needed.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

hailo_status `hailo_create_output_transform_context(const hailo_stream_info_t *stream_info, ...)`

Creates an output transform_context object. Allocates all necessary buffers used for the transformation (post-process).

Note: To release the transform_context, call the `hailo_release_output_transform_context` function with the returned `hailo_output_transform_context`.

Parameters

- `stream_info` - **[in]** - A `hailo_stream_info_t` object
- `transform_params` - **[in]** - A `hailo_transform_params_t` user transformation parameters.
- `transform_context` - **[out]** - A `hailo_output_transform_context`

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

hailo_status `hailo_create_output_transform_context_by_stream(hailo_output_stream ...)`

Creates an output transform_context object. Allocates all necessary buffers used for the transformation (post-process).

Note: To release the transform_context, call the `hailo_release_output_transform_context` function with the returned `hailo_output_transform_context`.

Parameters

- `stream` - **[in]** A `hailo_output_stream` object
- `transform_params` - **[in]** A `hailo_transform_params_t` user transformation parameters.
- `transform_context` - **[out]** A `hailo_output_transform_context`

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

hailo_status `hailo_release_output_transform_context(hailo_output_transform_context ...)`

Releases a transform_context object including all allocated buffers.

Parameters `transform_context` - **[in]** - A `hailo_output_transform_context` object.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

hailo_status `hailo_transform_frame_by_output_transform_context(hailo_output_transform_context ...)`

Transforms an output frame pointed to by `src` directly to the buffer pointed to by `dst`.

Warning: The buffers must not overlap.

Parameters

- `transform_context` - **[in]** A `hailo_output_transform_context`.
- `src` - **[in]** A pointer to a buffer to be transformed.
- `src_size` - **[in]** The number of bytes to transform. This number must be equal to the output `hw_frame_size`, and less than or equal to the size of `src` buffer.
- `dst` - **[out]** A pointer to a buffer that receives the transformed data.
- `dst_size` - **[in]** The number of bytes in `dst` buffer. This number must be equal to the output `host_frame_size`, and less than or equal to the size of `dst` buffer.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_is_qp_valid(const hailo_quant_info_t quant_info, bool *is_qp_valid)`

Returns whether or not qp is valid

Note: QP will be invalid in case HEF file was compiled with multiple QPs, and then the user will try working with API for single QP. For example - if HEF was compiled with multiple QPs and then the user calls `hailo_get_input_stream_info`, The `hailo_quant_info_t` object of the `hailo_stream_info_t` object will be invalid.

Parameters

- `quant_info` - **[in]** A `hailo_quant_info_t` object.
- `is_qp_valid` - **[out]** Indicates whether or not qp is valid.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status hailo_create_demuxer_by_stream(hailo_output_stream stream, const ...)`

Creates an demuxer for the given mux stream. Allocates all necessary buffers used for the demux process.

Note: To release the demuxer, call the `hailo_release_output_demuxer` function with the returned `hailo_output_demuxer`.

Parameters

- `stream` - **[in]** - A `hailo_output_stream` object
- `demux_params` - **[in]** - A `hailo_demux_params_t` user demux parameters.
- `demuxer` - **[out]** - A `hailo_output_demuxer`, used to transform output frames

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status hailo_release_output_demuxer(hailo_output_demuxer demuxer)`

Releases a demuxer object.

Parameters `demuxer` - **[in]** - A `hailo_output_demuxer` object.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status hailo_demux_raw_frame_by_output_demuxer(hailo_output_demuxer demuxer, ...)`

Demultiplexing an output frame pointed to by `src` directly to the buffers pointed to by `raw_buffers`.

Note: The order of `raw_buffers` should be the same as returned from the function `'hailo_get_mux_infos_by_output_demuxer()'`.

Parameters

- `demuxer` - **[in]** A `hailo_output_demuxer` object used for the demuxing.
- `src` - **[in]** A pointer to a buffer to be demultiplexed.
- `src_size` - **[in]** The number of bytes to demultiplexed. This number must be equal to the `hw_frame_size`, and less than or equal to the size of `src` buffer.
- `raw_buffers` - **[inout]** A pointer to an array of `hailo_stream_raw_buffer_t` that receives the demultiplexed data read from the `stream`.
- `raw_buffers_count` - **[in]** The number of `hailo_stream_raw_buffer_t` elements in the array pointed to by `raw_buffers`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_demux_by_name_raw_frame_by_output_demuxer(hailo_output_demuxer ...)`

Demultiplexing an output frame pointed to by `src` directly to the buffers pointed to by `raw_buffers_by_name`.

Parameters

- `demuxer` - **[in]** A `hailo_output_demuxer` object used for the demuxing.
- `src` - **[in]** A pointer to a buffer to be demultiplexed.
- `src_size` - **[in]** The number of bytes to demultiplexed. This number must be equal to the `hw_frame_size`, and less than or equal to the size of `src` buffer.
- `raw_buffers_by_name` - **[inout]** A pointer to an array of `hailo_stream_raw_buffer_by_name_t` that receives the demultiplexed data read from the `stream`. `hailo_stream_raw_buffer_by_name_t::name` should be filled with the demuxes names.
- `raw_buffers_count` - **[in]** The number of `hailo_stream_raw_buffer_by_name_t` elements in the array pointed to by `raw_buffers_by_name`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_get_mux_infos_by_output_demuxer(hailo_output_demuxer demuxer, ...)`

Gets all multiplexed stream infos.

Parameters

- `demuxer` - **[in]** A `hailo_output_demuxer` object.
- `stream_infos` - **[out]** A pointer to a buffer of `hailo_stream_info_t` that receives the information.
- `number_of_streams` - **[inout]** The maximum amount of streams_info to fill. This variable will be filled with the actual number of multiplexed stream_infos. If the buffer is insufficient to hold the information this variable will be set to the requested value, `HAILO_INSUFFICIENT_BUFFER` would be returned.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `hailo_fuse_nms_frames(const hailo_nms_fuse_input_t *nms_fuse_inputs, uint32_t ...)`

Fuse multiple defused NMS buffers pointed by `nms_fuse_inputs` to the buffer pointed to by `fused_buffer`. This function should be called on `nms_fuse_inputs` after receiving them from HW, and before transformation. This function expects `nms_fuse_inputs` to be ordered by their `class_group_index` (lowest to highest).

Parameters

- `nms_fuse_inputs` - **[in]** Array of `hailo_nms_fuse_input_t` structs which contain the buffers to be fused.
- `inputs_count` - **[in]** How many members in `nms_fuse_inputs`.
- `fused_buffer` - **[out]** A pointer to a buffer which will contain the fused buffer.
- `fused_buffer_size` - **[in]** The fused buffer size.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

14.8. Power Measurement API Functions

`hailo_status hailo_power_measurement(hailo_device device, hailo_dvm_options_t dvm, ...)`

Perform a single power measurement.

Parameters

- `device` - **[in]** A `hailo_device` object.
- `dvm` - **[in]** Which DVM will be measured. Default (`HAILO_DVM_OPTIONS_AUTO`) will be different according to the board:
 - Default (`HAILO_DVM_OPTIONS_AUTO`) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums `HAILO_DVM_OPTIONS_VDD_CORE`, `HAILO_DVM_OPTIONS_MIPI_AVDD` and `HAILO_DVM_OPTIONS_AVDD_H`. Only `HAILO_POWER_MEASUREMENT_TYPES_POWER` can be measured with this option.
 - Default (`HAILO_DVM_OPTIONS_AUTO`) for platforms supporting current monitoring (such as M.2 and mPCIe): `OVERCURRENT_PROTECTION`.
- `measurement_type` - **[in]** The type of the measurement. Choosing `HAILO_POWER_MEASUREMENT_TYPES_AUTO` will select the default value according to the supported features.
- `measurement` - **[out]** The measured value. Measured units are determined due to `hailo_power_measurement_types_t`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_start_power_measurement(hailo_device device, hailo_averaging_factor_t ...)`

Start performing a long power measurement.

Parameters

- `device` - **[in]** A `hailo_device` object.
- `averaging_factor` - **[in]** Number of samples per time period, sensor configuration value.
- `sampling_period` - **[in]** Related conversion time, sensor configuration value. The sensor samples the power every `sampling_period` {us} and averages every `averaging_factor` samples. The sensor provides a new value every: $(2 * sampling_period * averaging_factor)$ {ms}. The firmware wakes up every `interval_milliseconds` {ms} and checks the sensor. If there is a new value to read from the sensor, the firmware reads it. Note that the average calculated by the firmware is 'average of averages', because it averages values that have already been averaged by the sensor.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status hailo_set_power_measurement(hailo_device device, hailo_measurement_buffer_index_t ...)`

Set parameters for long power measurement.

Parameters

- `device` - **[in]** A `hailo_device` object.
- `buffer_index` - **[in]** A `hailo_measurement_buffer_index_t` represents the buffer on the firmware the data would be saved at. Should match the one passed to `hailo_get_power_measurement`.
- `dvm` - **[in]** Which DVM will be measured. Default (`HAILO_DVM_OPTIONS_AUTO`) will be different according to the board:
 - Default (`HAILO_DVM_OPTIONS_AUTO`) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums `HAILO_DVM_OPTIONS_VDD_CORE`, `HAILO_DVM_OPTIONS_MIPI_AVDD` and `HAILO_DVM_OPTIONS_AVDD_H`. Only `HAILO_POWER_MEASUREMENT_TYPES_POWER` can be measured with this option.

- Default (*HAILO_DVM_OPTIONS_AUTO*) for platforms supporting current monitoring (such as M.2 and mPCIe): OVERCURRENT_PROTECTION.
- **measurement_type** - **[in]** The type of the measurement. Choosing *HAILO_POWER_MEASUREMENT_TYPES_AUTO* will select the default value according to the supported features.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status *hailo_get_power_measurement*(*hailo_device* device, *hailo_measurement_buffer_index_t* ...)

Read measured power from a long power measurement

Parameters

- **device** - **[in]** A *hailo_device* object.
- **buffer_index** - **[in]** A *hailo_measurement_buffer_index_t* represents the buffer on the firmware the data would be saved at. Should match the one passed to *hailo_set_power_measurement*.
- **should_clear** - **[in]** Flag indicating if the results saved at the firmware will be deleted after reading.
- **measurement_data** - **[out]** The measurement data, *hailo_power_measurement_data_t*. Measured units are determined due to *hailo_power_measurement_types_t* passed to *hailo_set_power_measurement*

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status *hailo_stop_power_measurement*(*hailo_device* device)

Stop performing a long power measurement.

Parameters **device** - **[in]** A *hailo_device* object.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

14.9. Multiple Networks API Functions

hailo_status *hailo_hef_get_network_infos*(*hailo_hef* hef, const char *network_group_name, ...)

Gets all network infos under a given network group

Parameters

- **hef** - **[in]** A *hailo_hef* object that contains the information.
- **network_group_name** - **[in]** Name of the network_group to get the network infos by. If NULL is passed, the first network_group in the HEF will be addressed.
- **networks_infos** - **[out]** A pointer to a buffer of *hailo_network_info_t* that receives the informations.
- **number_of_networks** - **[inout]** As input - the maximum amount of entries in *hailo_network_info_t* array. As output - the actual amount of entries written if the function returns with *HAILO_SUCCESS* or the amount of entries needed if the function returns *HAILO_INSUFFICIENT_BUFFER*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, if the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* would be returned. In any other case, returns a *hailo_status* error.

hailo_status *hailo_get_network_infos*(*hailo_configured_network_group* network_group, ...)

Gets all network infos under a given network group

Parameters

- **network_group - [in]** A [hailo_configured_network_group](#) object to get the network infos from.
- **networks_infos - [out]** A pointer to a buffer of [hailo_network_info_t](#) that receives the informations.
- **number_of_networks - [inout]** As input - the maximum amount of entries in [hailo_network_info_t](#) array. As output - the actual amount of entries written if the function returns with [HAILO_SUCCESS](#) or the amount of entries needed if the function returns [HAILO_INSUFFICIENT_BUFFER](#).

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, if the buffer is insufficient to hold the information a [HAILO_INSUFFICIENT_BUFFER](#) would be returned. In any other case, returns a [hailo_status](#) error.

14.10. Other API Definitions

```
enum hailo_status
```

Values:

enumerator HAILO_SUCCESS

Success - No error

enumerator HAILO_UNINITIALIZED

No error code was initialized

enumerator HAILO_INVALID_ARGUMENT

Invalid argument passed to function

enumerator HAILO_OUT_OF_HOST_MEMORY

Cannot allocate more memory at host

enumerator HAILO_TIMEOUT

Received a timeout

enumerator HAILO_INSUFFICIENT_BUFFER

Buffer is insufficient

enumerator HAILO_INVALID_OPERATION

Invalid operation

enumerator HAILO_NOT_IMPLEMENTED

Code has not been implemented

enumerator HAILO_INTERNAL_FAILURE

Unexpected internal failure

enumerator HAILO_DATA_ALIGNMENT_FAILURE

Data is not aligned

enumerator HAILO_CHUNK_TOO_LARGE

Chunk too large

enumerator HAILO_INVALID_LOGGER_LEVEL

Used non-compiled level

enumerator HAILO_CLOSE_FAILURE

Failed to close fd

enumerator HAILO_OPEN_FILE_FAILURE

Failed to open file

enumerator HAILO_FILE_OPERATION_FAILURE

File operation failure

enumerator HAILO_UNSUPPORTED_CONTROL_PROTOCOL_VERSION

Unsupported control protocol version

enumerator HAILO_UNSUPPORTED_FW_VERSION

Unsupported firmware version

enumerator HAILO_INVALID_CONTROL_RESPONSE

Invalid control response

enumerator HAILO_FW_CONTROL_FAILURE

Control failed in firmware

enumerator HAILO_ETH_FAILURE

Ethernet operation has failed

enumerator HAILO_ETH_INTERFACE_NOT_FOUND

Ethernet interface not found

enumerator HAILO_ETH_RECV_FAILURE

Ethernet failed at recv operation

enumerator HAILO_ETH_SEND_FAILURE

Ethernet failed at send operation

enumerator HAILO_INVALID_FIRMWARE

Firmware bin is invalid

enumerator HAILO_INVALID_CONTEXT_COUNT

Host build too many contexts

enumerator HAILO_INVALID_FRAME

Part or all of the result data is invalid

enumerator HAILO_INVALID_HEF

Invalid HEF

enumerator HAILO_PCIE_NOT_SUPPORTED_ON_PLATFORM

PCIe not supported on platform

enumerator HAILO_INTERRUPTED_BY_SIGNAL

Blocking syscall was interrupted by a signal

enumerator HAILO_START_VDMA_CHANNEL_FAIL

Starting VDMA channel failure

enumerator HAILO_SYNC_VDMA_BUFFER_FAIL

Synchronizing VDMA buffer failure

enumerator HAILO_STOP_VDMA_CHANNEL_FAIL

Stopping VDMA channel failure

enumerator HAILO_CLOSE_VDMA_CHANNEL_FAIL

Closing VDMA channel failure

enumerator HAILO_ATR_TABLES_CONF_VALIDATION_FAIL

Validating address translation tables failure, for FW control use

enumerator HAILO_EVENT_CREATE_FAIL

Creating event failure

enumerator HAILO_READ_EVENT_FAIL

Reading event failure

enumerator HAILO_DRIVER_OPERATION_FAILED

Driver operation (i.e ioctl) returned failure. Read driver log for more info (dmesg for linux)

enumerator HAILO_INVALID_FIRMWARE_MAGIC

Invalid FW magic

enumerator HAILO_INVALID_FIRMWARE_CODE_SIZE

Invalid FW code size

enumerator HAILO_INVALID_KEY_CERTIFICATE_SIZE

Invalid key certificate size

enumerator HAILO_INVALID_CONTENT_CERTIFICATE_SIZE

Invalid content certificate size

enumerator HAILO_MISMATCHING_FIRMWARE_BUFFER_SIZES

FW buffer sizes mismatch

enumerator HAILO_INVALID_FIRMWARE_CPU_ID

Invalid CPU ID in FW

enumerator HAILO_CONTROL_RESPONSE_MD5_MISMATCH

MD5 of control response does not match expected MD5

enumerator HAILO_GET_CONTROL_RESPONSE_FAIL

Get control response failed

enumerator HAILO_GET_D2H_EVENT_MESSAGE_FAIL

Reading device-to-host message failure

enumerator HAILO_MUTEX_INIT_FAIL

Mutex initialization failure

enumerator HAILO_OUT_OF_DESCRIPTORS

Cannot allocate more descriptors

enumerator HAILO_UNSUPPORTED_OPCODE

Unsupported opcode was sent to device

enumerator HAILO_USER_MODE_RATE_LIMITER_NOT_SUPPORTED

User mode rate limiter not supported on platform

enumerator HAILO_RATE_LIMIT_MAXIMUM_BANDWIDTH_EXCEEDED

Rate limit exceeded HAILO_DEFAULT_MAX_ETHERNET_BANDWIDTH_BYTES_PER_SEC

enumerator HAILO_ANSI_TO_UTF16_CONVERSION_FAILED

Failed converting ANSI string to UNICODE

enumerator HAILO_UTF16_TO_ANSI_CONVERSION_FAILED

Failed converting UNICODE string to ANSI

enumerator HAILO_UNEXPECTED_INTERFACE_INFO_FAILURE

Failed retrieving interface info

enumerator HAILO_UNEXPECTED_ARP_TABLE_FAILURE

Failed retrieving arp table

enumerator HAILO_MAC_ADDRESS_NOT_FOUND

MAC address not found in the arp table

enumerator HAILO_NO_IPV4_INTERFACES_FOUND

No interfaces found with an IPv4 address

enumerator HAILO_SHUTDOWN_EVENT_SIGNALLED

A shutdown event has been signaled

enumerator HAILO_THREAD_ALREADY_ACTIVATED

The given thread has already been activated

enumerator HAILO_THREAD_NOT_ACTIVATED

The given thread has not been activated

enumerator HAILO_THREAD_NOT_JOINABLE

The given thread is not joinable

enumerator HAILO_NOT_FOUND

Could not find element

enumerator HAILO_COMMUNICATION_CLOSED

The communication between endpoints is closed

enumerator HAILO_STREAM_ABORT

Stream recv/send was aborted

enumerator HAILO_DRIVER_NOT_INSTALLED

Driver is not installed/running on the system.

enumerator HAILO_NOT_AVAILABLE

Component is not available

enumerator HAILO_TRAFFIC_CONTROL_FAILURE

Traffic control failure

enumerator HAILO_INVALID_SECOND_STAGE

Second stage bin is invalid

enumerator HAILO_INVALID_PIPELINE

Pipeline is invalid

enumerator HAILO_NETWORK_GROUP_NOT_ACTIVATED

Network group is not activated

enumerator HAILO_VSTREAM_PIPELINE_NOT_ACTIVATED

VStream pipeline is not activated

enumerator HAILO_OUT_OF_FW_MEMORY

Cannot allocate more memory at fw

enumerator HAILO_STREAM_NOT_ACTIVATED

Stream is not activated

enumerator HAILO_DEVICE_IN_USE

The device is already in use

enumerator HAILO_OUT_OF_PHYSICAL_DEVICES

There are not enough physical devices

enumerator HAILO_INVALID_DEVICE_ARCHITECTURE

Invalid device architecture

enumerator HAILO_INVALID_DRIVER_VERSION

Invalid driver version

enumerator HAILO_RPC_FAILED

RPC failed

enumerator HAILO_INVALID_SERVICE_VERSION

Invalid service version

enumerator HAILO_NOT_SUPPORTED

Not supported operation

enumerator HAILO_NMS_BURST_INVALID_DATA

Invalid data in NMS burst

enumerator HAILO_OUT_OF_HOST_CMA_MEMORY

Cannot allocate more CMA memory at host

enumerator HAILO_QUEUE_IS_FULL

Cannot push more items into the queue

enumerator HAILO_DMA_MAPPING_ALREADY_EXISTS

DMA mapping already exists

enumerator HAILO_CANT_MEET_BUFFER_REQUIREMENTS

can't meet buffer requirements

enumerator HAILO_DRIVER_INVALID_RESPONSE

Driver returned invalid response. Make sure the driver version is the same as libhailort

enumerator HAILO_DRIVER_INVALID_IOCTL

Driver cannot handle ioctl. Can happen on libhailort vs driver version mismatch or when ioctl function is not supported

enumerator HAILO_DRIVER_TIMEOUT

Driver operation returned a timeout. Device reset may be required.

enumerator HAILO_DRIVER_INTERRUPTED

Driver operation interrupted by system request (i.e can happen on application exit)

enumerator HAILO_CONNECTION_REFUSED

Connection was refused by other side

enumerator HAILO_DRIVER_WAIT_CANCELED

Driver operation was canceled

enumerator HAILO_HEF_FILE_CORRUPTED

HEF file is corrupted

enumerator HAILO_HEF_NOT_SUPPORTED

HEF file is not supported. Make sure the DFC version is compatible.

enumerator HAILO_HEF_NOT_COMPATIBLE_WITH_DEVICE

HEF file is not compatible with device.

enumerator HAILO_INVALID_HEF_USE

Invalid HEF use (i.e. when using HEF from a file path without first copying its content to a mapped buffer while shared_weights is enabled)

enumerator HAILO_OPERATION_ABORTED

Operation was aborted

enumerator HAILO_DEVICE_NOT_CONNECTED

Device is not connected

enumerator HAILO_DEVICE_TEMPORARILY_UNAVAILABLE

Device is temporarily unavailable, try again later

enumerator HAILO_STATUS_COUNT

Must be last!

enumerator HAILO_STATUS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_dvm_options_e

Enum that represents the type of devices that would be measured

Values:

enumerator HAILO_DVM_OPTIONS_VDD_CORE

VDD_CORE DVM

enumerator HAILO_DVM_OPTIONS_VDD_IO

VDD_IO DVM

enumerator HAILO_DVM_OPTIONS_MIPI_AVDD

MIPI_AVDD DVM

enumerator HAILO_DVM_OPTIONS_MIPI_AVDD_H

MIPI_AVDD_H DVM

enumerator HAILO_DVM_OPTIONS_USB_AVDD_IO

USB_AVDD_IO DVM

enumerator HAILO_DVM_OPTIONS_VDD_TOP
VDD_TOP DVM

enumerator HAILO_DVM_OPTIONS_USB_AVDD_IO_HV
USB_AVDD_IO_HV DVM

enumerator HAILO_DVM_OPTIONS_AVDD_H
AVDD_H DVM

enumerator HAILO_DVM_OPTIONS_SDIO_VDD_IO
SDIO_VDD_IO DVM

enumerator HAILO_DVM_OPTIONS_OVERCURRENT_PROTECTION
OVERCURRENT_PROTECTION DVM

enumerator HAILO_DVM_OPTIONS_COUNT
Must be last!

enumerator HAILO_DVM_OPTIONS_AUTO
Select the default DVM option according to the supported features

enumerator HAILO_DVM_OPTIONS_MAX_ENUM
Max enum value to maintain ABI Integrity

enum `hailo_power_measurement_types_e`
Enum that represents what would be measured on the selected device

Values:

enumerator HAILO_POWER_MEASUREMENT_TYPES_SHUNT_VOLTAGE
SHUNT_VOLTAGE measurement type, measured in mV

enumerator HAILO_POWER_MEASUREMENT_TYPES_BUS_VOLTAGE
BUS_VOLTAGE measurement type, measured in mV

enumerator HAILO_POWER_MEASUREMENT_TYPES_POWER
POWER measurement type, measured in W

enumerator HAILO_POWER_MEASUREMENT_TYPES_CURRENT
CURRENT measurement type, measured in mA

enumerator HAILO_POWER_MEASUREMENT_TYPES_COUNT
Must be last!

enumerator HAILO_POWER_MEASUREMENT_TYPES_AUTO
Select the default measurement type according to the supported features

enumerator HAILO_POWER_MEASUREMENT_TYPES_MAX_ENUM
Max enum value to maintain ABI Integrity

enum hailo_sampling_period_e

Enum that represents all the bit options and related conversion times for each bit setting for Bus Voltage and Shunt Voltage

Values:

enumerator HAILO_SAMPLING_PERIOD_140US

enumerator HAILO_SAMPLING_PERIOD_204US

enumerator HAILO_SAMPLING_PERIOD_332US

enumerator HAILO_SAMPLING_PERIOD_588US

enumerator HAILO_SAMPLING_PERIOD_1100US

enumerator HAILO_SAMPLING_PERIOD_2116US

enumerator HAILO_SAMPLING_PERIOD_4156US

enumerator HAILO_SAMPLING_PERIOD_8244US

enumerator HAILO_SAMPLING_PERIOD_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_averaging_factor_e

Enum that represents all the AVG bit settings and related number of averages for each bit setting

Values:

enumerator HAILO_AVERAGE_FACTOR_1

enumerator HAILO_AVERAGE_FACTOR_4

enumerator HAILO_AVERAGE_FACTOR_16

enumerator HAILO_AVERAGE_FACTOR_64

enumerator HAILO_AVERAGE_FACTOR_128

enumerator HAILO_AVERAGE_FACTOR_256

enumerator HAILO_AVERAGE_FACTOR_512

enumerator HAILO_AVERAGE_FACTOR_1024

enumerator HAILO_AVERAGE_FACTOR_MAX_ENUM

Max enum value to maintain ABI Integrity

```
enum hailo_measurement_buffer_index_e
```

Enum that represents buffers on the device for power measurements storing

Values:

```
enumerator HAILO_MEASUREMENT_BUFFER_INDEX_0
```

```
enumerator HAILO_MEASUREMENT_BUFFER_INDEX_1
```

```
enumerator HAILO_MEASUREMENT_BUFFER_INDEX_2
```

```
enumerator HAILO_MEASUREMENT_BUFFER_INDEX_3
```

```
enumerator HAILO_MEASUREMENT_BUFFER_INDEX_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_device_type_t
```

Hailo device type

Values:

```
enumerator HAILO_DEVICE_TYPE_PCIE
```

```
enumerator HAILO_DEVICE_TYPE_ETH
```

```
enumerator HAILO_DEVICE_TYPE_INTEGRATED
```

```
enumerator HAILO_DEVICE_TYPE_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_scheduling_algorithm_e
```

Scheduler algorithm

Values:

```
enumerator HAILO_SCHEDULING_ALGORITHM_NONE
```

Scheduling disabled

```
enumerator HAILO_SCHEDULING_ALGORITHM_ROUND_ROBIN
```

Round Robin

```
enumerator HAILO_SCHEDULING_ALGORITHM_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_device_architecture_e
```

Device architecture

Values:

```
enumerator HAILO_ARCH_HAILO8_A0
```

```
enumerator HAILO_ARCH_HAILO8  
enumerator HAILO_ARCH_HAILO8L  
enumerator HAILO_ARCH_HAILO15H  
enumerator HAILO_ARCH_HAILO15L  
enumerator HAILO_ARCH_HAILO15M  
enumerator HAILO_ARCH_HAILO10H  
enumerator HAILO_ARCH_HAILO12L  
enumerator HAILO_ARCH_MAX_ENUM  
    Max enum value to maintain ABI Integrity
```

```
enum hailo_cpu_id_t
```

Values:

```
enumerator HAILO_CPU_ID_0  
enumerator HAILO_CPU_ID_1  
enumerator HAILO_CPU_ID_MAX_ENUM  
    Max enum value to maintain ABI Integrity
```

```
enum hailo_device_boot_source_t
```

Values:

```
enumerator HAILO_DEVICE_BOOT_SOURCE_INVALID  
enumerator HAILO_DEVICE_BOOT_SOURCE_PCIE  
enumerator HAILO_DEVICE_BOOT_SOURCE_FLASH  
enumerator HAILO_DEVICE_BOOT_SOURCE_MAX  
    Max enum value to maintain ABI Integrity
```

```
enum hailo_endianness_t
```

Endianness (byte order)

Values:

```
enumerator HAILO_BIG_ENDIAN  
enumerator HAILO_LITTLE_ENDIAN
```

enumerator HAILO_ENDIANNESS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum `hailo_format_type_t`

Data format types

Values:

enumerator HAILO_FORMAT_TYPE_AUTO

Chosen automatically to match the format expected by the device, usually `UINT8`. Can be checked using `hailo_stream_info_t` `format.type`.

enumerator HAILO_FORMAT_TYPE_UINT8

Data format type `uint8_t` - 1 byte per item, host/device side

enumerator HAILO_FORMAT_TYPE_UINT16

Data format type `uint16_t` - 2 bytes per item, host/device side

enumerator HAILO_FORMAT_TYPE_FLOAT32

Data format type `float32_t` - used only on host side (Translated in the quantization process)

enumerator HAILO_FORMAT_TYPE_MAX_ENUM

Max enum value to maintain ABI Integrity

enum `hailo_format_order_t`

Data format orders, i.e. how the rows, columns and features are ordered:

- N: Number of images in the batch
- H: Height of the image
- W: Width of the image
- C: Number of channels of the image (e.g. 3 for RGB, 1 for grayscale...)

Values:

enumerator HAILO_FORMAT_ORDER_AUTO

Chosen automatically to match the format expected by the device.

enumerator HAILO_FORMAT_ORDER_NHWC

- Host side: [N, H, W, C]
- Device side: [N, H, W, C], where width is padded to 8 bytes

enumerator HAILO_FORMAT_ORDER_NHCW

- Not used for host side
- Device side: [N, H, C, W], where width is padded to 8 bytes

enumerator HAILO_FORMAT_ORDER_FCR

FCR means first channels (features) are sent to HW:

- Host side: [N, H, W, C]

- Device side: [N, H, W, C]:
 - Input - channels are expected to be aligned to 8 bytes
 - Output - width is padded to 8 bytes

enumerator **HAILO_FORMAT_ORDER_F8CR**

F8CR means first 8-channels X width are sent to HW:

- Host side: [N, H, W, C]
- Device side: [N, H, W, 8C], where channels are padded to 8 elements:
- ROW1:
 - W X 8C_1, W X 8C_2, ... , W X 8C_n
- ROW2:
 - W X 8C_1, W X 8C_2, ... , W X 8C_n ...

enumerator **HAILO_FORMAT_ORDER_NHW**

Output format of argmax layer:

- Host side: [N, H, W, 1]
- Device side: [N, H, W, 1], where width is padded to 8 bytes

enumerator **HAILO_FORMAT_ORDER_NC**

Channels only:

- Host side: [N,C]
- Device side: [N, C], where channels are padded to 8 bytes

enumerator **HAILO_FORMAT_ORDER_BAYER_RGB**

Bayer format:

- Host side: [N, H, W, 1]
- Device side: [N, H, W, 1], where width is padded to 8 bytes

enumerator **HAILO_FORMAT_ORDER_12_BIT_BAYER_RGB**

Bayer format, same as [**HAILO_FORMAT_ORDER_BAYER_RGB**](#) where Channel is 12-bit

enumerator **HAILO_FORMAT_ORDER_HAILO_NMS**

Deprecated. Should use **HAILO_FORMAT_ORDER_HAILO_NMS_BY_CLASS**, **HAILO_FORMAT_ORDER_HAILO_NMS_BY_SCORE** (user formats) or **HAILO_FORMAT_ORDER_HAILO_NMS_ON_CHIP** (device format) instead.

enumerator **HAILO_FORMAT_ORDER_RGB888**

- Not used for host side
- Device side: [N, H, W, C], where channels are 4 (RGB + 1 padded zero byte) and width is padded to 8 elements

enumerator **HAILO_FORMAT_ORDER_NCHW**

- Host side: [N, C, H, W]
- Not used for device side

enumerator HAILO_FORMAT_ORDER_YUY2

YUV format, encoding 2 pixels in 32 bits [Y0, U0, Y1, V0] represents [Y0, U0, V0], [Y1, U0, V0]

- Host side: [Y0, U0, Y1, V0]
- Device side: [Y0, U0, Y1, V0]

enumerator HAILO_FORMAT_ORDER_NV12

YUV format, encoding 8 pixels in 96 bits [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, U0, V0, U1, V1] represents [Y0, U0, V0], [Y1, U0, V0], [Y2, U0, V0], [Y3, U0, V0], [Y4, U1, V1], [Y5, U1, V1], [Y6, U1, V1], [Y7, U1, V1]

- Not used for device side

enumerator HAILO_FORMAT_ORDER_NV21

YUV format, encoding 8 pixels in 96 bits [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, V0, U0, V1, U1] represents [Y0, V0, U0], [Y1, V0, U0], [Y2, V0, U0], [Y3, V0, U0], [Y4, V1, U1], [Y5, V1, U1], [Y6, V1, U1], [Y7, V1, U1]

- Not used for device side

enumerator HAILO_FORMAT_ORDER_HAILO_YYUV

Internal implementation for HAILO_FORMAT_ORDER_NV12 format [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, U0, V0, U1, V1] is represented by [Y0, Y1, Y2, Y3, U0, V0, Y4, Y5, Y6, Y7, U1, V1]

- Not used for host side

enumerator HAILO_FORMAT_ORDER_HAILO_YYVU

Internal implementation for HAILO_FORMAT_ORDER_NV21 format [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, V0, U0, V1, U1] is represented by [Y0, Y1, Y2, Y3, V0, U0, Y4, Y5, Y6, Y7, V1, U1]

- Not used for host side

enumerator HAILO_FORMAT_ORDER_RGB4

RGB, where every row is padded to 4.

- Host side: [N, H, W, C], where width*channels are padded to 4.
- Not used for device side

enumerator HAILO_FORMAT_ORDER_I420

YUV format, encoding 8 pixels in 96 bits [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, U0, U1, V0, V1] represents [Y0, U0, V0], [Y1, U0, V0], [Y2, U0, V0], [Y3, U0, V0], [Y4, U1, V1], [Y5, U1, V1], [Y6, U1, V1], [Y7, U1, V1]

- Not used for device side

enumerator HAILO_FORMAT_ORDER_HAILO_YYYYUV

Internal implementation for HAILO_FORMAT_ORDER_I420 format [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, U0, U1, V0, V1] is represented by [Y0, Y1, Y2, Y3, U0, V0, Y4, Y5, Y6, Y7, U1, V1]

- Not used for host side

enumerator HAILO_FORMAT_ORDER_HAILO_NMS_WITH_BYTE_MASK

NMS_WITH_BYTE_MASK format

- Host side

```
struct (packed) {
    uint16_t detections_count;
    hailo_detection_with_byte_mask_t[detections_count];
};
```

The host format type supported [HAILO_FORMAT_TYPE_UINT8](#).

- Not used for device side

enumerator HAILO_FORMAT_ORDER_HAILO_NMS_ON_CHIP

NMS bbox

- Device side
- Result of NMS layer on chip (Internal implementation)
- Not used for host side

enumerator HAILO_FORMAT_ORDER_HAILO_NMS_BY_CLASS

NMS bbox

- Host side

For each class ([hailo_nms_shape_t.number_of_classes](#)), the layout is

```
struct (packed) {
    float32_t bbox_count;
    hailo_bbox_float32_t bbox[bbox_count];
};
```

Maximum amount of bboxes per [class](#) is `::hailo_nms_shape_t.max_bboxes_per_class`.

- Not used for device side

enumerator HAILO_FORMAT_ORDER_HAILO_NMS_BY_SCORE

NMS bbox

- Host side

For all classes the layout is

```
struct (packed) {
    uint16_t bbox_count;
    hailo_detection_t bbox[bbox_count];
};
```

Maximum amount of bboxes is `::hailo_nms_shape_t.max_bboxes_total`.
It [is](#) possible to use `::hailo_detections_t` to parse the data.

The host [format type](#) supported `::HAILO_FORMAT_TYPE_UINT8`.

- Not used for device side

enumerator HAILO_FORMAT_ORDER_MAX_ENUM

Max enum value to maintain ABI Integrity

enum `hailo_format_flags_t`

Data format flags

Values:

enumerator HAILO_FORMAT_FLAGS_NONE

enumerator HAILO_FORMAT_FLAGS_QUANTIZED

If not set, HailoRT performs the quantization (scaling) step. If set:

- Input data: HailoRT assumes that the data is already quantized (scaled) by the user, so it does not perform the quantization (scaling) step.
- Output data: The data will be returned to the user without rescaling (i.e., the data won't be rescaled by HailoRT).

Note: This flag is deprecated and its usage is ignored. Determine whether to quantize (or de-quantize) the data will be decided by the src-data and dst-data types.

enumerator HAILO_FORMAT_FLAGS_TRANSPOSED

If set, the frame height/width are transposed. Supported orders:

- `HAILO_FORMAT_ORDER_NHWC`
- `HAILO_FORMAT_ORDER_NHW`
- `HAILO_FORMAT_ORDER_BAYER_RGB`
- `HAILO_FORMAT_ORDER_12_BIT_BAYER_RGB`
- `HAILO_FORMAT_ORDER_FCR`
- `HAILO_FORMAT_ORDER_F8CR`

When set on host side, `hailo_stream_info_t` shape of the stream will be a transposed version of the host buffer (The height and width will be swapped)

enumerator HAILO_FORMAT_FLAGS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum `hailo_stream_transform_mode_t`

Indicates how transformations on the data should be done

Values:

enumerator HAILO_STREAM_NO_TRANSFORM

The vstream will not run the transformation (The data will be in hw format)

enumerator HAILO_STREAM_TRANSFORM_COPY

The transformation process will be part of the vstream send/recv (The data will be in host format).

enumerator HAILO_STREAM_MAX_ENUM

Max enum value to maintain ABI Integrity

```
enum hailo_stream_direction_t
    Stream direction - host to device or device to host
```

Values:

```
enumerator HAILO_H2D_STREAM
```

```
enumerator HAILO_D2H_STREAM
```

```
enumerator HAILO_STREAM_DIRECTION_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_stream_flags_t
```

Stream flags

Values:

```
enumerator HAILO_STREAM_FLAGS_NONE
```

No flags

```
enumerator HAILO_STREAM_FLAGS_ASYNC
```

Async stream

```
enumerator HAILO_STREAM_FLAGS_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_dma_buffer_direction_t
```

Hailo dma buffer direction

Values:

```
enumerator HAILO_DMA_BUFFER_DIRECTION_H2D
```

Buffers sent from the host (H) to the device (D). Used for input streams

```
enumerator HAILO_DMA_BUFFER_DIRECTION_D2H
```

Buffers received from the device (D) to the host (H). Used for output streams

```
enumerator HAILO_DMA_BUFFER_DIRECTION_BOTH
```

Buffers can be used both send to the device and received from the device

```
enumerator HAILO_DMA_BUFFER_DIRECTION_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_buffer_flags_t
```

Hailo buffer flags

Values:

```
enumerator HAILO_BUFFER_FLAGS_NONE
```

No flags - heap allocated buffer

enumerator HAILO_BUFFER_FLAGS_DMA

Buffer is mapped to DMA (will be page aligned implicitly)

enumerator HAILO_BUFFER_FLAGS_CONTINUOUS

Buffer is physically continuous (will be page aligned implicitly)

enumerator HAILO_BUFFER_FLAGS_SHARED_MEMORY

Buffer is shared memory (will be page aligned implicitly)

enumerator HAILO_BUFFER_FLAGS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_stream_interface_t

Values:

enumerator HAILO_STREAM_INTERFACE_PCIE

enumerator HAILO_STREAM_INTERFACE_ETH

enumerator HAILO_STREAM_INTERFACE_MIPI

enumerator HAILO_STREAM_INTERFACE_INTEGRATED

enumerator HAILO_STREAM_INTERFACE_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_vstream_stats_flags_t

Virtual stream statistics flags

Values:

enumerator HAILO_VSTREAM_STATS_NONE

No stats

enumerator HAILO_VSTREAM_STATS_MEASURE_FPS

Measure vstream FPS

enumerator HAILO_VSTREAM_STATS_MEASURE_LATENCY

Measure vstream latency

enumerator HAILO_VSTREAM_STATS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_pipeline_elem_stats_flags_t

Pipeline element statistics flags

Values:

enumerator HAILO_PIPELINE_ELEM_STATS_NONE

No stats

enumerator HAILO_PIPELINE_ELEM_STATS_MEASURE_FPS

Measure element FPS

enumerator HAILO_PIPELINE_ELEM_STATS_MEASURE_LATENCY

Measure element latency

enumerator HAILO_PIPELINE_ELEM_STATS_MEASURE_QUEUE_SIZE

Measure element queue size

enumerator HAILO_PIPELINE_ELEM_STATS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_pix_buffer_memory_type_t

Values:

enumerator HAILO_PIX_BUFFER_MEMORY_TYPE_USERPTR

enumerator HAILO_PIX_BUFFER_MEMORY_TYPE_DMABUF

enum hailo_nms_burst_type_t

Values:

enumerator HAILO_BURST_TYPE_H8_BBOX

enumerator HAILO_BURST_TYPE_H15_BBOX

enumerator HAILO_BURST_TYPE_H8_PER_CLASS

enumerator HAILO_BURST_TYPE_H15_PER_CLASS

enumerator HAILO_BURST_TYPE_H15_PER_FRAME

enumerator HAILO_BURST_TYPE_COUNT

enum hailo_power_mode_t

Power modes

Values:

enumerator HAILO_POWER_MODE_PERFORMANCE

enumerator HAILO_POWER_MODE_ULTRA_PERFORMANCE

enumerator HAILO_POWER_MODE_MAX_ENUM

Max enum value to maintain ABI Integrity

```
enum hailo_latency_measurement_flags_t
```

Latency measurement flags

Values:

```
enumerator HAILO_LATENCY_NONE
```

```
enumerator HAILO_LATENCY_MEASURE
```

```
enumerator HAILO_LATENCY_CLEAR_AFTER_GET
```

```
enumerator HAILO_LATENCY_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_notification_id_t
```

Notification IDs and structures section start Notification IDs, for each notification, one of the *hailo_notification_message_parameters_t* union will be set.

Values:

```
enumerator HAILO_NOTIFICATION_ID_ETHERNET_RX_ERROR
```

Matches *hailo_notification_message_parameters_t::rx_error_notification*.

```
enumerator HAILO_NOTIFICATION_ID_HEALTH_MONITOR_TEMPERATURE_ALARM
```

Matches *hailo_notification_message_parameters_t::health_monitor_temperature_alarm_notification*

```
enumerator HAILO_NOTIFICATION_ID_HEALTH_MONITOR_DATAFLOW_SHUTDOWN
```

Matches *hailo_notification_message_parameters_t::health_monitor_dataflow_shutdown_notification*

```
enumerator HAILO_NOTIFICATION_ID_HEALTH_MONITOR_OVERCURRENT_ALARM
```

Matches *hailo_notification_message_parameters_t::health_monitor_overcurrent_alert_notification*

```
enumerator HAILO_NOTIFICATION_ID_LCU_ECC_CORRECTABLE_ERROR
```

Matches *hailo_notification_message_parameters_t::health_monitor_lcu_ecc_error_notification*

```
enumerator HAILO_NOTIFICATION_ID_LCU_ECC_UNCORRECTABLE_ERROR
```

Matches *hailo_notification_message_parameters_t::health_monitor_lcu_ecc_error_notification*

```
enumerator HAILO_NOTIFICATION_ID_CPU_ECC_ERROR
```

Matches *hailo_notification_message_parameters_t::health_monitor_cpu_ecc_notification*

```
enumerator HAILO_NOTIFICATION_ID_CPU_ECC_FATAL
```

Matches *hailo_notification_message_parameters_t::health_monitor_cpu_ecc_notification*

```
enumerator HAILO_NOTIFICATION_ID_DEBUG
```

Matches *hailo_notification_message_parameters_t::debug_notification*

```
enumerator HAILO_NOTIFICATION_ID_CONTEXT_SWITCH_BREAKPOINT_REACHED
```

Matches *hailo_notification_message_parameters_t::context_switch_breakpoint_reached_notification*

enumerator HAILO_NOTIFICATION_ID_HEALTH_MONITOR_CLOCK_CHANGED_EVENT

Matches *hailo_notification_message_parameters_t::health_monitor_clock_changed_notification*

enumerator HAILO_NOTIFICATION_ID_HW_INFER_MANAGER_INFERENCE_DONE

Matches *hailo_notification_message_parameters_t::hailo_hw_infer_manager_inference_done_notification*

enumerator HAILO_NOTIFICATION_ID_CONTEXT_SWITCH_RUN_TIME_ERROR_EVENT

Matches *hailo_notification_message_parameters_t::context_switch_run_time_error*

enumerator HAILO_NOTIFICATION_ID_NN_CORE_CRC_ERROR_EVENT

Empty message: no parameters.

enumerator HAILO_NOTIFICATION_ID_THROTTLING_STATE_CHANGE_EVENT

Matches *hailo_notification_message_parameters_t::throttling_state_change*

enumerator HAILO_NOTIFICATION_ID_COUNT

Must be last!

enumerator HAILO_NOTIFICATION_ID_MAX_ENUM

Max enum value to maintain ABI Integrity

enum *hailo_temperature_protection_temperature_zone_t*

Values:

enumerator HAILO_TEMPERATURE_PROTECTION_TEMPERATURE_ZONE__GREEN

enumerator HAILO_TEMPERATURE_PROTECTION_TEMPERATURE_ZONE__ORANGE

enumerator HAILO_TEMPERATURE_PROTECTION_TEMPERATURE_ZONE__RED

enum *hailo_overcurrent_protection_overcurrent_zone_t*

Values:

enumerator HAILO_OVERCURRENT_PROTECTION_OVERCURRENT_ZONE__GREEN

enumerator HAILO_OVERCURRENT_PROTECTION_OVERCURRENT_ZONE__RED

enum *hailo_hw_throttling_state_e*

NOTE: Enum values match *mercury_throttling_state* enum defined by Hailo SCU.

Values:

enumerator HAILO_THROTTLING_STATE_NONE

enumerator HAILO_THROTTLING_STATE_0_LIGHT

enumerator HAILO_THROTTLING_STATE_1_MEDIUM

```
enumerator HAILO_THROTTLING_STATE_2_HEAVY

enumerator HAILO_THROTTLING_STATE_3_SEVERE

enumerator HAILO_THROTTLING_STATE_4_STREAMS_OFF

enumerator HAILO_THROTTLING_STATE_OVERHEAT

enumerator HAILO_THROTTLING_STATE_COUNT

enum hailo_reset_device_mode_t
    Hailo device reset modes

    Values:

    enumerator HAILO_RESET_DEVICE_MODE_CHIP

    enumerator HAILO_RESET_DEVICE_MODE_NN_CORE

    enumerator HAILO_RESET_DEVICE_MODE_SOFT

    enumerator HAILO_RESET_DEVICE_MODE_FORCED_SOFT

    enumerator HAILO_RESET_DEVICE_MODE_REBOOT

    enumerator HAILO_RESET_DEVICE_MODE_MAX_ENUM

enum hailo_watchdog_mode_t
    Values:

    enumerator HAILO_WATCHDOG_MODE_HW_SW

    enumerator HAILO_WATCHDOG_MODE_HW_ONLY

    enumerator HAILO_WATCHDOG_MODE_MAX_ENUM

enum hailo_sensor_types_t
    Values:

    enumerator HAILO_SENSOR_TYPES_GENERIC

    enumerator HAILO_SENSOR_TYPES_ONSEMI_AR0220AT

    enumerator HAILO_SENSOR_TYPES_RASPICAM

    enumerator HAILO_SENSOR_TYPES_ONSEMI_AS0149AT
```

```
enumerator HAILO_SENSOR_TYPES_HAILO8_ISP
```

```
enumerator HAILO_SENSOR_TYPES_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_fw_logger_interface_t
```

Values:

```
enumerator HAILO_FW_LOGGER_INTERFACE_PCIE
```

```
enumerator HAILO_FW_LOGGER_INTERFACE_UART
```

```
enumerator HAILO_FW_LOGGER_INTERFACE_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_fw_logger_level_t
```

Values:

```
enumerator HAILO_FW_LOGGER_LEVEL_TRACE
```

```
enumerator HAILO_FW_LOGGER_LEVEL_DEBUG
```

```
enumerator HAILO_FW_LOGGER_LEVEL_INFO
```

```
enumerator HAILO_FW_LOGGER_LEVEL_WARN
```

```
enumerator HAILO_FW_LOGGER_LEVEL_ERROR
```

```
enumerator HAILO_FW_LOGGER_LEVEL_FATAL
```

```
enumerator HAILO_FW_LOGGER_LEVEL_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_log_type_t
```

Values:

```
enumerator HAILO_LOG_TYPE__RUNTIME
```

```
enumerator HAILO_LOG_TYPE__SYSTEM_CONTROL
```

```
enumerator HAILO_LOG_TYPE__NNC
```

```
enumerator HAILO_LOG_TYPE_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
typedef float float32_t
```

typedef double float64_t

typedef uint16_t nms_bbox_counter_t

typedef struct _hailo_device *hailo_device

Represents the device (chip)

typedef struct _hailo_vdevice *hailo_vdevice

Represents a virtual device which manages several physical devices

typedef struct _hailo_hef *hailo_hef

Compiled HEF model that can be loaded to Hailo devices

typedef struct _hailo_input_stream *hailo_input_stream

Input (host to device) stream representation

typedef struct _hailo_output_stream *hailo_output_stream

Output (device to host) stream representation

typedef struct _hailo_configured_network_group *hailo_configured_network_group

Loaded network_group that can be activated

typedef struct _hailo_activated_network_group *hailo_activated_network_group

Activated network_group that can be used to send/receive data

typedef struct _hailo_input_transform_context *hailo_input_transform_context

Object used for input stream transformation, store all necessary allocated buffers

typedef struct _hailo_output_transform_context *hailo_output_transform_context

Object used for output stream transformation, store all necessary allocated buffers

typedef struct _hailo_output_demuxer *hailo_output_demuxer

Object used to demux muxed stream

typedef struct _hailo_input_vstream *hailo_input_vstream

Input virtual stream

typedef struct _hailo_output_vstream *hailo_output_vstream

Output virtual stream

typedef enum *hailo_dvm_options_e* hailo_dvm_options_t

Enum that represents the type of devices that would be measured

typedef enum *hailo_power_measurement_types_e* hailo_power_measurement_types_t

Enum that represents what would be measured on the selected device

typedef enum *hailo_sampling_period_e* hailo_sampling_period_t

Enum that represents all the bit options and related conversion times for each bit setting for Bus Voltage and Shunt Voltage

`typedef enum hailo_averaging_factor_e hailo_averaging_factor_t`

Enum that represents all the AVG bit settings and related number of averages for each bit setting

`typedef enum hailo_measurement_buffer_index_e hailo_measurement_buffer_index_t`

Enum that represents buffers on the device for power measurements storing

`typedef struct _hailo_scan_devices_params_t hailo_scan_devices_params_t`

Additional scan params, for future compatibility

`typedef enum hailo_scheduling_algorithm_e hailo_scheduling_algorithm_t`

Scheduler algorithm

`typedef enum hailo_device_architecture_e hailo_device_architecture_t`

Device architecture

`typedef void (*hailo_stream_write_async_callback_t)(const
 hailo_stream_write_async_completion_info_t *info)`

Async stream write complete callback prototype.

`typedef void (*hailo_stream_read_async_callback_t)(const
 hailo_stream_read_async_completion_info_t *info)`

Async stream read complete callback prototype.

`typedef enum hailo_hw_throttling_state_e hailo_hw_throttling_state_t`

NOTE: Enum values match `mercury_throttling_state` enum defined by Hailo SCU.

`typedef void (*hailo_notification_callback)(hailo_device device, const hailo_notification_t *notification,
void *opaque)`

Notification IDs and structures section end A notification callback. See [`hailo_set_notification_callback`](#)

Warning: Throwing exceptions in the callback is not supported!

Param device [in] The `hailo_device` that got the notification.

Param notification [in] The notification data.

Param opaque [in] User specific data.

`hailo_status hailo_get_library_version(hailo_version_t *version)`

Retrieves hailort library version.

Parameters `version - [out]` Will be filled with hailort library version.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`const char *hailo_get_status_message(hailo_status status)`

Returns a string format of @ status.

Parameters `status - [in]` A `hailo_status` to be converted to string format.

Returns Upon success, returns @ status as a string format. Otherwise, returns `nullptr`.

`HAILO_MAX_ENUM`

HAILO_INFINITE

HAILO_DEFAULT_ETH_SCAN_TIMEOUT_MS

HAILO_DEFAULT_ETH_DEVICE_PORT

HAILO_DEFAULT_ETH_MAX_PAYLOAD_SIZE

HAILO_DEFAULT_ETH_MAX_NUMBER_OF_RETRIES

HAILO_ETH_ADDRESS_ANY

HAILO_ETH_PORT_ANY

HAILO_MAX_NAME_SIZE

HAILO_MAX_STREAM_NAME_SIZE

HAILO_MAX_BOARD_NAME_LENGTH

HAILO_MAX_DEVICE_ID_LENGTH

HAILO_MAX_SERIAL_NUMBER_LENGTH

HAILO_MAX_PART_NUMBER_LENGTH

HAILO_MAX_PRODUCT_NAME_LENGTH

HAILO_DEFAULT_INIT_SAMPLING_PERIOD_US

HAILO_DEFAULT_INIT_AVERAGING_FACTOR

HAILO_DEFAULT_BUFFERS_THRESHOLD

HAILO_DEFAULT_MAX_ETHERNET_BANDWIDTH_BYTES_PER_SEC

HAILO_MAX_STREAMS_COUNT

HAILO_DEFAULT_BATCH_SIZE

HAILO_MAX_NETWORK_GROUPS

HAILO_MAX_NETWORK_GROUP_NAME_SIZE

HAILO_MAX_NETWORK_NAME_SIZE

HAILO_MAX_NETWORKS_IN_NETWORK_GROUP

HAILO_PCIE_ANY_DOMAIN

HAILO_DEFAULT_VSTREAM_QUEUE_SIZE

HAILO_DEFAULT_VSTREAM_TIMEOUT_MS

HAILO_DEFAULT_ASYNC_INFERENCE_TIMEOUT_MS

HAILO_DEFAULT_ASYNC_INFERENCE_QUEUE_SIZE

HAILO_DEFAULT_DEVICE_COUNT

HAILO_SOC_ID_LENGTH

HAILO_ETH_MAC_LENGTH

HAILO_UNIT_LEVEL_TRACKING_BYTES_LENGTH

HAILO_SOC_PM_VALUES_BYTES_LENGTH

HAILO_GPIO_MASK_VALUES_LENGTH

HAILO_MAX_TEMPERATURE_THROTTLING_LEVELS_NUMBER

HAILO_UNIQUE_VDEVICE_GROUP_ID

HAILO_DEFAULT_VDEVICE_GROUP_ID

HAILO_SCHEDULER_PRIORITY_NORMAL

HAILO_SCHEDULER_PRIORITY_MAX

HAILO_SCHEDULER_PRIORITY_MIN

MAX_NUMBER_OF_PLANES

NUMBER_OF_PLANES_NV12_NV21

NUMBER_OF_PLANES_I420

INVALID_QUANT_INFO

HAILO_RANDOM_SEED

`HAILO_STATUS_VARIABLES`

HailoRT return codes

`HAILO_STREAM_ABORTED_BY_USER`

`HAILO_DRIVER_FAIL`

`HAILO_PCIE_DRIVER_NOT_INSTALLED`

`HAILO_DEFAULT_TRANSFORM_PARAMS`

`HAILO_PCIE_STREAM_PARAMS_DEFAULT`

`HAILO_ACTIVATE_NETWORK_GROUP_PARAMS_DEFAULT`

`struct hailo_version_t`

`#include <hailort.h>` HailoRT library version

Public Members

`uint32_t major`

`uint32_t minor`

`uint32_t revision`

`struct hailo_power_measurement_data_t`

`#include <hailort.h>` Data of the power measurement samples

Public Members

`float32_t average_value`

`float32_t average_time_value_milliseconds`

`float32_t min_value`

`float32_t max_value`

`uint32_t total_number_of_samples`

`struct hailo_PCIE_device_info_t`

`#include <hailort.h>` PCIe device information

Public Members`uint32_t domain``uint32_t bus``uint32_t device``uint32_t func``struct hailo_device_id_t`

`#include <hailort.h>` Hailo device ID string - BDF for PCIe devices, IP address for Ethernet devices.

Public Members`char id[(32)]``struct hailo_vdevice_params_t`

`#include <hailort.h>` Virtual device parameters

Public Members`uint32_t device_count`

Requested number of physical devices. If `device_ids` is not NULL represents the number of `hailo_device_id_t` in `device_ids`.

`hailo_device_id_t *device_ids`

Specific device ids to create the vdevice from. If NULL, the vdevice will try to occupy devices from the available pool.

`hailo_scheduling_algorithm_t scheduling_algorithm`

The scheduling algorithm to use for network group scheduling

`const char *group_id`

Key for using a shared VDevice. To create a unique VDevice, use HAILO_UNIQUE_VDEVICE_GROUP_ID

`bool multi_process_service`

Flag specifies whether to create the VDevice in HailoRT service or not. Defaults to false

`struct hailo_firmware_version_t`

`#include <hailort.h>` Hailo firmware version

Public Members`uint32_t major``uint32_t minor``uint32_t revision``struct hailo_device_identity_t`

`#include <hailort.h>` Hailo device identity

Public Members

```
uint32_t protocol_version  
  
hailo_firmware_version_t fw_version  
  
uint32_t logger_version  
  
uint8_t board_name_length  
  
char board_name[(32)]  
  
bool is_release  
  
bool extended_context_switch_buffer  
  
bool extended_fw_check  
  
hailo_device_architecture_t device_architecture  
  
uint8_t serial_number_length  
  
char serial_number[(16)]  
  
uint8_t part_number_length  
  
char part_number[(16)]  
  
uint8_t product_name_length  
  
char product_name[(42)]
```

```
struct hailo_core_information_t
```

Public Members

```
bool is_release  
  
bool extended_context_switch_buffer  
  
bool extended_fw_check  
  
hailo_firmware_version_t fw_version
```

```
struct hailo_device_supported_features_t  
#include <hailort.h> Hailo device supported features
```

Public Members

```
bool ethernet
    Is ethernet supported

bool mipi
    Is mipi supported

bool pcie
    Is pcie supported

bool current_monitoring
    Is current monitoring supported

bool mdio
    Is current mdio supported

bool power_measurement
    Is power measurement supported

struct hailo_extended_device_information_t
    #include <hailort.h> Hailo extended device information
```

Public Members

```
uint32_t neural_network_core_clock_rate
    The core clock rate

hailo_device_supported_features_t supported_features
    Hailo device supported features

hailo_device_boot_source_t boot_source
    Device boot source

uint8_t soc_id[(32)]
    SOC id

uint8_t lcs
    Device lcs

uint8_t eth_mac_address[(6)]
    Device Ethernet Mac address

uint8_t unit_level_tracking_id[(12)]
    Hailo device unit level tracking id

uint8_t soc_pm_values[(24)]
    Hailo device pm values
```

```
uint16_t gpio_mask  
Hailo device GPIO mask values
```

```
struct hailo_i2c_slave_config_t  
#include <hailort.h> I2C slave configuration
```

Public Members

```
hailo_endianness_t endianness
```

```
uint16_t slave_address
```

```
uint8_t register_address_size
```

```
uint8_t bus_index
```

```
bool should_hold_bus
```

```
struct hailo_fw_user_config_information_t  
#include <hailort.h> Firmware user config information
```

Public Members

```
uint32_t version
```

```
uint32_t entry_count
```

```
uint32_t total_size
```

```
struct hailo_format_t  
#include <hailort.h> Hailo data format
```

Public Members

```
hailo_format_type_t type
```

```
hailo_format_order_t order
```

```
hailo_format_flags_t flags
```

```
struct hailo_buffer_parameters_t  
#include <hailort.h> Hailo buffer parameters
```

Public Members

hailo_buffer_flags_t flags

```
struct hailo_transform_params_t  
#include <hailort.h> Input or output data transform parameters
```

Public Members

hailo_stream_transform_mode_t transform_mode

hailo_format_t user_buffer_format

```
struct hailo_demux_params_t  
#include <hailort.h> Demuxer params
```

Public Members

uint8_t reserved

```
struct hailo_quant_info_t
```

#include <hailort.h> Quantization information. Property of *hailo_stream_info_t*, *hailo_vstream_info_t*. Hailo devices require input data to be quantized/scaled before it is sent. Similarly, data outputted from the device needs to be ‘de-quantized’/rescaled as well. Each input/output layer is assigned two floating point values that are parameters to an input/output transformation: qp_zp (zero_point) and qp_scale. These values are stored in the HEF.

- Input transformation: Input data is divided by qp_scale and then qp_zp is added to the result.
- Output transformation: qp_zp is subtracted from output data and then the result is multiplied by qp_scale.

If the output’s quant_info is INVALID_QUANT_INFO, it means there are multiple quant_infos. In that case, use *hailo_get_output_stream_quant_infos* or *hailo_get_output_vstream_quant_infos* to get the quant info list.

Public Members

float32_t qp_zp

zero_point

float32_t qp_scale

scale

float32_t limvals_min

min limit value

float32_t limvals_max

max limit value

```
struct hailo_pcnie_input_stream_params_t
```

#include <hailort.h> PCIe input stream (host to device) parameters

Public Members

```
uint8_t reserved
```

```
struct hailo_PCIE_output_stream_params_t  
#include <hailort.h> PCIe output stream (device to host) parameters
```

Public Members

```
uint8_t reserved
```

```
struct hailo_integrated_input_stream_params_t  
#include <hailort.h> Core input stream (host to device) parameters
```

Public Members

```
uint8_t reserved
```

```
struct hailo_integrated_output_stream_params_t  
#include <hailort.h> Core output stream (device to host) parameters
```

Public Members

```
uint8_t reserved
```

```
struct hailo_stream_parameters_t  
#include <hailort.h> Hailo stream parameters
```

Public Members

```
hailo_stream_interface_t stream_interface
```

```
hailo_stream_direction_t direction
```

```
hailo_stream_flags_t flags
```

```
hailo_PCIE_input_stream_params_t pcie_input_params
```

```
hailo_integrated_input_stream_params_t integrated_input_params
```

```
hailo_PCIE_output_stream_params_t pcie_output_params
```

```
hailo_integrated_output_stream_params_t integrated_output_params
```

```
union hailo_stream_parameters_t::[anonymous] [anonymous]
```

```
struct hailo_stream_parameters_by_name_t  
#include <hailort.h> Hailo stream parameters per stream_name
```

Public Members

```
char name[((128))]
```

```
hailo_stream_parameters_t stream_params
```

```
struct hailo_vstream_params_t
```

```
#include <hailort.h> Virtual stream params
```

Public Members

```
hailo_format_t user_buffer_format
```

```
uint32_t timeout_ms
```

```
uint32_t queue_size
```

```
hailo_vstream_stats_flags_t vstream_stats_flags
```

```
hailo_pipeline_elem_stats_flags_t pipeline_elements_stats_flags
```

```
struct hailo_input_vstream_params_by_name_t
```

```
#include <hailort.h> Input virtual stream parameters
```

Public Members

```
char name[((128))]
```

```
hailo_vstream_params_t params
```

```
struct hailo_output_vstream_params_by_name_t
```

```
#include <hailort.h> Output virtual stream parameters
```

Public Members

```
char name[((128))]
```

```
hailo_vstream_params_t params
```

```
struct hailo_output_vstream_name_by_group_t
```

```
#include <hailort.h> Output virtual stream name by group
```

Public Members

```
char name[((128))]
```

```
uint8_t pipeline_group_index
```

```
struct hailo_3d_image_shape_t
```

```
#include <hailort.h> Image shape
```

Public Members

```
uint32_t height
```

```
uint32_t width
```

```
uint32_t features
```

```
struct hailo_pix_buffer_plane_t  
#include <hailort.h> image buffer plane
```

Public Members

```
uint32_t bytes_used
```

```
actual data
```

```
uint32_t plane_size
```

```
void *user_ptr
```

```
int fd
```

```
union hailo_pix_buffer_plane_t::[anonymous] [anonymous]
```

```
struct hailo_pix_buffer_t  
#include <hailort.h> image buffer
```

Public Members

```
uint32_t index
```

```
hailo_pix_buffer_plane_t planes[(4)]
```

```
uint32_t number_of_planes
```

```
hailo_pix_buffer_memory_type_t memory_type
```

```
struct hailo_dma_buffer_t
```

```
#include <hailort.h> dma buffer - intended for use with Linux's dma-buf sub system
```

Public Members

```
int fd
```

```
size_t size
```

```
struct hailo_nms_defuse_info_t
```

Public Members

```
uint32_t class_group_index
```

```
char original_name[((128))]
```

```
struct hailo_nms_info_t
```

```
#include <hailort.h> NMS Internal HW Info
```

Public Members

```
uint32_t number_of_classes
```

```
Amount of NMS classes
```

```
uint32_t max_bboxes_per_class
```

```
Maximum amount of bboxes per nms class
```

```
uint32_t max_bboxes_total
```

```
Maximum amount of total bboxes
```

```
uint32_t bbox_size
```

```
Internal usage
```

```
uint32_t chunks_per_frame
```

```
Internal usage
```

```
bool is_defused
```

```
hailo_nms_defuse_info_t defuse_info
```

```
uint32_t burst_size
```

```
Size of NMS burst in bytes
```

```
hailo_nms_burst_type_t burst_type
```

```
NMS burst type
```

```
struct hailo_nms_fuse_input_t
```

```
#include <hailort.h> NMS Fuse Input
```

Public Members

```
void *buffer
```

```
size_t size
```

```
hailo_nms_info_t nms_info
```

```
struct hailo_nms_shape_t
```

```
#include <hailort.h> Shape of nms result
```

Public Members`uint32_t number_of_classes`

Amount of NMS classes

`uint32_t max_bboxes_per_class`

Maximum amount of bboxes per nms class

`uint32_t max_bboxes_total`

Maximum amount of total bboxes

`uint32_t max_accumulated_mask_size`

Maximum accumulated mask size for all of the detections in a frame. Used only with 'HAILO_FORMAT_ORDER_HAILO_NMS_WITH_BYTE_MASK' format order. The default value is (`input_image_size * 2`)

`struct hailo_bbox_t`**Public Members**`uint16_t y_min``uint16_t x_min``uint16_t y_max``uint16_t x_max``uint16_t score``struct hailo_bbox_float32_t`**Public Members**`float32_t y_min``float32_t x_min``float32_t y_max``float32_t x_max``float32_t score``struct hailo_rectangle_t`

Public Members*float32_t* y_min*float32_t* x_min*float32_t* y_max*float32_t* x_max

struct hailo_detection_t

Public Members*float32_t* y_min*float32_t* x_min*float32_t* y_max*float32_t* x_max*float32_t* score

uint16_t class_id

struct hailo_detections_t

Public Members

uint16_t count

Number of detections

hailo_detection_t detections[0]

Array of detections (it's size is determined by count field)

struct hailo_detection_with_byte_mask_t

Public Members*hailo_rectangle_t* box

Detection's box coordinates

float32_t score

Detection's score

uint16_t class_id

Detection's class id

```
size_t mask_size
```

Mask size in bytes

```
uint8_t *mask
```

Mask offset in bytes from the start of the buffer. The mask is a binary mask that defines a region of interest (ROI) of the image. Mask pixel values of 1 indicate image pixels that belong to the ROI. Mask pixel values of 0 indicate image pixels that are part of the background.

The size of the mask is the size of the box, in the original input image's dimensions. Mask width = $\text{ceil}((\text{box.x_max} - \text{box.x_min}) * \text{image_width})$ Mask height = $\text{ceil}((\text{box.y_max} - \text{box.y_min}) * \text{image_height})$ First pixel represents the pixel $(\text{x_min} * \text{image_width}, \text{y_min} * \text{image_height})$ in the original input image.

```
size_t mask_offset
```

```
struct hailo_stream_write_async_completion_info_t
```

#include <hailort.h> Completion info struct passed to the *hailo_stream_write_async_callback_t* after the async operation is done or has failed.

Public Members

```
hailo_status status
```

Status of the async transfer:

- *HAILO_SUCCESS* - The transfer is complete.
- *HAILO_STREAM_ABORT* - The transfer was canceled (can happen after network deactivation).
- Any other *hailo_status* on unexpected errors.

```
const void *buffer_addr
```

Address of the buffer passed to the async operation

```
size_t buffer_size
```

Size of the buffer passed to the async operation.

```
void *opaque
```

User specific data. Can be used as a context for the callback.

```
struct hailo_stream_read_async_completion_info_t
```

#include <hailort.h> Completion info struct passed to the *hailo_stream_read_async_callback_t* after the async operation is done or has failed.

Public Members

```
hailo_status status
```

Status of the async transfer:

- *HAILO_SUCCESS* - The transfer is complete.
- *HAILO_STREAM_ABORT* - The transfer was canceled (can happen after network deactivation).
- Any other *hailo_status* on unexpected errors.

```
void *buffer_addr
```

Address of the buffer passed to the async operation

```
size_t buffer_size  
Size of the buffer passed to the async operation.
```

```
void *opaque  
User specific data. Can be used as a context for the callback.
```

```
struct hailo_stream_info_t
```

```
#include <hailort.h> Input or output stream information. In case of multiple inputs or outputs, each one has its own stream.
```

Public Members

```
hailo_3d_image_shape_t shape
```

```
hailo_3d_image_shape_t hw_shape
```

```
hailo_nms_info_t nms_info
```

```
union hailo_stream_info_t::[anonymous] [anonymous]
```

```
uint32_t hw_data_bytes
```

```
uint32_t hw_frame_size
```

```
hailo_format_t format
```

```
hailo_stream_direction_t direction
```

```
uint8_t index
```

```
char name[((128))]
```

```
hailo_quant_info_t quant_info
```

```
bool is_mux
```

```
struct hailo_vstream_info_t
```

```
#include <hailort.h> Input or output vstream information.
```

Public Members

```
char name[((128))]
```

```
char network_name[((((128)) + 1 + (128)))
```

```
hailo_stream_direction_t direction
```

```
hailo_format_t format
```

```
hailo_3d_image_shape_t shape  
  
hailo_nms_shape_t nms_shape  
  
union hailo_vstream_info_t::[anonymous] [anonymous]  
  
hailo_quant_info_t quant_info
```

```
struct hailo_network_parameters_t
```

Public Members

```
uint16_t batch_size
```

This parameter determines the number of frames that will be sent for inference in a single batch. If a scheduler is enabled, this parameter determines the 'burst size' - the max number of frames after which the scheduler will attempt to switch to another model. If scheduler is disabled, the number of frames for inference should be a multiplication of batch_size (unless model is in single context).

User is advised to modify this (single network parameter) or *hailo_configure_network_group_params_t* batch size parameter. Not both. In case user wishes to work with the same batch size for all networks inside a network group, user is advised to set batch_size in *hailo_configure_network_group_params_t*. In case user wished to work with batch size per network, user is advised to use this parameter.

Note: The default value is *HAILO_DEFAULT_BATCH_SIZE* - which means the batch is determined by HailoRT automatically.

```
struct hailo_network_parameters_by_name_t
```

Public Members

```
char name[((128)) + 1 + (128))]
```

```
hailo_network_parameters_t network_params
```

```
struct hailo_configure_network_group_params_t
```

```
#include <hailort.h> Hailo configure parameters per network_group
```

Public Members

```
char name[((128))]
```

```
uint16_t batch_size
```

This parameter is only used in multi-context network_groups. In case of name missmatch, default value *HAILO_DEFAULT_BATCH_SIZE* is used

```
hailo_power_mode_t power_mode
```

```
hailo_latency_measurement_flags_t latency
```

```
bool enable_kv_cache
```

```
size_t stream_params_by_name_count  
  
hailo_stream_parameters_by_name_t stream_params_by_name[(40)]  
  
size_t network_params_by_name_count  
  
hailo_network_parameters_by_name_t network_params_by_name[(8)]  
  
struct hailo_configure_params_t  
#include <hailort.h> Hailo configure parameters  
Public Members  
  
size_t network_group_params_count  
  
hailo_configure_network_group_params_t network_group_params[(8)]  
  
struct hailo_activate_network_group_params_t  
#include <hailort.h> Hailo network_group parameters  
Public Members  
  
uint8_t reserved  
  
struct hailo_network_group_info_t  
#include <hailort.h> Hailo network group info  
Public Members  
  
char name[((128))]  
  
bool is_multi_context  
  
struct hailo_layer_name_t  
#include <hailort.h> Hailo layer name  
Public Members  
  
char name[((128))]  
  
struct hailo_network_info_t  
Public Members  
  
char name[((((128)) + 1 + (128)))  
  
struct hailo_rx_error_notification_message_t  
#include <hailort.h> Rx error notification message
```

Public Members

```
uint32_t error  
  
uint32_t queue_number  
  
uint32_t rx_errors_count
```

```
struct hailo_debug_notification_message_t
```

```
#include <hailort.h> Debug notification message
```

Public Members

```
uint32_t connection_status  
  
uint32_t connection_type  
  
uint32_t vdma_is_active  
  
uint32_t host_port  
  
uint32_t host_ip_addr
```

```
struct hailo_health_monitor_dataflow_shutdown_notification_message_t
```

```
#include <hailort.h> Health monitor - Dataflow shutdown notification message
```

Public Members

```
float32_t ts0_temperature  
  
float32_t ts1_temperature
```

```
struct hailo_health_monitor_temperature_alarm_notification_message_t
```

```
#include <hailort.h> Health monitor - Temperature alarm notification message
```

Public Members

```
hailo_temperature_protection_temperature_zone_t temperature_zone  
  
uint32_t alarm_ts_id  
  
float32_t ts0_temperature  
  
float32_t ts1_temperature
```

```
struct hailo_health_monitor_overcurrent_alert_notification_message_t
```

```
#include <hailort.h> Health monitor - Overcurrent alert notification message
```

Public Members

```
hailo_overcurrent_protection_overcurrent_zone_t overcurrent_zone
```

```
float32_t exceeded_alert_threshold
```

```
bool is_last_overcurrentViolation_reached
```

```
struct hailo_health_monitor_lcu_ecc_error_notification_message_t
```

```
#include <hailort.h> Health monitor - LCU ECC error notification message
```

Public Members

```
uint16_t cluster_error
```

```
struct hailo_health_monitor_cpu_ecc_notification_message_t
```

```
#include <hailort.h> Health monitor - CPU ECC error notification message
```

Public Members

```
uint32_t memory_bitmap
```

```
struct hailo_performance_stats_t
```

```
#include <hailort.h> Performance stats (value of '-1' in any field indicates that there was an error retrieving that specific stat)
```

Public Members

```
float32_t cpu_utilization
```

Percentage

```
int64_t ram_size_total
```

kibiBytes (KiBs)

```
int64_t ram_size_used
```

KibiBytes (KiBs)

```
float32_t nnc_utilization
```

Percentage

```
int32_t ddr_noc_total_transactions
```

Per second (not implemented)

```
int32_t dsp_utilization
```

Percentage (round numbers between 1-100)

```
struct hailo_health_stats_t
```

```
#include <hailort.h> Health stats (value of '-1' in any field indicates that there was an error retrieving that specific stat)
```

Public Members

```
float32_t on_die_temperature
```

Degrees celsius

```
int32_t on_die_voltage
```

mV

```
int32_t bist_failure_mask
```

Bit mask

```
struct hailo_context_switch_breakpoint_reached_message_t
```

```
#include <hailort.h> Context switch - breakpoint reached notification message
```

Public Members

```
uint8_t network_group_index
```

```
uint32_t batch_index
```

```
uint16_t context_index
```

```
uint16_t action_index
```

```
struct hailo_health_monitor_clock_changed_notification_message_t
```

```
#include <hailort.h> Health monitor - System's clock has been changed notification message
```

Public Members

```
uint32_t previous_clock
```

```
uint32_t current_clock
```

```
struct hailo_hw_infer_manager_infer_done_notification_message_t
```

Public Members

```
uint32_t infer_cycles
```

```
struct hailo_start_update_cache_offset_notification_message_t
```

Public Members

```
uint64_t cache_id_bitmask
```

```
struct hailo_context_switch_run_time_error_message_t
```

Public Members`uint32_t exit_status``uint8_t network_group_index``uint16_t batch_index``uint16_t context_index``uint16_t action_index``struct hailo_throttling_state_change_message_t`**Public Members**`uint16_t new_state``union hailo_notification_message_parameters_t`

#include <hailort.h> Union of all notification messages parameters. See [hailo_notification_t](#)

Public Members`hailo_rx_error_notification_message_t rx_error_notification`

Ethernet rx error

`hailo_debug_notification_message_t debug_notification`

Internal usage

`hailo_health_monitor_dataflow_shutdown_notification_message_t``health_monitor_dataflow_shutdown_notification`

Dataflow shutdown due to health monitor event

`hailo_health_monitor_temperature_alarm_notification_message_t``health_monitor_temperature_alarm_notification`

Chip temperature alarm

`hailo_health_monitor_overcurrent_alert_notification_message_t``health_monitor_overcurrent_alert_notification`

Chip overcurrent alert

`hailo_health_monitor_lcu_ecc_error_notification_message_t``health_monitor_lcu_ecc_error_notification`

Core ecc error notification

`hailo_health_monitor_cpu_ecc_notification_message_t health_monitor_cpu_ecc_notification`

Chip ecc error notification

`hailo_context_switch_breakpoint_reached_notification_message_t``context_switch_breakpoint_reached_notification`

Internal usage

```
hailo_health_monitor_clock_changed_notification_message_t  
health_monitor_clock_changed_notification  
    Neural network core clock changed due to health monitor event  
  
hailo_hw_infer_manager_infer_done_notification_message_t  
hw_infer_manager_infer_done_notification  
  
hailo_context_switch_run_time_error_message_t context_switch_run_time_error  
    context switch run time error event  
  
hailo_start_update_cache_offset_notification_message_t  
start_update_cache_offset_notification  
    Start cache offset update notification  
  
hailo_throttling_state_change_message_t throttling_state_change  
    Change in throttling-state notification
```

```
struct hailo_notification_t
```

```
#include <hailort.h> Notification data that will be passed to the callback passed in hailo_notification_callback
```

Public Members

```
    hailo_notification_id_t id
```

```
    uint32_t sequence
```

```
    hailo_notification_message_parameters_t body
```

```
struct hailo_chip_temperature_info_t
```

```
#include <hailort.h> Hailo chip temperature info. The temperature is in Celsius.
```

Public Members

```
    float32_t ts0_temperature
```

```
    float32_t ts1_temperature
```

```
    uint16_t sample_count
```

```
struct hailo_throttling_level_t
```

Public Members

```
    float32_t temperature_threshold
```

```
    float32_t hysteresis_temperature_threshold
```

```
    uint32_t throttling_nn_clock_freq
```

```
struct hailo_health_info_t
```

Public Members

```
bool overcurrent_protection_active  
  
uint8_t current_overcurrent_zone  
  
float32_t red_overcurrent_threshold  
  
bool overcurrent_throttling_active  
  
bool temperature_throttling_active  
  
uint8_t current_temperature_zone  
  
int8_t current_temperature_throttling_level  
  
hailo_throttling_level_t temperature_throttling_levels[(4)]  
  
int32_t orange_temperature_threshold  
  
int32_t orange_hysteresis_temperature_threshold  
  
int32_t red_temperature_threshold  
  
int32_t red_hysteresis_temperature_threshold  
  
uint32_t requested_overcurrent_clock_freq  
  
uint32_t requested_temperature_clock_freq  
  
struct hailo_stream_raw_buffer_t
```

Public Members

```
void *buffer  
  
size_t size  
  
struct hailo_stream_raw_buffer_by_name_t
```

Public Members

```
char name[((128))]
```

```
hailo_stream_raw_buffer_t raw_buffer
```

```
struct hailo_latency_measurement_result_t
```

Public Members

```
float64_t avg_hw_latency_ms
```

```
struct hailo_rate_limit_t
```

Public Members

```
char stream_name[((128))]
```

```
uint32_t rate
```

15. HailoRT C++ API Reference

HailoRT is Hailo's runtime library. The C++ API is used for running inference over compiled models (HEFs) in a C++ program.

Note: (for C++ API users) In order to maintain ABI Integrity between libhailort and your application that uses the C++ API, you should *compile libhailort* instead of using the pre-compiled binaries.

15.1. Device and Control API Functions

```
class hailort : :Device  
Represents the Hailo device (chip).
```

Public Types

```
enum Type  
The device type  
Values:  
enumerator PCIE  
  
enumerator ETH  
  
enumerator INTEGRATED
```

Public Functions

```
Expected<NetworkGroupsParamsMap> create_configure_params(Hef &hef) const  
Create the default configure params from an hef.
```

Parameters *hef* – [in] A reference to an *Hef* object to create configure params by

Returns Upon success, returns *Expected* of a NetworkGroupsParamsMap (map of string and ConfiguredNetworkParams). Otherwise, returns *Unexpected* of *hailo_status* error.

```
Expected<ConfigureNetworkParams> create_configure_params(Hef &hef, const std::string ... )  
Create the default configure params from an hef.
```

Parameters

- *hef* – [in] A reference to an *Hef* object to create configure params by
- *network_group_name* – [in] Name of network_group to make configure params for.

Returns Upon success, returns *Expected* of a NetworkGroupsParamsMap (map of string and ConfiguredNetworkParams). Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<ConfiguredNetworkGroupVector> configure(Hef &hef, const ... )  
Configure the device from an hef.
```

Parameters

- *hef* – [in] A reference to an *Hef* object to configure the device by.
- *configure_params* – [in] A map of configured network group name and parameters.

Returns Upon success, returns *Expected* of a vector of configured network groups. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<size_t> *read_log*(MemoryView &buffer, *hailo_cpu_id_t* cpu_id) = 0

Read data from the debug log buffer.

Parameters

- *buffer* - **[in]** A buffer that would receive the debug log data.
- *cpu_id* - **[in]** The cpu source of the debug log.

Returns Upon success, returns *Expected* of the number of bytes that were read. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<size_t> *fetch_logs*(MemoryView buffer, *hailo_log_type_t* log_type) = 0

Fetch halo logs from the Hailo device and returns them as buffer.

Note: Buffer size should be the maximum log size of the device. Use '*Device::get_max_logs_size*' to get it.

Parameters

- *buffer* - **[in]** A buffer that would receive the log data.
- *log_type* - **[in]** The log type to fetch.

Returns Upon success, returns *Expected* of the size in bytes of the log data. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<size_t> *get_max_logs_size*(*hailo_log_type_t* log_type)

Gets the max logs size for the Hailo device (used for '*Device::fetch_logs*').

Parameters *log_type* - **[in]** The log type.

Returns Upon success, the max logs size in bytes. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<*hailo_device_identity_t*> *identify*()

Sends identify control to a Hailo device.

Returns Upon success, returns *Expected* of *hailo_device_identity_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*hailo_core_information_t*> *core_identify*()

Receive information about the core cpu.

Returns Upon success, returns *Expected* of *hailo_core_information_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<*hailo_extended_device_information_t*> *get_extended_device_information*()

Get extended device information about the Hailo device.

Returns Upon success, returns *Expected* of *hailo_extended_device_information_t* containing the extended information about the device. Otherwise, returns *Unexpected* of *hailo_status* error.

hailo_status *set_fw_logger*(*hailo_fw_logger_level_t* level, uint32_t interface_mask)

Configure fw logger level and interface of sending.

Parameters

- *level* - **[in]** The minimum logger level.
- *interface_mask* - **[in]** Output interfaces (mix of *hailo_fw_logger_interface_t*).

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

`hailo_status set_throttling_state(bool should_activate)`

Change throttling state of temperature protection and overcurrent protection components. In case that change throttling state of temperature protection didn't succeed, the change throttling state of overcurrent protection is executed.

Parameters `should_activate - [in]` Should be true to enable or false to disable.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status write_memory(uint32_t address, const MemoryView &data)`

Writes data to device memory.

Parameters

- `address - [in]` The address data would be written to.
- `data - [in]` A buffer that contains the data to be written to the memory.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status read_memory(uint32_t address, MemoryView &data)`

Reads data from device memory.

Parameters

- `address - [in]` The address data would be read from.
- `data - [in]` A buffer that receives the data read from memory.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`Expected<bool> get_throttling_state()`

Get current throttling state of temperature protection and overcurrent protection components. If any throttling is enabled, the function return true.

Returns Upon success, returns `Expected<bool>`, indicates whether the throttling state is active or not. Otherwise, returns `Unexpected<hailo_status>` error.

`hailo_status wd_enable(hailo_cpu_id_t cpu_id)`

Enable firmware watchdog.

Note: Advanced API. Please use with care.

Parameters `cpu_id - [in]` A `hailo_cpu_id_t` indicating which CPU WD to enable.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status wd_disable(hailo_cpu_id_t cpu_id)`

Disable firmware watchdog.

Note: Advanced API. Please use with care.

Parameters `cpu_id - [in]` A `hailo_cpu_id_t` indicating which CPU WD to disable.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns an `hailo_status` error.

`hailo_status wd_config(hailo_cpu_id_t cpu_id, uint32_t wd_cycles, hailo_watchdog_mode_t wd_mode)`

Configure firmware watchdog.

Note: Advanced API. Please use with care.

Parameters

- **cpu_id** - **[in]** A *hailo_cpu_id_t* indicating which CPU WD to configure.
- **wd_cycles** - **[in]** Number of cycles until watchdog is triggered.
- **wd_mode** - **[in]** A *hailo_watchdog_mode_t* indicating which WD mode to configure.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Expected<uint32_t> **previous_system_state**(*hailo_cpu_id_t* cpu_id)

Read the FW previous system state.

Note: Advanced API. Please use with care.

Parameters **cpu_id** - **[in]** A *hailo_cpu_id_t* indicating which CPU to state to read.

Returns Upon success, returns *Expected* of *uint32_t* indicating the previous system state. 0 indicating external reset, 1 indicating WD HW reset, 2 indicating WD SW reset, 3 indicating SW control reset. Otherwise, returns an *hailo_status* error.

hailo_status **set_pause_frames**(bool rx_pause_frames_enable)

Enable/Disable Pause frames.

Parameters **rx_pause_frames_enable** - **[in]** Indicating whether to enable or disable pause frames.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status **i2c_read**(const *hailo_i2c_slave_config_t* &slave_config, uint32_t register_address, ...)

Read data from an I2C slave.

Parameters

- **slave_config** - **[in]** The *hailo_i2c_slave_config_t* configuration of the slave.
- **register_address** - **[in]** The address of the register from which the data will be read.
- **data** - **[in]** A buffer that would store the read data.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status **i2c_write**(const *hailo_i2c_slave_config_t* &slave_config, uint32_t register_address, const ...)

Write data to an I2C slave.

Parameters

- **slave_config** - **[in]** The *hailo_i2c_slave_config_t* configuration of the slave.
- **register_address** - **[in]** The address of the register to which the data will be written.
- **data** - **[in]** A buffer that contains the data to be written to the slave.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

virtual *Expected*<float32_t> **power_measurement**(*hailo_dvm_options_t* dvm, ...)

Perform a single power measurement.

Parameters

- **dvm** - **[in]** Which DVM will be measured. Default (*HAILO_DVM_OPTIONS_AUTO*) will be different according to the board:

- Default (`HAILO_DVM_OPTIONS_AUTO`) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums `HAILO_DVM_OPTIONS_VDD_CORE`, `HAILO_DVM_OPTIONS_MIPI_AVDD` and `HAILO_DVM_OPTIONS_AVDD_H`. Only `HAILO_POWER_MEASUREMENT_TYPES_POWER` can be measured with this option.
- Default (`HAILO_DVM_OPTIONS_AUTO`) for platforms supporting current monitoring (such as M.2 and mPCIe): `OVERCURRENT_PROTECTION`.
- `measurement_type` - **[in]** The type of the measurement. Choosing `HAILO_POWER_MEASUREMENT_TYPES_AUTO` will select the default value according to the supported features.

Returns Upon success, returns `uint32_t` measurement. Measured units are determined due to `hailo_power_measurement_types_t`. Otherwise, returns a `hailo_status` error.

```
virtual hailo_status start_power_measurement(hailo_averaging_factor_t averaging_factor, ...)
```

Start performing a long power measurement.

Parameters

- `averaging_factor` - **[in]** Number of samples per time period, sensor configuration value.
- `sampling_period` - **[in]** Related conversion time, sensor configuration value. The sensor samples the power every sampling_period {us} and averages every averaging_factor samples. The sensor provides a new value every: $(2 * \text{sampling_period} * \text{averaging_factor})\{\text{ms}\}$. The firmware wakes up every interval_milliseconds {ms} and checks the sensor. If there is a new value to read from the sensor, the firmware reads it. Note that the average calculated by the firmware is 'average of averages', because it averages values that have already been averaged by the sensor.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
virtual hailo_status set_power_measurement(hailo_measurement_buffer_index_t buffer_index, ...)
```

Set parameters for long power measurement.

Parameters

- `buffer_index` - **[in]** A `hailo_measurement_buffer_index_t` represents the buffer on the firmware the data would be saved at. Should match the one passed to '`Device::get_power_measurement`'.
- `dvm` - **[in]** Which DVM will be measured. Default (`HAILO_DVM_OPTIONS_AUTO`) will be different according to the board:
 - Default (`HAILO_DVM_OPTIONS_AUTO`) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums `HAILO_DVM_OPTIONS_VDD_CORE`, `HAILO_DVM_OPTIONS_MIPI_AVDD` and `HAILO_DVM_OPTIONS_AVDD_H`. Only `HAILO_POWER_MEASUREMENT_TYPES_POWER` can be measured with this option.
 - Default (`HAILO_DVM_OPTIONS_AUTO`) for platforms supporting current monitoring (such as M.2 and mPCIe): `OVERCURRENT_PROTECTION`.
- `measurement_type` - **[in]** The type of the measurement. Choosing `HAILO_POWER_MEASUREMENT_TYPES_AUTO` will select the default value according to the supported features.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
virtual Expected<hailo_power_measurement_data_t> get_power_measurement(hailo_measurement_buffer_index_t ...)
```

Read measured power from a long power measurement

Parameters

- `buffer_index` - **[in]** A `hailo_measurement_buffer_index_t` represents the buffer on the firmware the data would be saved at. Should match the one passed to '`Device::set_power_measurement`'.

- **should_clear** - **[in]** Flag indicating if the results saved at the firmware will be deleted after reading.

Returns Upon success, returns `hailo_power_measurement_data_t`. Measured units are determined due to `hailo_power_measurement_types_t` passed to '`Device::set_power_measurement`'. Otherwise, returns a `hailo_status` error.

```
virtual hailo_status stop_power_measurement()
```

Stop performing a long power measurement.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
virtual Expected<hailo_chip_temperature_info_t> get_chip_temperature()
```

Get temperature information on the device

Note: Temperature in Celsius of the two internal temperature sensors (TS).

Returns Upon success, returns `hailo_chip_temperature_info_t`, containing temperature information on the device. Otherwise, returns a `hailo_status` error.

```
virtual Expected<hailo_health_stats_t> query_health_stats()
```

Gets health stats of the Hailo device.

Note: Supported only on Hailo-10/Hailo-15 devices running on Linux.

Returns Upon success, returns `hailo_health_stats_t`, containing health information. Otherwise, returns a `hailo_status` error.

```
virtual Expected<hailo_performance_stats_t> query_performance_stats()
```

Gets performance stats of the Hailo device, and of the system it is connected to.

Note: Supported only on Hailo-10/Hailo-15 devices running on Linux.

Returns Upon success, returns `hailo_performance_stats_t`, containing performance information. Otherwise, returns a `hailo_status` error.

```
virtual hailo_status reset(hailo_reset_device_mode_t mode) = 0
```

Reset device.

Note: Calling this function while running other operations on the device (including inference) will lead to unexpected results!

Note: The object used to call this function is not to be used after calling this function! A new instance should be created.

Parameters `mode` - **[in]** The mode of the reset.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

virtual `hailo_status` `set_notification_callback`(const `NotificationCallback` &func, ...)

Sets a callback to be called when a notification with ID `notification_id` will be received.

Parameters

- `func` - **[in]** The callback function to be called.
- `notification_id` - **[in]** The ID of the notification.
- `opaque` - **[in]** User specific data.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

virtual `hailo_status` `remove_notification_callback`(`hailo_notification_id_t` notification_id) = 0

Removes a previously set callback with ID `notification_id`.

Parameters `notification_id` - **[in]** The ID of the notification to remove.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `test_chip_memories`()

Test chip memories using BIST.

Note: Cannot be called during inference!

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `set_sleep_state`(`hailo_sleep_state_t` sleep_state)

Set chip sleep state..

Note: This is an advanced API. Please be advised not to use this API, unless supported by Hailo.

Parameters `sleep_state` - **[in]** The requested sleep state of the chip

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

virtual `hailo_status` `firmware_update`(const `MemoryView` &firmware_binary, bool should_reset) = 0

Update the firmware of a Hailo device.

Note: Calling this function while running other operations on the device (including inference) will lead to unexpected results!

Note: The object used to call this function is not to be used after calling this function! A new instance should be created.

Parameters

- `firmware_binary` - **[in]** The firmware code to be updated to the device.
- `should_reset` - **[in]** Bool indicating whether to reset the device after updating.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
virtual hailo_status second_stage_update(uint8_t *second_stage_binary, uint32_t ...)
```

Update the second stage binary.

Note: Calling this function while running other operations on the device (including inference) will lead to unexpected results!

Note: The object used to call this function is not to be used after calling this function! A new instance should be created.

Parameters

- `second_stage_binary - [in]` The SSB code to be updated to the device.
- `second_stage_binary_length - [in]` The length of the SSB to be updated.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
virtual hailo_status store_sensor_config(uint32_t section_index, hailo_sensor_types_t ...)
```

Store sensor configuration to Hailo chip flash memory.

Parameters

- `section_index - [in]` Flash section index to write to. [0-6]
- `sensor_type - [in]` Sensor type.
- `reset_config_size - [in]` Size of reset configuration.
- `config_height - [in]` Configuration resolution height.
- `config_width - [in]` Configuration resolution width.
- `config_fps - [in]` Configuration FPS.
- `config_file_path - [in]` Sensor configuration file path.
- `config_name - [in]` Sensor configuration name.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
virtual hailo_status store_isp_config(uint32_t reset_config_size, uint16_t config_height, uint16_t ...)
```

Store sensor ISP configuration to Hailo chip flash memory.

Parameters

- `reset_config_size - [in]` Size of reset configuration.
- `config_height - [in]` Configuration resolution height.
- `config_width - [in]` Configuration resolution width.
- `config_fps - [in]` Configuration FPS.
- `isp_static_config_file_path - [in]` ISP static configuration file path.
- `isp_runtime_config_file_path - [in]` ISP runtime configuration file path.
- `config_name - [in]` Sensor configuration name.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
virtual Expected<Buffer> sensor_get_sections_info() = 0
```

Gets the sections information of the sensor.

Returns Upon success, returns `Expected` of a buffer containing the sensor's sections information. Otherwise, returns `Unexpected` of `hailo_status` error.

virtual *hailo_status* sensor_dump_config(uint32_t section_index, const std::string ...)

Dump config of given section index into a csv file.

Parameters

- **section_index - [in]** Flash section index to load config from. [0-7]
- **config_file_path - [in]** File path to dump section configuration into.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* sensor_set_i2c_bus_index(*hailo_sensor_types_t* sensor_type, uint32_t ...)

Set the I2C bus to which the sensor of the specified type is connected.

Parameters

- **sensor_type - [in]** The sensor type.
- **bus_index - [in]** The I2C bus index of the sensor.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* sensor_load_and_start_config(uint32_t section_index) = 0

Load the configuration with I2C in the section index.

Parameters **section_index - [in]** Flash section index to load config from. [0-6]

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* sensor_reset(uint32_t section_index) = 0

Reset the sensor that is related to the section index config.

Parameters **section_index - [in]** Flash section index to load config from. [0-6]

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* sensor_set_generic_i2c_slave(uint16_t slave_address, uint8_t ...)

Set a generic I2C slave for sensor usage.

Parameters

- **slave_address - [in]** The address of the i2c slave.
- **offset_size - [in]** Slave offset size (in bytes).
- **bus_index - [in]** The bus number the i2c slave is connected to.
- **should_hold_bus - [in]** Should hold the bus during the read.
- **slave_endianness - [in]** BIG_ENDIAN or LITTLE_ENDIAN.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *Expected*<Buffer> read_board_config() = 0

Reads board configuration from device.

Returns Upon success, returns *Expected* of a buffer containing the data read. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *hailo_status* write_board_config(const MemoryView &buffer) = 0

Write board configuration to device

Parameters **buffer - [in]** A buffer that contains the data to be written .

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *Expected*<*hailo_fw_user_config_information_t*> examine_user_config() = 0

Gets firmware user configuration information from device.

Returns Upon success, returns *Expected* of *hailo_fw_user_config_information_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<Buffer> read_user_config() = 0
```

Reads firmware user configuration from device.

Returns Upon success, returns *Expected* of a buffer containing the data read. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual hailo_status write_user_config(const MemoryView &buffer) = 0
```

Write firmware user configuration to device.

Parameters *buffer* - **[in]** A buffer that contains the data to be written .

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

```
virtual hailo_status erase_user_config() = 0
```

Erase firmware user configuration from the device.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

```
virtual Expected<hailo_device_architecture_t> get_architecture() const = 0
```

Gets the device architecture.

Returns Upon success, returns *Expected* of *hailo_device_architecture_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
Type get_type() const
```

Gets the device type.

Returns Upon success, returns Type. Otherwise, returns a *hailo_status* error.

```
virtual const char *get_dev_id() const = 0
```

Gets the device id.

Returns An identification string of the device. For Pcie device, returns the BDF. For Ethernet device, returns the IP address. For Core device, returns "Core".

```
Expected<hailo_stream_interface_t> get_default_streams_interface() const
```

Gets the stream's default interface.

Returns Upon success, returns *Expected* of *hailo_stream_interface_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual bool is_stream_interface_supported(const hailo_stream_interface_t ...)
```

Returns true if the stream's interface is supported, false otherwise.

```
virtual hailo_status dma_map(void *address, size_t size, hailo_dma_buffer_direction_t direction)
```

Maps the buffer pointed to by *address* for DMA transfers to/from this device, in the specified *data_direction*. DMA mapping of buffers in advance may improve the performance of async API. This improvement will become apparent when the buffer is reused multiple times across different async operations.

For high level API (aka *InferModel*), buffers bound using *ConfiguredInferModel::Bindings::InferStream::set_buffer* can be mapped.

For low level API (aka *InputStream*/*OutputStream*), buffers passed to *InputStream::write_async* and *OutputStream::read_async* can be mapped.

Note: The DMA mapping will be released upon calling *dma_unmap()* with *address*, *size* and *data_direction*, or when the *VDevice* object is destroyed.

Note: The buffer pointed to by *address* cannot be released until it is unmapped (via *dma_unmap()* or *Device* destruction).

Parameters

- **address** - **[in]** The address of the buffer to be mapped.
- **size** - **[in]** The buffer's size in bytes.
- **direction** - **[in]** The direction of the mapping. For input streams, use `HAILO_DMA_BUFFER_DIRECTION_H2D` and for output streams, use `HAILO_DMA_BUFFER_DIRECTION_D2H`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `halo_status` error.

```
virtual halo_status dma_unmap(void *address, size_t size, halo_dma_buffer_direction_t direction)
```

Un-maps a buffer buffer pointed to by `address` for DMA transfers to/from this device, in the direction `direction`.

Parameters

- **address** - **[in]** The address of the buffer to be un-mapped.
- **size** - **[in]** The buffer's size in bytes.
- **direction** - **[in]** The direction of the mapping.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `halo_status` error.

```
virtual halo_status dma_map_dmabuf(int dmabuf_fd, size_t size, halo_dma_buffer_direction_t direction)
```

Maps the dmabuf represented by the file descriptor `dmabuf_fd` for DMA transfers to/from this device, in the specified `data_direction`. DMA mapping of buffers in advance may improve the performance of async API. This improvement will become apparent when the buffer is reused multiple times across different async operations.

For high level API (aka `InferModel`), buffers bound using `ConfiguredInferModel::Bindings::InferStream::set_buffer` can be mapped.

For low level API (aka `InputStream`/`OutputStream`), buffers passed to `InputStream::write_async` and `OutputStream::read_async` can be mapped.

Note: The DMA mapping will be released upon calling `dma_unmap()` with `dmabuf_fd`, `size` and `data_direction`, or when the `Device` object is destroyed.

Parameters

- **dmabuf_fd** - **[in]** The file descriptor of the dmabuf to be mapped.
- **size** - **[in]** The buffer's size in bytes.
- **direction** - **[in]** The direction of the mapping. For input streams, use `HAILO_DMA_BUFFER_DIRECTION_H2D` and for output streams, use `HAILO_DMA_BUFFER_DIRECTION_D2H`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `halo_status` error.

```
virtual halo_status dma_unmap_dmabuf(int dmabuf_fd, size_t size, halo_dma_buffer_direction_t direction)
```

Un-maps a dmabuf buffer represented by the file descriptor `dmabuf_fd` for DMA transfers to/from this device, in the direction `direction`.

Parameters

- **dmabuf_fd** - **[in]** The file descriptor of the dmabuf to be un-mapped.
- **size** - **[in]** The buffer's size in bytes.
- **direction** - **[in]** The direction of the mapping.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `halo_status` error.

Expected<Capabilities> `get_capabilities()`

Gets a struct specifying the device's capabilities.

Returns Upon success, returns *Expected* of *Capabilities*. Otherwise, returns *Unexpected* of *hailo_status* error.

Public Static Functions

`static Expected<std::vector<std::string>> scan()`

Returns the device_id string on all available devices in the system. The device id is a unique identifier for the device on the system.

Note: ethernet devices are not considered "devices in the system", so they are not scanned in this function. use :scan_eth for ethernet devices.

Returns Upon success, returns *Expected* of a vector of std::string containing the information. Otherwise, returns *Unexpected* of *hailo_status* error.

`static Expected<std::vector<hailo_pcie_device_info_t>> scan_PCIE()`

Returns information on all available pcie devices in the system.

Returns Upon success, returns *Expected* of a vector of *hailo_pcie_device_info_t* containing the information. Otherwise, returns *Unexpected* of *hailo_status* error.

`static Expected<std::unique_ptr<Device>> create()`

Creates a device. If there are more than one device detected in the system, an arbitrary device is returned.

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

`static Expected<std::unique_ptr<Device>> create(const std::string &device_id)`

Creates a device by the given device id.

Parameters `device_id - [in]` Device id string, can represent several device types: [-] for pcie devices - pcie bdf (XXXX:XX:XX.X) [-] for ethernet devices - ip address (xxx.xxx.xxx.xxx)

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

`static Expected<std::unique_ptr<Device>> create_PCIE()`

Creates pcie device. If there are more than one device detected in the system, an arbitrary pcie device is returned.

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

`static Expected<std::unique_ptr<Device>> create_PCIE(const hailo_pcie_device_info_t &device_info)`

Creates a PCIe device by the given info.

Parameters `device_info - [in]` Information about the device to open.

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

`static Expected<hailo_pcie_device_info_t> parse_PCIE_device_info(const std::string ...)`

Parse PCIe device BDF string into hailo device info structure.

Parameters `device_info_str - [in]` BDF device info, format <domain>.<bus>.<device>.<func>.

Returns Upon success, returns *Expected* of *hailo_pcie_device_info_t* containing the information. Otherwise, returns *Unexpected* of *hailo_status* error.

```
static Expected<std::string> pcie_device_info_to_string(const hailo_PCIE_Device_Info_t ...)
```

Returns a string of pcie device info.

Parameters `device_info - [in]` A *hailo_PCIE_Device_Info_t* containing the pcie device information.

Returns Upon success, returns *Expected* of a string containing the information. Otherwise, returns *Unexpected* of *hailo_Status* error.

```
static Expected<Type> get_device_type(const std::string &device_id)
```

Returns the device type of the given device id string.

Parameters `device_id - [in]` A std::string device id to check.

Returns Upon success, returns *Expected* of the device type. Otherwise, returns *Unexpected* of *hailo_Status* error.

```
static bool device_ids_equal(const std::string &first, const std::string &second)
```

Checks if 2 device ids represents the same device.

Parameters

- `first - [in]` A std::string first device id to check.
- `second - [in]` A std::string second device id to check.

Returns true if the device ids represents the same device.

```
struct Capabilities
```

The device supported capabilities

15.2. VDevice API Functions

```
class hailort :: VDevice
```

Represents a bundle of physical devices.

Public Functions

```
virtual Expected<ConfiguredNetworkGroupVector> configure(Hef &hef, const ...)
```

Configures the vdevice from an hef.

Parameters

- `hef - [in]` A reference to an *Hef* object to configure the vdevice by.
- `configure_params - [in]` A map of configured network group name and parameters.

Returns Upon success, returns *Expected* of a vector of configured network groups. Otherwise, returns *Unexpected* of *hailo_Status* error.

```
virtual Expected<std::shared_ptr<InferModel>> create_infer_model(const std::string &hef_path, ...)
```

Creates the infer model from an hef

Parameters

- `hef_path - [in]` A string of an hef file.
- `name - [in]` A string of the model name (optional).

Returns Upon success, returns *Expected* of a shared pointer of infer model. Otherwise, returns *Unexpected* of *hailo_Status* error.

virtual *Expected*<std::shared_ptr<*InferModel*>> **create_infer_model**(const MemoryView hef_buffer, ...)
Creates the infer model from an hef buffer

Note: During *Hef* creation, the hef_buffer's content is copied to an internal buffer.

Parameters

- **hef_buffer** - **[in]** A pointer to a buffer containing the hef file.
- **name** - **[in]** A string of the model name (optional).

Returns Upon success, returns *Expected* of a shared pointer of infer model. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::shared_ptr<*InferModel*>> **create_infer_model**(std::shared_ptr<Buffer> ...)
Creates the infer model from an hef buffer

Parameters

- **hef_buffer** - **[in]** A pointer to a buffer containing the hef file.
- **name** - **[in]** A string of the model name (optional).

Returns Upon success, returns *Expected* of a shared pointer of infer model. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::shared_ptr<*InferModel*>> **create_infer_model**(*Hef* hef, const std::string ...)
Creates the infer model from an hef

Parameters

- **hef** - **[in]** A *Hef* object
- **name** - **[in]** A string of the model name (optional).

Returns Upon success, returns *Expected* of a shared pointer of infer model. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<std::reference_wrapper<*Device*>>> **get_physical_devices**() const = 0
Gets the underlying physical devices.

Note: The returned physical devices are held in the scope of *vdevice*.

Returns Upon success, returns *Expected* of a vector of device objects. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<std::string>> **get_physical_devices_ids**() const = 0
Gets the physical device IDs.

Returns Upon success, returns *Expected* of a vector of std::string device ids objects. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<*hailo_stream_interface_t*> **get_default_streams_interface**() const = 0
Gets the stream's default interface.

Returns Upon success, returns *Expected* of *hailo_stream_interface_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<NetworkGroupsParamsMap> **create_configure_params**(*Hef* &hef) const
Create the default configure params from an hef.

Parameters **hef** - **[in]** A reference to an *Hef* object to create configure params by

Returns Upon success, returns *Expected* of a NetworkGroupsParamsMap (map of string and ConfiguredNetworkParams). Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<ConfigureNetworkParams> `create_configure_params(Hef &hef, const std::string ...)`

Create the default configure params from an hef.

Parameters

- `hef` - **[in]** A reference to an *Hef* object to create configure params by
- `network_group_name` - **[in]** Name of network_group to make configure params for.

Returns Upon success, returns *Expected* of a *ConfigureNetworkParams*. Otherwise, returns *Unexpected* of *hailo_status* error.

`virtual hailo_status dma_map(void *address, size_t size, hailo_dma_buffer_direction_t direction) = 0`

Maps the buffer pointed to by *address* for DMA transfers to/from this vdevice, in the specified *data_direction*. DMA mapping of buffers in advance may improve the performance of async API. This improvement will become apparent when the buffer is reused multiple times across different async operations.

For high level API (aka *InferModel*), buffers bound using *ConfiguredInferModel::Bindings::InferStream::set_buffer* can be mapped.

For low level API (aka *InputStream/OutputStream*), buffers passed to *InputStream::write_async* and *OutputStream::read_async* can be mapped.

Note: The DMA mapping will be released upon calling *dma_unmap()* with *address*, *size* and *data_direction*, or when the *VDevice* object is destroyed.

Note: The buffer pointed to by *address* cannot be released until it is unmapped (via *dma_unmap()* or *VDevice* destruction).

Parameters

- `address` - **[in]** The address of the buffer to be mapped.
- `size` - **[in]** The buffer's size in bytes.
- `direction` - **[in]** The direction of the mapping. For input streams, use `HAILO_DMA_BUFFER_DIRECTION_H2D` and for output streams, use `HAILO_DMA_BUFFER_DIRECTION_D2H`.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

`virtual hailo_status dma_unmap(void *address, size_t size, hailo_dma_buffer_direction_t direction) = 0`

Un-maps a buffer buffer pointed to by *address* for DMA transfers to/from this vdevice, in the direction *direction*.

Parameters

- `address` - **[in]** The address of the buffer to be un-mapped.
- `size` - **[in]** The buffer's size in bytes.
- `direction` - **[in]** The direction of the mapping.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* *dma_map_dmabuf* (int *dmabuf_fd*, *size_t* *size*, *hailo_dma_buffer_direction_t* *direction*) = 0

Maps the dmabuf represented by the file descriptor *dmabuf_fd* for DMA transfers to/from this vdevice, in the specified *data_direction*. DMA mapping of buffers in advance may improve the performance of async API. This improvement will become apparent when the buffer is reused multiple times across different async operations.

For high level API (aka *InferModel*), buffers bound using *ConfiguredInferModel::Bindings::InferStream::set_buffer* can be mapped.

For low level API (aka *InputStream*/*OutputStream*), buffers passed to *InputStream::write_async* and *OutputStream::read_async* can be mapped.

Note: The DMA mapping will be released upon calling *dma_unmap_dmabuf()* with *dmabuf_fd*, *size* and *data_direction*, or when the *VDevice* object is destroyed.

Parameters

- *dmabuf_fd* - **[in]** The file descriptor of the dmabuf to be mapped.
- *size* - **[in]** The buffer's size in bytes.
- *direction* - **[in]** The direction of the mapping. For input streams, use *HAILO_DMA_BUFFER_DIRECTION_H2D* and for output streams, use *HAILO_DMA_BUFFER_DIRECTION_D2H*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* *dma_unmap_dmabuf* (int *dmabuf_fd*, *size_t* *size*, *hailo_dma_buffer_direction_t* ...)

Un-maps a dmabuf buffer represented by the file descriptor *dmabuf_fd* for DMA transfers to/from this vdevice, in the direction *direction*.

Parameters

- *dmabuf_fd* - **[in]** The file descriptor of the dmabuf to be un-mapped.
- *size* - **[in]** The buffer's size in bytes.
- *direction* - **[in]** The direction of the mapping.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Public Static Functions

static *Expected*<std::unique_ptr<*VDevice*>> *create* (const *hailo_vdevice_params_t* &*params*)

Creates a vdevice.

Parameters *params* - **[in]** A *hailo_vdevice_params_t*.

Returns Upon success, returns *Expected* of a unique_ptr to *VDevice* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::shared_ptr<*VDevice*>> *create_shared* (const *hailo_vdevice_params_t* &*params*)

Creates a vdevice.

Parameters *params* - **[in]** A *hailo_vdevice_params_t*.

Returns Upon success, returns *Expected* of a shared_ptr to *VDevice* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*VDevice*>> *create* ()

Creates a vdevice.

Note: Calling this create method will apply default vdevice params.

Returns Upon success, returns *Expected* of a unique_ptr to *VDevice* object. Otherwise, returns *Unexpected* of *hailo_status* error.

```
static Expected<std::shared_ptr<VDevice>> create_shared()
```

Creates a vdevice.

Note: Calling this create method will apply default vdevice params.

Returns Upon success, returns *Expected* of a shared_ptr to *VDevice* object. Otherwise, returns *Unexpected* of *hailo_status* error.

```
static Expected<std::unique_ptr<VDevice>> create(const std::vector<std::string> &device_ids)
```

Creates a vdevice from the given physical device ids.

Note: Calling this create method will apply default vdevice params.

Parameters *device_ids* - **[in]** A vector of std::string, represents the device-ids from which to create the *VDevice*.

Returns Upon success, returns *Expected* of a unique_ptr to *VDevice* object. Otherwise, returns *Unexpected* of *hailo_status* error.

```
static Expected<std::shared_ptr<VDevice>> create_shared(const std::vector<std::string> &device_ids)
```

Creates a vdevice from the given physical device ids.

Note: Calling this create method will apply default vdevice params.

Parameters *device_ids* - **[in]** A vector of std::string, represents the device-ids from which to create the *VDevice*.

Returns Upon success, returns *Expected* of a shared_ptr to *VDevice* object. Otherwise, returns *Unexpected* of *hailo_status* error.

15.3. HEF API Definitions and Functions

```
struct hailort::ConfigureNetworkParams
```

Hailo configure parameters per network_group. Analogical to *hailo_configure_network_group_params_t*

Public Functions

```
ConfigureNetworkParams() = default
```

```
inline ConfigureNetworkParams(const hailo_configure_network_group_params_t &params)
```

```
bool operator==(const ConfigureNetworkParams &other) const
```

```
bool operator!=(const ConfigureNetworkParams &other) const
```

Public Members

```
uint16_t batch_size  
  
hailo_power_mode_t power_mode  
  
hailo_latency_measurement_flags_t latency  
  
bool enable_kv_cache  
  
std::map<std::string, hailo_stream_parameters_t> stream_params_by_name  
  
std::map<std::string, hailo_network_parameters_t> network_params_by_name
```

class **hailort** :: Hef

HEF model that can be loaded to Hailo devices

Public Functions

Expected<std::vector<*hailo_stream_info_t*>> get_input_stream_infos (const std::string &name = ...)

Gets input streams informations.

Parameters name - **[in]** The name of the network or network_group which contains the input stream_infos. In case network group name is given, the function returns the input stream infos of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns the input stream infos of the given network. If NULL is passed, the function returns the input stream infos of all the networks of the first network group.

Returns Upon success, returns a vector of *hailo_stream_info_t*, containing each stream's information. Otherwise, returns a *hailo_status* error.

Expected<std::vector<*hailo_stream_info_t*>> get_output_stream_infos (const std::string &name = ...)

Gets output streams informations.

Parameters name - **[in]** The name of the network or network_group which contains the output stream_infos. In case network group name is given, the function returns the output stream infos of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns the output stream infos of the given network. If NULL is passed, the function returns the output stream infos of all the networks of the first network group.

Returns Upon success, returns a vector of *hailo_stream_info_t*, containing each stream's information. Otherwise, returns a *hailo_status* error.

Expected<std::vector<*hailo_stream_info_t*>> get_all_stream_infos (const std::string &name = "") const

Gets all streams informations.

Parameters name - **[in]** The name of the network or network_group which contains the stream_infos. In case network group name is given, the function returns all stream infos of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns all stream infos of the given network. If NULL is passed, the function returns all the stream infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of *hailo_stream_info_t*, containing each stream's information. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<hailo_stream_info_t> get_stream_info_by_name(const std::string &stream_name, ...)

Gets stream's information from it's name.

Parameters

- **stream_name** - **[in]** The name of the stream as presented in the [Hef](#).
- **stream_direction** - **[in]** Indicates the stream direction.
- **net_group_name** - **[in]** The name of the network_group which contains the stream's information. If not passed, the first network_group in the [Hef](#) will be addressed.

Returns Upon success, returns *Expected* of [hailo_stream_info_t](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<std::vector<[hailo_vstream_info_t](#)>> get_input_vstream_infos(const std::string &name = ...)

Gets input virtual streams infos.

Parameters **name** - **[in]** The name of the network or network_group which contains the input virtual stream_infos. In case network group name is given, the function returns the input virtual stream infos of all the networks of the given network group. In case network name is given (provided by [get_network_infos](#)), the function returns the input virtual stream infos of the given network. If NULL is passed, the function returns the input virtual stream infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of [hailo_vstream_info_t](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<std::vector<[hailo_vstream_info_t](#)>> get_output_vstream_infos(const std::string &name ...)

Gets output virtual streams infos.

Parameters **name** - **[in]** The name of the network or network_group which contains the output virtual stream_infos. In case network group name is given, the function returns the output virtual stream infos of all the networks of the given network group. In case network name is given (provided by [get_network_infos](#)), the function returns the output virtual stream infos of the given network. If NULL is passed, the function returns the output virtual stream infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of [hailo_vstream_info_t](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<std::vector<[hailo_vstream_info_t](#)>> get_all_vstream_infos(const std::string &name = ...)

Gets all virtual streams infos.

Parameters **name** - **[in]** The name of the network or network_group which contains the virtual stream_infos. In case network group name is given, the function returns all virtual stream infos of all the networks of the given network group. In case network name is given (provided by [get_network_infos](#)), the function returns all virtual stream infos of the given network. If NULL is passed, the function returns all the virtual stream infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of [hailo_vstream_info_t](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<std::vector<std::string>> get_sorted_output_names(const std::string ...)

Gets sorted output vstreams names.

Parameters **net_group_name** - **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the [Hef](#) will be addressed.

Returns Upon success, returns *Expected* of a sorted vector of output vstreams names. Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<size_t> get_number_of_input_streams(const std::string &net_group_name = "") const

Gets the number of low-level input streams.

Parameters `net_group_name` - **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the `Hef` will be addressed.

Returns Upon success, returns `Expected` containing the number of low-level input streams. Otherwise, returns `Unexpected` of `hailo_status` error.

`Expected<size_t> get_number_of_output_streams (const std::string &net_group_name = "") const`
Gets the number of low-level output streams.

Parameters `net_group_name` - **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the `Hef` will be addressed.

Returns Upon success, returns `Expected` containing the number of low-level output streams. Otherwise, returns `Unexpected` of `hailo_status` error.

`Expected<float64_t> get_bottleneck_fps (const std::string &net_group_name = "") const`
Gets bottleneck FPS.

Parameters `net_group_name` - **[in]** The name of the network_group which contains the information. If not passed, the first network_group in the `Hef` will be addressed.

Returns Upon success, returns `Expected` containing the bottleneck FPS number. Otherwise, returns `Unexpected` of `hailo_status` error.

`Expected<hailo_device_architecture_t> get_hef_device_arch() const`
Get device Architecture HEF was compiled for.

Deprecated:

Use `get_compatible_device_archs` instead.

Returns Upon success, returns `Expected` containing the device architecture the HEF was compiled for. Otherwise, returns `Unexpected` of `hailo_status` error.

`Expected<std::vector<hailo_device_architecture_t>> get_compatible_device_archs() const`
Get all device architectures that the HEF is compatible with.

Returns Upon success, returns `Expected` of a vector of `hailo_device_architecture_t`. Otherwise, returns `Unexpected` of `hailo_status` error.

`Expected<std::vector<std::string>> get_stream_names_from_vstream_name (const std::string ...)`
Gets all stream names under the given vstream name

Parameters

- `vstream_name` - **[in]** The name of the vstream.
- `net_group_name` - **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the `Hef` will be addressed.

Returns Upon success, returns `Expected` of a vector of all stream names linked to the provided vstream. Otherwise, returns `Unexpected` of `hailo_status` error.

`Expected<std::vector<std::string>> get_vstream_names_from_stream_name (const std::string ...)`
Get all vstream names under the given stream name

Parameters

- `stream_name` - **[in]** The name of the low-level stream.
- `net_group_name` - **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the `Hef` will be addressed.

Returns Upon success, returns `Expected` of a vector of all stream names linked to the provided vstream.

Returns Upon success, returns of a vector of all vstream names linked to the provided stream. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::string> `get_vstream_name_from_original_name(const std::string ...)`

Gets vstream name from original layer name.

Parameters

- `original_name` - **[in]** The original layer name as presented in the *Hef*.
- `net_group_name` - **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* containing the vstream's name. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::vector<std::string>> `get_original_names_from_vstream_name(const ...)`

Gets original names from vstream name.

Parameters

- `vstream_name` - **[in]** The name of the vstream as presented in the *Hef*.
- `net_group_name` - **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* of a vector of all original names linked to the provided vstream. Otherwise, returns *Unexpected* of *hailo_status* error.

`std::vector<std::string> get_network_groups_names()` const

Gets all network groups names in the *Hef*.

Returns Returns a vector of all network groups names.

Expected<std::vector<*hailo_network_group_info_t*>> `get_network_groups_infos()` const

Gets all network groups infos in the *Hef*.

Returns Upon success, returns *Expected* of a vector of *hailo_network_group_info_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<NetworkGroupsParamsMap> `create_configure_params(hailo_stream_interface_t ...)`

Creates the default configure params for the *Hef*. The user can modify the given params before configuring the network.

Parameters `stream_interface` - **[in]** Stream interface to use on the network.

Returns Upon success, returns *Expected* of NetworkGroupsParamsMap. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<ConfigureNetworkParams> `create_configure_params(hailo_stream_interface_t ...)`

Creates the default configure params for the *Hef*. The user can modify the given params before configuring the network.

Parameters

- `stream_interface` - **[in]** Stream interface to use on the network.
- `network_group_name` - **[in]** Name of network_group to make configure params for.

Returns Upon success, returns *Expected* of *ConfigureNetworkParams*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::map<std::string, *hailo_stream_parameters_t*>> `create_stream_parameters_by_name(const ...)`

Creates streams params with default values.

Parameters

- **net_group_name** - **[in]** The name of the network_group for which to create the stream parameters for. If an empty string is given, the first network_group in the [Hef](#) will be addressed.
- **stream_interface** - **[in]** A [*halo_stream_interface_t*](#) indicating which [*halo_stream_parameters_t*](#) to create for the output streams.

Returns Upon success, returns [Expected](#) of a map of stream name to stream params. Otherwise, returns [Unexpected](#) of [*halo_status*](#) error.

[*Expected*<std::map<std::string, *halo_network_parameters_t*>>](#) **create_network_parameters_by_name**(const ...)

Creates networks params with default values.

Parameters **net_group_name** - **[in]** The name of the network_group for which to create the network parameters for. If an empty string is given, the first network_group in the [Hef](#) will be addressed.

Returns Upon success, returns [Expected](#) of a map of network name to network params. Otherwise, returns [Unexpected](#) of [*halo_status*](#) error.

[*Expected*<std::map<std::string, *halo_vstream_params_t*>>](#) **make_input_vstream_params**(const ...)

Creates input virtual stream params for a given network_group.

Parameters

- **name** - **[in]** The name of the network or network_group which user wishes to create input virtual stream params for. In case network group name is given, the function returns the input virtual stream params of all the networks of the given network group. In case network name is given (provided by [get_network_infos](#)), the function returns the input virtual stream params of the given network. If NULL is passed, the function returns the input virtual stream params of all the networks of the first network group.
- **unused** - **[in]** Unused.
- **format_type** - **[in]** The default format type for all input virtual streams.
- **timeout_ms** - **[in]** The default timeout in milliseconds for all input virtual streams.
- **queue_size** - **[in]** The default queue size for all input virtual streams.

Returns Upon success, returns [Expected](#) of a map of input virtual stream name to params. Otherwise, returns [Unexpected](#) of [*halo_status*](#) error.

[*Expected*<std::map<std::string, *halo_vstream_params_t*>>](#) **make_output_vstream_params**(const ...)

Creates output virtual stream params for a given network_group.

Parameters

- **name** - **[in]** The name of the network or network_group which user wishes to create output virtual stream params for. In case network group name is given, the function returns the output virtual stream params of all the networks of the given network group. In case network name is given (provided by [get_network_infos](#)), the function returns the output virtual stream params of the given network. If NULL is passed, the function returns the output virtual stream params of all the networks of the first network group.
- **unused** - **[in]** Unused.
- **format_type** - **[in]** The default format type for all output virtual streams.
- **timeout_ms** - **[in]** The default timeout in milliseconds for all output virtual streams.
- **queue_size** - **[in]** The default queue size for all output virtual streams.

Returns Upon success, returns [Expected](#) of a map of output virtual stream name to params. Otherwise, returns [Unexpected](#) of [*halo_status*](#) error.

Expected<std::vector<*hailo_network_info_t*>> **get_network_infos**(const std::string &net_group_name ...)

Gets all networks informations.

Parameters *net_group_name* - **[in]** The name of the network_group which contains the network information. If NULL is passed, the function returns the network infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of *hailo_network_info_t*, containing each networks's information. Otherwise, returns *Unexpected* of *hailo_status* error.

std::string **hash**() const

Returns a unique hash for the specific *Hef*.

Returns A unique string hash for the *Hef*. Hefs created based on identical files will return identical hashes.

Public Static Functions

static *Expected*<*Hef*> **create**(const std::string &hef_path)

Creates an *Hef* from a file.

Parameters *hef_path* - **[in]** The path of the *Hef* file.

Returns Upon success, returns *Expected* of *Hef*. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<*Hef*> **create**(const MemoryView &hef_buffer)

Creates an *Hef* from a buffer.

Note: During *Hef* creation, the buffer's content is copied to an internal buffer.

Parameters *hef_buffer* - **[in]** A buffer that contains the *Hef* content.

Returns Upon success, returns *Expected* of *Hef*. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<*Hef*> **create**(std::shared_ptr<Buffer> hef_buffer)

Creates an *Hef* from a buffer.

Parameters *hef_buffer* - **[in]** A buffer that contains the *Hef* content.

Returns Upon success, returns *Expected* of *Hef*. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::string> **device_arch_to_string**(const *hailo_device_architecture_t* arch)

Get string of device architecture HEF was compiled for.

Parameters *arch* - **[in]** *hailo_device_architecture_t* representing the device architecture of the HEF

Returns Upon success, returns string representing the device architecture the HEF was compiled for. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::string> **hash**(const std::string &hef_path)

Returns a unique hash for the specific *Hef* file.

Parameters *hef_path* - **[in]** The path of the *Hef* file.

Returns A unique string hash for the *Hef* file. Hefs created based on identical files will return identical hashes. Otherwise, returns *Unexpected* of *hailo_status* error.

struct *HefChunksInfo*

```
struct HefParseForTransferResult
```

15.4. Network group API Definitions and Functions

```
class hailort::ActivatedNetworkGroup
```

Activated network_group that can be used to send/receive data

Public Functions

```
virtual const std::string &get_network_group_name() const = 0
```

Returns The network group name.

```
virtual uint32_t get_invalid_frames_count() = 0
```

Returns The number of invalid frames.

```
class hailort::ConfiguredNetworkGroup
```

Loaded network_group that can be activated

Public Functions

```
virtual const std::string &get_network_group_name() const = 0
```

Returns The network group name.

```
virtual const std::string &name() const = 0
```

Returns The network group name.

```
virtual Expected<hailo\_stream\_interface\_t> get_default_streams_interface() = 0
```

Gets the stream's default interface.

Returns Upon success, returns [Expected](#) of [hailo_stream_interface_t](#). Otherwise, returns [Unexpected](#) of [hailo_status](#) error.

```
virtual std::vector<std::reference_wrapper<InputStream>> get_input_streams_by_interface(hailo\_stream\_interface\_t) = 0
```

Gets all input streams with the given *interface*.

Parameters *stream_interface* – [in] A [hailo_stream_interface_t](#) indicating which [InputStream](#) to return.

Returns Upon success, returns a vector of [InputStream](#).

```
virtual std::vector<std::reference_wrapper<OutputStream>> get_output_streams_by_interface(hailo\_stream\_interface\_t) = 0
```

Gets all output streams with the given *interface*.

Parameters *stream_interface* – [in] A [hailo_stream_interface_t](#) indicating which [OutputStream](#) to return.

Returns Upon success, returns a vector of [OutputStream](#).

```
virtual ExpectedRef<InputStream> get_input_stream_by_name(const std::string &name) = 0
```

Gets input stream by stream name.

Parameters *name* – [in] The name of the input stream to retrieve.

Returns Upon success, returns [ExpectedRef](#) of [InputStream](#). Otherwise, returns [Unexpected](#) of [hailo_status](#) error.

```
virtual ExpectedRef<OutputStream> get_output_stream_by_name(const std::string &name) = 0
```

Gets output stream by stream name.

Parameters name - [in] The name of the input stream to retrieve.

Returns Upon success, returns ExpectedRef of *OutputStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<InputStreamRefVector> get_input_streams_by_network(const std::string ...)
```

Parameters network_name - [in] Network name of the requested input streams. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of vector *InputStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<OutputStreamRefVector> get_output_streams_by_network(const ...)
```

Parameters network_name - [in] Network name of the requested output streams. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of vector *OutputStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual InputStreamRefVector get_input_streams() = 0
```

Returns All input streams.

```
virtual OutputStreamRefVector get_output_streams() = 0
```

Returns All output streams.

```
virtual Expected<LatencyMeasurementResult> get_latency_measurement(const std::string ...)
```

Parameters network_name - [in] Network name of the requested latency measurement. If not passed, all the networks in the network group will be addressed, and the resulted measurement is avarage latency of all networks.

Returns Upon success, returns *Expected* of *LatencyMeasurementResult* object containing the output latency result. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<OutputStreamWithParamsVector> get_output_streams_from_vstream_names(const ...)
```

Gets output streams and their vstream params from vstreams names.

Note: The output streams returned here are streams that corresponds to the names in outputs_params. If outputs_params contains a demux edge name, then all its predecessors names must be in outputs_params as well and the return value will contain the stream that corresponds to those edges.

Parameters outputs_params - [in] Map of output vstream name and params.

Returns Upon success, returns *Expected* of *OutputStreamWithParamsVector*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
Expected<std::unique_ptr<ActivatedNetworkGroup>> activate()
```

Activates hailo device inner-resources for inference.

Returns Upon success, returns *Expected* of a pointer to *ActivatedNetworkGroup* object. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<std::unique_ptr<ActivatedNetworkGroup>> activate(const ...)
```

Activates hailo device inner-resources for inference.

Parameters network_group_params - [in] Parameters for the activation.

Returns Upon success, returns *Expected* of a pointer to *ActivatedNetworkGroup* object. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual hailo_status wait_for_activation(const std::chrono::milliseconds &timeout) = 0
```

Block until network group is activated, or until timeout is passed.

Parameters *timeout* - **[in]** The timeout in milliseconds. If *timeout* is zero, the function returns immediately. If *timeout* is HAILO_INFINITE, the function returns only when the event is signaled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

```
virtual hailo_status shutdown() = 0
```

Shutdown the network group. Makes sure all ongoing async operations are canceled. All async callbacks of transfers that have not been completed will be called with status *HAILO_STREAM_ABORT*. Any resources attached to the network group may be released after function returns.

Note: Calling this function is optional, and it is used to shutdown network group while there is still ongoing inference.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

```
virtual Expected<std::map<std::string, hailo_vstream_params_t>> make_input_vstream_params(bool...)
```

Creates input virtual stream params.

Parameters

- *unused* - **[in]** Unused.
- *format_type* - **[in]** The default format type for all input virtual streams.
- *timeout_ms* - **[in]** The default timeout in milliseconds for all input virtual streams.
- *queue_size* - **[in]** The default queue size for all input virtual streams.
- *network_name* - **[in]** Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of a map of name to vstream params. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<std::map<std::string, hailo_vstream_params_t>> make_output_vstream_params(bool...)
```

Creates output virtual stream params.

Parameters

- *unused* - **[in]** Unused.
- *format_type* - **[in]** The default format type for all output virtual streams.
- *timeout_ms* - **[in]** The default timeout in milliseconds for all output virtual streams.
- *queue_size* - **[in]** The default queue size for all output virtual streams.
- *network_name* - **[in]** Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of a map of name to vstream params. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<std::vector<std::map<std::string, hailo_vstream_params_t>>> make_output_vstream_params_groups()
```

Creates output virtual stream params. The groups are split with respect to their low-level streams.

Parameters

- *unused* - **[in]** Unused.

- **format_type** - **[in]** The default format type for all output virtual streams.
- **timeout_ms** - **[in]** The default timeout in milliseconds for all output virtual streams.
- **queue_size** - **[in]** The default queue size for all output virtual streams.

Returns Upon success, returns *Expected* of a vector of maps, mapping name to vstream params, where each map represents a params group. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<std::vector<std::vector<std::string>> get_output_vstream_groups() = 0
```

Gets output virtual stream groups for given network_group. The groups are split with respect to their low-level streams.

Returns Upon success, returns *Expected* of a map of vstream name to group index. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<std::vector<hailo_network_info_t>> get_network_infos() const = 0
```

Gets all network infos of the configured network group.

Returns Upon success, returns *Expected* of a vector of all network infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<std::vector<hailo_stream_info_t>> get_all_stream_infos(const std::string ...)
```

Parameters **network_name** - **[in]** Network name of the requested stream infos. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of all stream infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<std::vector<hailo_vstream_info_t>> get_input_vstream_infos(const std::string ...)
```

Parameters **network_name** - **[in]** Network name of the requested virtual stream infos. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of all input vstreams infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<std::vector<hailo_vstream_info_t>> get_output_vstream_infos(const std::string ...)
```

Parameters **network_name** - **[in]** Network name of the requested virtual stream infos. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of all output vstreams infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<std::vector<hailo_vstream_info_t>> get_all_vstream_infos(const std::string ...)
```

Parameters **network_name** - **[in]** Network name of the requested virtual stream infos. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of all vstreams infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual bool is_scheduled() const = 0
```

Returns whether the network group is managed by the model scheduler.

```
virtual hailo_status set_scheduler_timeout(const std::chrono::milliseconds &timeout, const ...)
```

Sets the maximum time period that may pass before receiving run time from the scheduler. This will occur providing at least one send request has been sent, there is no minimum requirement for send requests, (e.g. threshold - see *set_scheduler_threshold()*).

Note: The new time period will be measured after the previous time the scheduler allocated run time to this network group.

Note: Using this function is only allowed when scheduling_algorithm is not [HAILO_SCHEDULING_ALGORITHM_NONE](#).

Note: The default timeout is 0ms.

Note: Currently, setting the timeout for a specific network is not supported.

Parameters

- **timeout** - **[in]** Timeout in milliseconds.
- **network_name** - **[in]** Network name for which to set the timeout. If not passed, the timeout will be set for all the networks in the network group.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

virtual [hailo_status](#) [set_scheduler_threshold](#)(uint32_t threshold, const std::string ...)

Sets the minimum number of send requests required before the network is considered ready to get run time from the scheduler.

Note: Using this function is only allowed when scheduling_algorithm is not [HAILO_SCHEDULING_ALGORITHM_NONE](#).

Note: The default threshold is 1.

Note: If at least one send request has been sent, but the threshold is not reached within a set time period (e.g. timeout - see [hailo_set_scheduler_timeout\(\)](#)), the scheduler will consider the network ready regardless.

Note: Currently, setting the threshold for a specific network is not supported.

Parameters

- **threshold** - **[in]** Threshold in number of frames.
- **network_name** - **[in]** Network name for which to set the threshold. If not passed, the threshold will be set for all the networks in the network group.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

virtual [hailo_status](#) [set_scheduler_priority](#)(uint8_t priority, const std::string &network_name = ...)

Sets the priority of the network. When the network group scheduler will choose the next network, networks with higher priority will be prioritized in the selection. Larger number represents higher priority

Note: Using this function is only allowed when scheduling_algorithm is not [HAILO_SCHEDULING_ALGORITHM_NONE](#).

Note: The default priority is HAILO_SCHEDULER_PRIORITY_NORMAL.

Note: Currently, setting the priority for a specific network is not supported.

Parameters

- **priority** - **[in]** Priority as a number between HAILO_SCHEDULER_PRIORITY_MIN - HAILO_SCHEDULER_PRIORITY_MAX.
- **network_name** - **[in]** Network name for which to set the Priority. If not passed, the priority will be set for all the networks in the network group.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

```
virtual bool is_multi_context() const = 0
```

Returns Is the network group multi-context or not.

```
virtual const ConfigureNetworkParams get_config_params() const = 0
```

Returns The configuration parameters this network group was initialized with.

15.5. Stream API Functions

```
class hailort :: InputStream
```

Input (host to device) stream representation

Public Types

```
using TransferDoneCallback = std::function<void(const CompletionInfo &completion_info)>
```

Async transfer complete callback prototype.

Public Functions

```
virtual hailo\_status set_timeout(std::chrono::milliseconds timeout) = 0
```

Set new timeout value to the input stream

Parameters **timeout** - **[in]** The new timeout value to be set in milliseconds.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

```
virtual std::chrono::milliseconds get_timeout() const = 0
```

Returns The input stream's timeout in milliseconds.

```
virtual hailo\_stream\_interface\_t get_interface() const = 0
```

Returns The input stream's interface.

```
virtual hailo\_status abort() = 0
```

Aborting the stream.

Note: This function is deprecated. One should use [ConfiguredNetworkGroup::shutdown\(\)](#)

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

```
virtual hailo_status clear_abort() = 0  
Clearing the aborted state of the stream.
```

Note: This function is deprecated. To reuse network after shutdown, reconfigure it.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

```
virtual hailo_status flush()  
Writes all pending data to the underlying stream.
```

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

```
EventPtr &get_network_group_activated_event()
```

Returns a pointer for network group activated event.

```
virtual bool is_scheduled() = 0
```

Returns whether the stream is managed by the model scheduler.

```
virtual hailo_status write(const MemoryView &buffer) = 0  
Writes the entire buffer to the stream without transformations.
```

Note: *buffer* is expected to be in the format dictated by this.stream_info.format

Note: *buffer.size()* is expected to be *get_frame_size()*.

Parameters *buffer* - **[in]** The buffer to be written.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

```
virtual hailo_status write(const void *buffer, size_t size) = 0  
Writes the entire buffer to the stream without transformations.
```

Note: *buffer* is expected to be in the format dictated by this.stream_info.format

Note: *size* is expected to be *get_frame_size()*.

Parameters

- *buffer* - **[in]** The buffer to be written.
- *size* - **[in]** The size of the buffer given.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

```
virtual hailo_status wait_for_async_ready(size_t transfer_size, std::chrono::milliseconds timeout)  
Waits until the stream is ready to launch a new write_async() operation. Each stream contains some limited sized queue for ongoing transfers. Calling get_async_max_queue_size() will return the queue size for current stream.
```

Parameters

- *transfer_size* - **[in]** Must be *get_frame_size()*.

- **timeout** - **[in]** Amount of time to wait until the stream is ready in milliseconds.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise:

- If *timeout* has passed and the stream is not ready, returns [HAILO_TIMEOUT](#).
- In any other error case, returns [hailo_status](#) error.

virtual [Expected](#)<size_t> [get_async_max_queue_size](#)() const

Returns the maximum amount of frames that can be simultaneously written to the stream (by [write_async\(\)](#) calls) before any one of the write operations is complete, as signified by *user_callback* being called.

Returns Upon success, returns [Expected](#) of the queue size. Otherwise, returns [Unexpected](#) of [hailo_status](#) error.

virtual [hailo_status](#) [write_async](#)(const MemoryView &buffer, const [TransferDoneCallback](#) ...)

Writes the contents of *buffer* to the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., [write_async\(\)](#) returns [HAILO_SUCCESS](#)), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., [write_async\(\)](#) returns a status other than [HAILO_SUCCESS](#)), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a [CompletionInfo](#) object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*).

Note: *user_callback* should run as quickly as possible.

Note: The buffer's format comes from the *format* field inside [get_info\(\)](#) and the shape comes from the *hw_shape* field inside [get_info\(\)](#).

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Note: Pre-mapping *buffer* to DMA via [Device::dma_map\(\)](#) may improve performance, if *buffer* is used for multiple async transfers.

Parameters

- **buffer** - **[in]** The buffer to be written. The buffer must be aligned to the system page size.
- **user_callback** - **[in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise:

- If the stream queue is full, returns [HAILO_QUEUE_IS_FULL](#). In this case please wait until *user_callback* is called on previous writes, or call [wait_for_async_ready\(\)](#). The size of the queue can be determined by calling [get_async_max_queue_size\(\)](#).
- In any other error case, returns a [hailo_status](#) error.

```
virtual hailo_status write_async(const void *buffer, size_t size, const TransferDoneCallback ...)
```

Writes the contents of *buffer* to the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., *write_async()* returns *HAILO_SUCCESS*), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., *write_async()* returns a status other than *HAILO_SUCCESS*), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a *CompletionInfo* object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*).

Note: *user_callback* should run as quickly as possible.

Note: The buffer's format comes from the *format* field inside *get_info()* and the shape comes from the *hw_shape* field inside *get_info()*.

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using *mmap* on Unix-like systems or *VirtualAlloc* on Windows.

Note: Pre-mapping *buffer* to DMA via *Device::dma_map()* may improve performance, if *buffer* is used for multiple async transfers.

Parameters

- **buffer - [in]** The buffer to be written. The buffer must be aligned to the system page size.
- **size - [in]** The size of the given buffer, expected to be *get_frame_size()*.
- **user_callback - [in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If the stream queue is full, returns *HAILO_QUEUE_IS_FULL*. In this case please wait until *user_callback* is called on previous writes, or call *wait_for_async_ready()*. The size of the queue can be determined by calling *get_async_max_queue_size()*.
- In any other error case, returns a *hailo_status* error.

```
virtual hailo_status write_async(int dmabuf_fd, size_t size, const TransferDoneCallback ...)
```

Writes the contents of the dmabuf associated with the fd *dmabuf_fd* to the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., *write_async()* returns *HAILO_SUCCESS*), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *dmabuf_fd*.
- If the function call fails (i.e., *write_async()* returns a status other than *HAILO_SUCCESS*), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *dmabuf_fd*.

- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a [CompletionInfo](#) object containing a fd representing the transferred buffer (*dmabuf_fd*) and the transfer status (*status*).

Note: *user_callback* should run as quickly as possible.

Note: The dmabuf fd must be a linux-system dmabuf fd - otherwise behavior will fail.

Note: This API of write_async is currently experimental.

Parameters

- *dmabuf_fd* - **[in]** The File descriptor to the dmabuf
- *size* - **[in]** The size of the given buffer, expected to be [get_frame_size\(\)](#).
- *user_callback* - **[in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise:

- If the stream queue is full, returns [HAILO_QUEUE_IS_FULL](#). In this case please wait until *user_callback* is called on previous writes, or call [wait_for_async_ready\(\)](#). The size of the queue can be determined by calling [get_async_max_queue_size\(\)](#).
- In any other error case, returns a [hailo_status](#) error.

inline virtual const [hailo_stream_info_t](#) &get_info() const

Returns A [hailo_stream_info_t](#) object containing the stream's info.

inline virtual const std::vector<[hailo_quant_info_t](#)> &get_quant_infos() const

Returns the quant_infos - quant info per feature.

inline virtual size_t get_frame_size() const

Returns the stream's hw frame size in bytes.

inline std::string name() const

Returns the stream's name.

virtual std::string to_string() const

Returns the stream's description containing it's name and index.

struct CompletionInfo

Context passed to the [TransferDoneCallback](#) after the async operation is done or has failed.

Public Members

`hailo_status status`

Status of the async transfer.

- `HAILO_SUCCESS` - When transfer is complete successfully.
- `HAILO_STREAM_ABORT` - The transfer was canceled (can happen after network deactivation).
- Any other `hailo_status` on unexpected errors.

class `hailort` :: `OutputStream`

Output (device to host) stream representation

Public Types

using `TransferDoneCallback` = std::function<void(const `CompletionInfo` &completion_info)>

Async transfer complete callback prototype.

Public Functions

`virtual hailo_status set_timeout(std::chrono::milliseconds timeout) = 0`

Set new timeout value to the output stream

Parameters `timeout` - [in] The new timeout value to be set in milliseconds.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`virtual std::chrono::milliseconds get_timeout() const = 0`

Returns returns the output stream's timeout in milliseconds.

`virtual hailo_stream_interface_t get_interface() const = 0`

Returns returns the output stream's interface.

`virtual hailo_status abort() = 0`

Aborting the stream.

Note: This function is deprecated. One should use `ConfiguredNetworkGroup::shutdown()`

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`virtual hailo_status clear_abort() = 0`

Clearing the abort flag of the stream.

Note: This function is deprecated. To reuse network after shutdown, reconfigure it.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`EventPtr &get_network_group_activated_event()`

Returns a pointer for network group activated event.

`virtual bool is_scheduled() = 0`

Returns whether the stream is managed by the model scheduler.

`inline virtual const hailo_stream_info_t &get_info() const`

Returns the stream's info.

```
inline virtual const std::vector<hailo_quant_info_t> &get_quant_infos() const
```

Returns the quant_infos - quant info per feature.

```
inline virtual size_t get_frame_size() const
```

Returns the stream's hw frame size.

```
inline std::string name() const
```

Returns the stream's name.

```
virtual std::string to_string() const
```

Returns the stream's description containing it's name and index.

```
uint32_t get_invalid_frames_count() const
```

Returns the number of invalid frames received by the stream.

```
virtual hailo_status read(MemoryView buffer) = 0
```

Reads the entire buffer from the stream without transformations

Note: Upon return, *buffer* is expected to be in the format dictated by this.get_info().format

Note: *size* is expected to be [get_frame_size\(\)](#).

Parameters *buffer* - **[in]** A buffer that receives the data read from the stream.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns an [hailo_status](#) error.

```
virtual hailo_status read(void *buffer, size_t size) = 0
```

Reads the entire buffer from the stream without transformations

Note: Upon return, *buffer* is expected to be in the format dictated by this.get_info().format

Note: *size* is expected to be [get_frame_size\(\)](#).

Parameters

- *buffer* - **[in]** A pointer to a buffer that receives the data read from the stream.
- *size* - **[in]** The size of the given buffer.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns an [hailo_status](#) error.

```
virtual hailo_status wait_for_async_ready(size_t transfer_size, std::chrono::milliseconds timeout)
```

Waits until the stream is ready to launch a new [read_async\(\)](#) operation. Each stream contains some limited sized queue for ongoing transfers. Calling [get_async_max_queue_size\(\)](#) will return the queue size for current stream.

Parameters

- *transfer_size* - **[in]** Must be [get_frame_size\(\)](#).
- *timeout* - **[in]** Amount of time to wait until the stream is ready in milliseconds.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise:

- If *timeout* has passed and the stream is not ready, returns [HAILO_TIMEOUT](#).

- In any other error case, returns *hailo_status* error.

virtual *Expected*<size_t> *get_async_max_queue_size()* const

Returns the maximum amount of frames that can be simultaneously read from the stream (by *read_async()* calls) before any one of the read operations is complete, as signified by *user_callback* being called.

Returns Upon success, returns *Expected* of the queue size. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *hailo_status* *read_async* (MemoryView *buffer*, const *TransferDoneCallback* &*user_callback*) = 0

Reads into *buffer* from the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., *read_async()* returns *HAILO_SUCCESS*), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., *read_async()* returns a status other than *HAILO_SUCCESS*), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a *CompletionInfo* object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*). If the operation has completed successfully, the contents of *buffer* will have been updated by the read operation.

Note: *user_callback* should execute as quickly as possible.

Note: The buffer's format is determined by the *format* field inside *get_info()*, and the shape is determined by the *hw_shape* field inside *get_info()*.

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using *mmap* on Unix-like systems or *VirtualAlloc* on Windows.

Note: Pre-mapping *buffer* to DMA via *Device::dma_map()* may improve performance, if *buffer* is used for multiple async transfers.

Parameters

- *buffer* - **[in]** The buffer to be read into. The buffer must be aligned to the system page size.
- *user_callback* - **[in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If the stream queue is full, returns *HAILO_QUEUE_IS_FULL*. In this case, please wait until *user_callback* is called on previous reads, or call *wait_for_async_ready()*. The size of the queue can be determined by calling *get_async_max_queue_size()*.
- In any other error case, returns a *hailo_status* error.

virtual *hailo_status* *read_async* (void **buffer*, size_t *size*, const *TransferDoneCallback* &*user_callback*) = 0

Reads into *buffer* from the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., `read_async()` returns `HAILO_SUCCESS`), the deferred operation has been initiated. Until `user_callback` is called, the user cannot change or delete `buffer`.
- If the function call fails (i.e., `read_async()` returns a status other than `HAILO_SUCCESS`), the deferred operation will not be initiated and `user_callback` will not be invoked. The user is free to change or delete `buffer`.
- `user_callback` is triggered upon successful completion or failure of the deferred operation. The callback receives a `CompletionInfo` object containing a pointer to the transferred buffer (`buffer_addr`) and the transfer status (`status`). If the operation has completed successfully, the contents of `buffer` will have been updated by the read operation.

Note: `user_callback` should execute as quickly as possible.

Note: The buffer's format is determined by the `format` field inside `get_info()`, and the shape is determined by the `hw_shape` field inside `get_info()`

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Note: Pre-mapping `buffer` to DMA via `Device::dma_map()` may improve performance, if `buffer` is used for multiple async transfers.

Parameters

- `buffer` - **[in]** The buffer to be read into. The buffer must be aligned to the system page size.
- `size` - **[in]** The size of the given buffer, expected to be `get_frame_size()`.
- `user_callback` - **[in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise:

- If the stream queue is full, returns `HAILO_QUEUE_IS_FULL`. In this case, please wait until `user_callback` is called on previous reads, or call `wait_for_async_ready()`. The size of the queue can be determined by calling `get_async_max_queue_size()`.
- In any other error case, returns a `hailo_status` error.

virtual `hailo_status read_async(int dmabuf_fd, size_t size, const TransferDoneCallback &user_callback) = 0`

Reads into dmabuf represented by fd `dmabuf_fd` from the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., `read_async()` returns `HAILO_SUCCESS`), the deferred operation has been initiated. Until `user_callback` is called, the user cannot change or delete `dmabuf_fd`.
- If the function call fails (i.e., `read_async()` returns a status other than `HAILO_SUCCESS`), the deferred operation will not be initiated and `user_callback` will not be invoked. The user is free to change or delete `dmabuf_fd`.
- `user_callback` is triggered upon successful completion or failure of the deferred operation. The callback receives a `CompletionInfo` object containing a fd representing the transferred buffer (`dmabuf_fd`)

and the transfer status (*status*). If the operation has completed successfully, the contents of the dmabuf will have been updated by the read operation.

Note: *user_callback* should execute as quickly as possible.

Note: The dmabuf fd must be a linux-system dmabuf fd - otherwise behavior will fail.

Note: This API of read_async is currently experimental.

Parameters

- `dmabuf_fd` - **[in]** The File descriptor to the dmabuf
- `size` - **[in]** The size of the given buffer, expected to be [get_frame_size\(\)](#).
- `user_callback` - **[in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise:

- If the stream queue is full, returns [HAILO_QUEUE_IS_FULL](#). In this case, please wait until *user_callback* is called on previous reads, or call [wait_for_async_ready\(\)](#). The size of the queue can be determined by calling [get_async_max_queue_size\(\)](#).
- In any other error case, returns a [hailo_status](#) error.

struct CompletionInfo

Context passed to the [TransferDoneCallback](#) after the async operation is done or has failed.

Public Members

[hailo_status](#) status

Status of the async transfer.

- [HAILO_SUCCESS](#) - When transfer is complete successfully.
- [HAILO_STREAM_ABORT](#) - The transfer was canceled (can happen after network deactivation).
- Any other [hailo_status](#) on unexpected errors.

15.6. Transformation API Functions

HAILORTAPI [hailo_status](#) transpose_buffer(const MemoryView src, const [hailo_3d_image_shape_t](#) &shape, ...)

Transposed *src* buffer (whose shape and format are given) to *dst* buffer (whose shape will be the same as *shape* but the height and width are replaced)

Note: assumes src and dst not overlaid

Parameters

- `src` - **[in]** A buffer containing the data to be transposed.
- `shape` - **[in]** The shape of *src*.
- `format` - **[in]** The format of *src*.
- `dst` - **[out]** The *dst* buffer to contain the transposed data.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
HAILORTAPI inline size_t get_transpose_buffer_size(const hailo_3d_image_shape_t &src_shape, ...)
```

Returns The size of transpose buffer by its src shape and dst format type.

```
HAILORTAPI hailo_status fuse_buffers(const std::vector<MemoryView> &buffers, const ...)
```

Fuse multiple defused NMS buffers referred by *buffers* to the buffer referred by *dst*. This function expects *buffers* to be ordered by their *class_group_index* (lowest to highest).

Parameters

- *buffers* - **[in]** A vector of buffers to be fused.
- *infos_of_buffers* - **[in]** A vector of `hailo_nms_info_t` containing the *buffers* information.
- *dst* - **[out]** A pointer to a buffer which will contain the fused buffer.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
class hailort::InputTransformContext
```

Object used for input stream transformation

Public Functions

```
hailo_status transform(const MemoryView src, MemoryView dst)
```

Transforms an input frame referred by *src* directly to the buffer referred by *dst*.

Parameters

- *src* - **[in]** A src buffer to be transformed.
- *dst* - **[out]** A dst buffer that receives the transformed data.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
size_t get_src_frame_size() const
```

Returns The size of the src frame on the host side in bytes.

```
size_t get_dst_frame_size() const
```

Returns The size of the dst frame on the hw side in bytes.

```
virtual std::string description() const
```

Returns A human-readable description of the transformation parameters.

Public Static Functions

```
static Expected<std::unique_ptr<InputTransformContext>> create(const hailo_3d_image_shape_t ...)
```

Creates input transform_context.

Parameters

- *src_image_shape* - **[in]** The shape of the src buffer to be transformed.
- *src_format* - **[in]** The format of the src buffer to be transformed.
- *dst_image_shape* - **[in]** The shape of the dst buffer that receives the transformed data.
- *dst_format* - **[in]** The format of the dst buffer that receives the transformed data.
- *dst_quant_infos* - **[in]** A vector of `hailo_quant_info_t` object containing quantization information per feature. Might also contain a vector with a single `hailo_quant_info_t` object.

Returns Upon success, returns *Expected* of a pointer to *InputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
static Expected<std::unique_ptr<InputTransformContext>> create(const hailo_3d_image_shape_t ...)
```

Creates input transform_context.

Note: This function is deprecated

Parameters

- *src_image_shape* - **[in]** The shape of the src buffer to be transformed.
- *src_format* - **[in]** The format of the src buffer to be transformed.
- *dst_image_shape* - **[in]** The shape of the dst buffer that receives the transformed data.
- *dst_format* - **[in]** The format of the dst buffer that receives the transformed data.
- *dst_quant_info* - **[in]** A *hailo_quant_info_t* object containing quantization information.

Returns Upon success, returns *Expected* of a pointer to *InputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
static Expected<std::unique_ptr<InputTransformContext>> create(const hailo_stream_info_t &stream_info, ...)
```

Creates input transform_context.

Parameters

- *stream_info* - **[in]** Creates transform_context that fits this stream info.
- *transform_params* - **[in]** A *hailo_transform_params_t* object containing user transformation parameters.

Returns Upon success, returns *Expected* of a pointer to *InputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
static Expected<std::unique_ptr<InputTransformContext>> create(const hailo_stream_info_t &stream_info, ...)
```

Creates input transform_context.

Parameters

- *stream_info* - **[in]** Creates transform_context that fits this stream info.
- *unused* - **[in]** Unused.
- *format_type* - **[in]** The type of the buffer sent to the transform_context.

Returns Upon success, returns *Expected* of a pointer to *InputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
static Expected<std::unique_ptr<InputTransformContext>> create(InputStream &input_stream, const ...)
```

Creates input transform_context by output stream

Parameters

- *input_stream* - **[in]** Creates transform_context that fits this input stream.
- *transform_params* - **[in]** A *hailo_transform_params_t* object containing user transformation parameters.

Returns Upon success, returns *Expected* of a pointer to *InputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
static Expected<bool> is_transformation_required(const hailo_3d_image_shape_t ...)
```

Check whether or not a transformation is needed.

Note: In case the function returns false, the src frame is ready to be sent to HW without any transformation.

Parameters

- `src_image_shape` - **[in]** The shape of the src buffer (host shape).
- `src_format` - **[in]** The format of the src buffer (host format).
- `dst_image_shape` - **[in]** The shape of the dst buffer (hw shape).
- `dst_format` - **[in]** The format of the dst buffer (hw format).
- `quant_infos` - **[in]** A vector of `hailo_quant_info_t` object containing quantization information per feature.

Returns Returns `Expected` of boolean, whether or not a transformation is needed.

```
static bool is_transformation_required(const hailo_3d_image_shape_t &src_image_shape, ...)
```

Check whether or not a transformation is needed.

Note: In case the function returns false, the src frame is ready to be sent to HW without any transformation.

Note: This function is deprecated.

Parameters

- `src_image_shape` - **[in]** The shape of the src buffer (host shape).
- `src_format` - **[in]** The format of the src buffer (host format).
- `dst_image_shape` - **[in]** The shape of the dst buffer (hw shape).
- `dst_format` - **[in]** The format of the dst buffer (hw format).
- `quant_info` - **[in]** A `hailo_quant_info_t` object containing quantization information.

Returns Returns whether or not a transformation is needed.

```
class hailort::OutputTransformContext
```

Object used for output stream transformation

Public Functions

```
virtual hailo_status transform(const MemoryView src, MemoryView dst) = 0
```

Transforms an output frame referred by `src` directly to the buffer referred by `dst`.

Parameters

- `src` - **[in]** A src buffer to be transformed.
- `dst` - **[out]** A dst buffer that receives the transformed data.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
size_t get_src_frame_size() const
```

Returns The size of the src frame on the hw side in bytes.

```
size_t get_dst_frame_size() const
```

Returns The size of the dst frame on the host side in bytes.

```
virtual std::string description() const = 0
```

Returns A human-readable description of the transformation parameters.

Public Static Functions

```
static Expected<std::unique_ptr<OutputTransformContext>> create(const hailo_3d_image_shape_t ...)
```

Creates output transform_context.

Parameters

- `src_image_shape` - **[in]** The shape of the src buffer to be transformed.
- `src_format` - **[in]** The format of the src buffer to be transformed.
- `dst_image_shape` - **[in]** The shape of the dst buffer that receives the transformed data.
- `dst_format` - **[in]** The format of the dst buffer that receives the transformed data.
- `dst_quant_infos` - **[in]** A vector of `hailo_quant_info_t` object containing quantization information per feature. Might also contain a vector with a single `hailo_quant_info_t` object.
- `nms_info` - **[in]** A `hailo_nms_info_t` object containing nms information.

Returns Upon success, returns `Expected` of a pointer to `OutputTransformContext`. Otherwise, returns `Unexpected` of `hailo_status` error.

```
static Expected<std::unique_ptr<OutputTransformContext>> create(const hailo_3d_image_shape_t ...)
```

Creates output transform_context.

Note: This function is deprecated.

Parameters

- `src_image_shape` - **[in]** The shape of the src buffer to be transformed.
- `src_format` - **[in]** The format of the src buffer to be transformed.
- `dst_image_shape` - **[in]** The shape of the dst buffer that receives the transformed data.
- `dst_format` - **[in]** The format of the dst buffer that receives the transformed data.
- `dst_quant_info` - **[in]** A `hailo_quant_info_t` object containing quantization information.
- `nms_info` - **[in]** A `hailo_nms_info_t` object containing nms information.

Returns Upon success, returns `Expected` of a pointer to `OutputTransformContext`. Otherwise, returns `Unexpected` of `hailo_status` error.

```
static Expected<std::unique_ptr<OutputTransformContext>> create(const hailo_stream_info_t ...)
```

Creates output transform_context.

Parameters

- `stream_info` - **[in]** Creates transform_context that fits this stream info.

- **transform_params** - **[in]** A *halo_transform_params_t* object containing user transformation parameters.

Returns Upon success, returns *Expected* of a pointer to *OutputTransformContext*. Otherwise, returns *Unexpected* of *halo_status* error.

```
static Expected<std::unique_ptr<OutputTransformContext>> create(const halo_stream_info_t ...)
```

Creates output transform_context with default transform parameters

Parameters

- **stream_info** - **[in]** Creates transform_context that fits this stream info.
- **unused** - **[in]** Unused.
- **format_type** - **[in]** The type of the buffer returned from the transform_context

Returns Upon success, returns *Expected* of a pointer to *OutputTransformContext*. Otherwise, returns *Unexpected* of *halo_status* error.

```
static Expected<std::unique_ptr<OutputTransformContext>> create(OutputStream &output_stream, const ...)
```

Creates output transform_context by output stream

Parameters

- **output_stream** - **[in]** Creates transform_context that fits this output stream.
- **transform_params** - **[in]** A *halo_transform_params_t* object containing user transformation parameters.

Returns Upon success, returns *Expected* of a pointer to *OutputTransformContext*. Otherwise, returns *Unexpected* of *halo_status* error.

```
static Expected<bool> is_transformation_required(const halo_3d_image_shape_t ...)
```

Check whether or not a transformation is needed.

Note: In case the function returns false, the src frame is already in the required format without any transformation.

Parameters

- **src_image_shape** - **[in]** The shape of the src buffer (hw shape).
- **src_format** - **[in]** The format of the src buffer (hw format).
- **dst_image_shape** - **[in]** The shape of the dst buffer (host shape).
- **dst_format** - **[in]** The format of the dst buffer (host format).
- **quant_infos** - **[in]** A vector of *halo_quant_info_t* object containing quantization information per feature.

Returns Returns *Expected* of boolean, whether or not a transformation is needed.

```
static bool is_transformation_required(const halo_3d_image_shape_t &src_image_shape, ...)
```

Check whether or not a transformation is needed.

Note: In case the function returns false, the src frame is already in the required format without any transformation.

Note: This function is deprecated.

Parameters

- `src_image_shape` - **[in]** The shape of the src buffer (hw shape).
- `src_format` - **[in]** The format of the src buffer (hw format).
- `dst_image_shape` - **[in]** The shape of the dst buffer (host shape).
- `dst_format` - **[in]** The format of the dst buffer (host format).
- `quant_info` - **[in]** A `hailo_quant_info_t` object containing quantization information.

Returns Returns whether or not a transformation is needed.

```
class hailort ::OutputDemuxer
```

Object used to demux muxed stream

Public Functions

```
virtual std::vector<hailo_stream_info_t> get_edges_stream_info() = 0
```

Returns The demuxer's edges information.

```
virtual hailo_status transform_demux(const MemoryView src, const std::map<std::string, ...>
```

Demultiplexing an output frame referred by `src` directly into the buffers referred by `dst_ptrs`.

Parameters

- `src` - **[in]** A buffer to be demultiplexed.
- `dst_ptrs` - **[out]** A mapping of output_name to its resulted demuxed data.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
virtual hailo_status transform_demux(const MemoryView src, std::vector<MemoryView> ...)
```

Demultiplexing an output frame referred by `src` directly into the buffers referred by `raw_buffers`.

Note: The order of `raw_buffers` should be the same as returned from the function '`get_edges_stream_info()`'.

Parameters

- `src` - **[in]** A buffer to be demultiplexed.
- `raw_buffers` - **[out]** A vector of buffers that receives the demultiplexed data read from the stream. The order of `raw_buffers` vector will remain as is.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

Public Static Functions

```
static Expected<std::unique_ptr<OutputDemuxer>> create(OutputStream &output_stream)
```

Creates an output demuxer for the given `output_stream`.

Parameters `output_stream` - **[in]** An `OutputStream` object to create demuxer from.

Returns Upon success, returns `Expected` of a pointer to `OutputDemuxer`. Otherwise, returns `Unexpected` of `hailo_status` error.

```
class hailort ::Quantization
```

Hailo device requires input data to be quantized/scaled before it is sent. Similarly, data outputted from the device needs to be 'de-quantized'/rescaled as well. When a neural network is compiled, each input/output layer in the neural network is assigned two floating point values that are parameters to an input/output transformation: `hailo_quant_info_t::qp_zp` (zero_point) and `hailo_quant_info_t::qp_scale` (fields of the struct `hailo_quant_info_t`). These values are stored in the `Hef`.

- Input transformation: input data is divided by `hailo_quant_info_t::qp_scale` and then `hailo_quant_info_t::qp_zp` is added to the result.
- Output transformation: `hailo_quant_info_t::qp_zp` is subtracted from output data and then the result is multiplied by `hailo_quant_info_t::qp_scale`.

Public Static Functions

```
template<typename T, typename Q>
```

```
static inline void dequantize_output_buffer(Q *src_ptr, T *dst_ptr, uint32_t ...)
```

De-quantize output buffer pointed by `src_ptr` from data type `Q` into the buffer pointed by `dst_ptr` of data type `T`.

Parameters

- `src_ptr` - **[in]** A pointer to the buffer containing the data that will be de-quantized.
- `dst_ptr` - **[out]** A pointer to the buffer that will contain the output de-quantized data.
- `buffer_elements_count` - **[in]** The number of elements in `src_ptr` and `dst_ptr` arrays.
- `quant_info` - **[in]** *Quantization* info.

```
template<typename T, typename Q>
```

```
static inline void dequantize_output_buffer_in_place(T *dst_ptr, uint32_t ...)
```

De-quantize in place the output buffer pointed by `dst_ptr` from data type `Q` to data type `T`.

Parameters

- `dst_ptr` - **[inout]** A pointer to the buffer to be de-quantized.
- `buffer_elements_count` - **[in]** The number of elements in `dst_ptr` array.
- `quant_info` - **[in]** *Quantization* info.

```
template<typename T, typename Q>
```

```
static inline void dequantize_output_buffer_in_place(T *dst_ptr, uint32_t offset, uint32_t ...)
```

De-quantize in place the output buffer pointed by `dst_ptr` starting from `offset` from data type `Q` to data type `T`.

Parameters

- `dst_ptr` - **[inout]** A pointer to the buffer to be de-quantized.
- `offset` - **[in]** The offset in `dst_ptr` array to start from.
- `buffer_elements_count` - **[in]** The number of elements in `dst_ptr` array.
- `qp_zp` - **[in]** *Quantization* zero point.
- `qp_scale` - **[in]** *Quantization* scale.

```
template<typename T, typename Q>
```

```
static inline void quantize_input_buffer(T *src_ptr, Q *dst_ptr, uint32_t buffer_elements_count, ...)
```

Quantize input buffer pointed by `src_ptr` of data type `T`, into the buffer pointed by `dst_ptr` of data type `Q`.

Parameters

- `src_ptr` - **[in]** A pointer to the buffer containing the data that will be quantized.
- `dst_ptr` - **[out]** A pointer to the buffer that will contain the output quantized data.
- `buffer_elements_count` - **[in]** The number of elements in `src_ptr` and `dst_ptr` arrays.
- `quant_info` - **[in]** *Quantization* info.

```
template<typename T, typename Q>
```

```
static inline void dequantize_output_buffer_nms (Q *src_ptr, T *dst_ptr, uint32_t ...)
```

De-quantize output NMS buffer pointed by *src_ptr* of data type *Q*, into the buffer pointed by *dst_ptr* of data type *T*.

Parameters

- *src_ptr* - **[in]** A pointer to the buffer containing the data that will be de-quantized.
- *dst_ptr* - **[out]** A pointer to the buffer that will contain the output de-quantized data.
- *buffer_elements_count* - **[in]** The number of elements in *src_ptr* and *dst_ptr* arrays.
- *quant_info* - **[in]** *Quantization* info.
- *number_of_classes* - **[in]** Amount of NMS classes.

```
static inline bool is_identity_qp (const halo_quant_info_t &quant_info)
```

Indicates whether the *quant_info* contains the identity scale. If true there is no need to fix the data's scale.

```
static inline bool is_identity_qp (float32_t qp_zp, float32_t qp_scale)
```

Indicates whether the *qp_zp* and *qp_scale* is the identity scale. If true there is no need to fix the data's scale.

```
template<typename T, typename Q>
```

```
static inline T dequantize_output (Q number, halo_quant_info_t quant_info)
```

De-quantize *number* from data type *Q* to data type *T* and fix it's scale according to *quant_info*.

Parameters

- *number* - **[in]** The value to be de-quantized.
- *quant_info* - **[in]** *Quantization* info.

Returns Returns the dequantized value of *number*.

```
template<typename T, typename Q>
```

```
static inline T dequantize_output (Q number, float32_t qp_zp, float32_t qp_scale)
```

De-quantize *number* from data type *Q* to data type *T* and fix it's scale according to *qp_zp* and *qp_scale*.

Parameters

- *number* - **[in]** The value to be de-quantized.
- *qp_zp* - **[in]** *Quantization* zero point.
- *qp_scale* - **[in]** *Quantization* scale.

Returns Returns the dequantized value of *number*.

```
static inline bool is_qp_valid (const halo_quant_info_t &quant_info)
```

Returns whether or not qp is valid

Note: QP will be invalid in case HEF file was compiled with multiple QPs, and then the user will try working with API for single QP. For example - if HEF was compiled with multiple QPs and then the user calls *halo_get_input_stream_info*, The *halo_quant_info_t* object of the *halo_stream_info_t* object will be invalid.

Parameters *quant_info* - **[in]** A *halo_quant_info_t* object.

Returns a bool, Indicates whether or not qp is valid.

15.7. InferModel API

```
class hailort::InferModel
```

Contains all of the necessary information for configuring the network for inference. This class is used to set up the model for inference and includes methods for setting and getting the model's parameters. By calling the `configure` function, the user can create a `ConfiguredInferModel` object, which is used to run inference.

Public Functions

```
virtual const Hef &hef( ) const = 0
```

Returns A constant reference to the `Hef` object associated with this `InferModel`.

```
virtual void set_batch_size(uint16_t batch_size) = 0
```

Sets the batch size of the `InferModel`. This parameter determines the number of frames that be sent for inference in a single batch. If a scheduler is enabled, this parameter determines the 'burst size' - the max number of frames after which the scheduler will attempt to switch to another model. If scheduler is disabled, the number of frames for inference should be a multiplication of batch_size (unless model is in single context).

Note: The default value is `HAILO_DEFAULT_BATCH_SIZE` - which means the batch is determined by HailoRT automatically.

Parameters `batch_size` - **[in]** The new batch size to be set.

```
virtual void set_power_mode(hailo\_power\_mode\_t power_mode) = 0
```

Sets the power mode of the `InferModel`. See `hailo_power_mode_t` for more information.

Parameters `power_mode` - **[in]** The new power mode to be set.

```
virtual void set_hw_latency_measurement_flags(hailo\_latency\_measurement\_flags\_t ...)
```

Sets the latency measurement flags of the `InferModel`. see `hailo_latency_measurement_flags_t` for more information.

Parameters `latency` - **[in]** The new latency measurement flags to be set.

```
virtual Expected<ConfiguredInferModel> configure( ) = 0
```

Configures the `InferModel` object. Also checks the validity of the configuration's formats.

Returns Upon success, returns `Expected` of `ConfiguredInferModel`, which can be used to perform an asynchronous inference. Otherwise, returns `Unexpected` of `hailo_status` error.

```
virtual Expected<InferStream> input( ) = 0
```

Returns the single input's `InferStream` object.

Note: If `InferModel` has multiple inputs, will return `HAILO_INVALID_OPERATION`. In that case - use `input(const std::string &name)` instead.

Returns Upon success, returns `Expected` of the single input's `InferStream` object. Otherwise, returns `Unexpected` of `hailo_status` error.

```
virtual Expected<InferStream> output( ) = 0
```

Returns the single output's `InferStream` object.

Note: If `InferModel` has multiple outputs, will return `HAILO_INVALID_OPERATION`. In that case - use `output(const std::string &name)` instead.

Returns Upon success, returns *Expected* of the single output's *InferStream* object. Otherwise, returns *Unexpected* of *hailo_status* error.

```
virtual Expected<InferStream> input(const std::string &name) = 0
```

Gets an input's *InferStream* object.

Parameters name – [in] The name of the input edge.

Returns Upon success, returns *Expected* of the relevant *InferStream* object. Otherwise, returns a *hailo_status* error.

```
virtual Expected<InferStream> output(const std::string &name) = 0
```

Gets an output's *InferStream* object.

Parameters name – [in] The name of the output edge.

Returns Upon success, returns *Expected* of the relevant *InferStream* object. Otherwise, returns a *hailo_status* error.

```
virtual const std::vector<InferStream> &inputs() const = 0
```

Returns A constant reference to the vector of input *InferStream* objects, each representing an input edge.

```
virtual const std::vector<InferStream> &outputs() const = 0
```

Returns A constant reference to the vector of output *InferStream* objects, each representing an output edge.

```
virtual const std::vector<std::string> &get_input_names() const = 0
```

Returns A constant reference to a vector of strings, each representing the name of an input stream.

```
virtual const std::vector<std::string> &get_output_names() const = 0
```

Returns A constant reference to a vector of strings, each representing the name of an output stream.

class InferStream

Represents the parameters of a stream. In default, the stream's parameters are set to the default values of the model. The user can change the stream's parameters by calling the *set_* functions.

Public Functions

```
const std::string name() const
```

Returns The name of the stream.

```
hailo_3d_image_shape_t shape() const
```

Returns The shape of the image that the stream will use for inference.

```
hailo_format_t format() const
```

Returns The format that the stream will use for inference.

```
size_t get_frame_size() const
```

Returns The size in bytes of a frame that the stream will use for inference.

```
Expected<hailo_nms_shape_t> get_nms_shape() const
```

Note: In case NMS is disabled, returns an unexpected of *HALO_INVALID_OPERATION*.

Returns upon success, an *Expected* of *hailo_nms_shape_t*, the NMS shape for the stream. Otherwise, returns *Unexpected* of *hailo_status* error.

```
void set_format_type(hailo_format_type_t type)
```

Sets the format type of the stream. This method is used to specify the format type that the stream will use for inference.

Parameters *type* - [in] The format type to be set for the stream. This should be a value of the *hailo_format_type_t* enum.

```
void set_format_order(hailo_format_order_t order)
```

Sets the format order of the stream. This method is used to specify the format order that the stream will use for inference.

Parameters *order* - [in] The format order to be set for the stream. This should be a value of the *hailo_format_order_t* enum.

```
std::vector<hailo_quant_info_t> get_quant_infos( ) const
```

Retrieves the quantization information for all layers in the model.

Returns A vector of *hailo_quant_info_t* structures, each representing the quantization information for a layer in the model.

```
bool is_nms( ) const
```

Checks if Non-Maximum Suppression (NMS) is enabled for the model.

Returns True if NMS is enabled, false otherwise.

```
void set_nms_score_threshold(float32_t threshold)
```

Set NMS score threshold, used for filtering out candidates. Any box with score<TH is suppressed.

Parameters *threshold* - [in] NMS score threshold to set.

```
void set_nms_iou_threshold(float32_t threshold)
```

Set NMS intersection over union overlap Threshold, used in the NMS iterative elimination process where potential duplicates of detected items are suppressed.

Parameters *threshold* - [in] NMS IoU threshold to set.

```
void set_nms_max_proposals_per_class(uint32_t max_proposals_per_class)
```

Set a limit for the maximum number of boxes per class.

Parameters *max_proposals_per_class* - [in] NMS max proposals per class to set.

```
void set_nms_max_proposals_total(uint32_t max_proposals_total)
```

Set a limit for the maximum number of boxes total (all classes).

Parameters *max_proposals_total* - [in] NMS max proposals total to set.

```
void set_nms_max_accumulated_mask_size(uint32_t max_accumulated_mask_size)
```

Set maximum accumulated mask size for all the detections in a frame.

Note: : Used in order to change the output buffer frame size in cases where the output buffer is too small for all the segmentation detections.

Parameters *max_accumulated_mask_size* - [in] NMS max accumulated mask size.

```
class hailort :: ConfiguredInferModel
```

Configured infer_model that can be used to perform an asynchronous inference

Public Functions

Expected<Bindings> `create_bindings()`

Creates a *Bindings* object.

Returns Upon success, returns *Expected* of *Bindings*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<Bindings> `create_bindings(const std::map<std::string, MemoryView> &buffers)`

Creates a *Bindings* object.

Parameters `buffers` - **[in]** map of input and output names and buffers.

Returns Upon success, returns *Expected* of *Bindings*. Otherwise, returns *Unexpected* of *hailo_status* error.

hailo_status `wait_for_async_ready(std::chrono::milliseconds timeout, uint32_t frames_count = 1)`

The readiness of the model to launch is determined by the ability to push buffers to the asynchronous inference pipeline. If the model is ready, the method will return immediately. If it's not ready initially, the method will wait for the model to become ready.

Note: Calling this function with frames_count greater than *get_async_queue_size()* will timeout.

Parameters

- `timeout` - **[in]** Amount of time to wait until the model is ready in milliseconds.
- `frames_count` - **[in]** The count of buffers you intent to infer in the next request. Useful for batch inference.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If `timeout` has passed and the model is not ready, returns *HAILO_TIMEOUT*.
- In any other error case, returns *hailo_status* error.

hailo_status `activate()`

Activates hailo device inner-resources for inference.

Note: Calling this function is invalid in case scheduler is enabled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns *hailo_status* error.

hailo_status `deactivate()`

Deactivates hailo device inner-resources for inference.

Note: Calling this function is invalid in case scheduler is enabled.

Returns Returns *HAILO_SUCCESS*.

hailo_status `run(const Bindings &bindings, std::chrono::milliseconds timeout)`

Launches a synchronous inference operation with the provided bindings.

Parameters

- `bindings` - **[in]** The bindings for the inputs and outputs of the model.
- `timeout` - **[in]** The maximum amount of time to wait for the inference operation to complete.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns `Unexpected` of `hailo_status` error.

`Expected<AsyncInferJob> run_async(const Bindings &bindings, std::function<void(const ...)`

Launches an asynchronous inference operation with the provided bindings. The completion of the operation is notified through the provided callback function.

Note: `callback` should execute as quickly as possible.

Note: The bindings' buffers should be kept intact until the async job is completed.

Note: To ensure the inference pipeline can handle new buffers, it is recommended to first call `wait_for_async_ready`

Parameters

- `bindings` - **[in]** The bindings for the inputs and outputs of the model.
- `callback` - **[in]** The function to be called upon completion of the asynchronous inference operation.

Returns Upon success, returns an instance of `Expected<AsyncInferJob>` representing the launched job. Otherwise, returns `Unexpected` of `hailo_status` error, and the interface shuts down completely.

`Expected<AsyncInferJob> run_async(const std::vector<Bindings> &bindings, std::function<void(const ...)`

Launches an asynchronous inference operation with the provided bindings. The completion of the operation is notified through the provided callback function. Overload for multiple-bindings inference (useful for batch inference).

Note: `callback` should execute as quickly as possible.

Note: The bindings' buffers should be kept intact until the async job is completed.

Note: To ensure the inference pipeline can handle new buffers, it is recommended to first call `wait_for_async_ready`

Parameters

- `bindings` - **[in]** The bindings for the inputs and outputs of the model.
- `callback` - **[in]** The function to be called upon completion of the asynchronous inference operation.

Returns Upon success, returns an instance of `Expected<AsyncInferJob>` representing the launched job. Otherwise, returns `Unexpected` of `hailo_status` error, and the interface shuts down completely.

`Expected<LatencyMeasurementResult> get_hw_latency_measurement()`

Returns Upon success, returns `Expected` of `LatencyMeasurementResult` object containing the output latency result. Otherwise, returns `Unexpected` of `hailo_status` error.

hailo_status `set_scheduler_timeout(const std::chrono::milliseconds &timeout)`

Sets the maximum time period that may pass before receiving run time from the scheduler. This will occur providing at least one send request has been sent, there is no minimum requirement for send requests, (e.g. threshold - see [set_scheduler_threshold\(\)](#)).

Note: The new time period will be measured after the previous time the scheduler allocated run time to this network group.

Note: Using this function is only allowed when `scheduling_algorithm` is not `HAILO_SCHEDULING_ALGORITHM_NONE`.

Note: The default timeout is 0ms.

Parameters `timeout` - **[in]** Timeout in milliseconds.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a *hailo_status* error.

hailo_status `set_scheduler_threshold(uint32_t threshold)`

Sets the minimum number of send requests required before the network is considered ready to get run time from the scheduler.

Note: Using this function is only allowed when `scheduling_algorithm` is not `HAILO_SCHEDULING_ALGORITHM_NONE`.

Note: The default threshold is 1.

Note: If at least one send request has been sent, but the threshold is not reached within a set time period (e.g. timeout - see [hailo_set_scheduler_timeout\(\)](#)), the scheduler will consider the network ready regardless.

Parameters `threshold` - **[in]** Threshold in number of frames.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a *hailo_status* error.

hailo_status `set_scheduler_priority(uint8_t priority)`

Sets the priority of the network. When the network group scheduler will choose the next network, networks with higher priority will be prioritized in the selection. Larger number represents higher priority

Note: Using this function is only allowed when `scheduling_algorithm` is not `HAILO_SCHEDULING_ALGORITHM_NONE`.

Note: The default priority is `HAILO_SCHEDULER_PRIORITY_NORMAL`.

Parameters `priority` - **[in]** Priority as a number between `HAILO_SCHEDULER_PRIORITY_MIN` - `HAILO_SCHEDULER_PRIORITY_MAX`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a *hailo_status* error.

Expected<size_t> `get_async_queue_size()` const

Returns Upon success, returns *Expected* of a the number of inferences that can be queued simultaneously for execution. Otherwise, returns *Unexpected* of `hailo_status` error.

`hailo_status shutdown()`

Shuts the inference down. After calling this method, the model is no longer usable.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error

class `Bindings`

Represents an asynchronous infer request - holds the input and output buffers of the request

Public Functions

Expected<`InferStream`> `input()`

Returns the single input's `InferStream` object.

Note: If `Bindings` has multiple inputs, will return `HAILO_INVALID_OPERATION`. In that case - use `input(const std::string &name)` instead.

Returns Upon success, returns *Expected* of the single input's `InferStream` object. Otherwise, returns *Unexpected* of `hailo_status` error.

Expected<`InferStream`> `output()`

Returns the single output's `InferStream` object.

Note: If `Bindings` has multiple outputs, will return `HAILO_INVALID_OPERATION`. In that case - use `output(const std::string &name)` instead.

Returns Upon success, returns *Expected* of the single output's `InferStream` object. Otherwise, returns *Unexpected* of `hailo_status` error.

Expected<`InferStream`> `input(const std::string &name)`

Gets an input's `InferStream` object.

Parameters name - [in] The name of the input edge.

Returns Upon success, returns *Expected* of the relevant `InferStream` object. Otherwise, returns a `hailo_status` error.

Expected<`InferStream`> `output(const std::string &name)`

Gets an output's `InferStream` object.

Parameters name - [in] The name of the output edge.

Returns Upon success, returns *Expected* of the relevant `InferStream` object. Otherwise, returns a `hailo_status` error.

Expected<`InferStream`> `input()` const

Returns the single input's `InferStream` object, as read only.

Note: If `Bindings` has multiple inputs, will return `HAILO_INVALID_OPERATION`. In that case - use `input(const std::string &name)` instead.

Returns Upon success, returns *Expected* of the single input's `InferStream` object. Otherwise, returns *Unexpected* of `hailo_status` error.

Expected<`InferStream`> `output()` const

Returns the single output's `InferStream` object, as read only.

Note: If *Bindings* has multiple outputs, will return *HAILO_INVALID_OPERATION*. In that case - use *output(const std::string &name)* instead.

Returns Upon success, returns *Expected* of the single output's *InferStream* object. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*InferStream*> *input* (const std::string &name) const

Gets an input's *InferStream* object, as read only.

Parameters name - **[in]** The name of the input edge.

Returns Upon success, returns *Expected* of the relevant *InferStream* object. Otherwise, returns a *hailo_status* error.

Expected<*InferStream*> *output* (const std::string &name) const

Gets an output's *InferStream* object, as read only.

Parameters name - **[in]** The name of the output edge.

Returns Upon success, returns *Expected* of the relevant *InferStream* object. Otherwise, returns a *hailo_status* error.

class *InferStream*

Holds the input and output buffers of the *Bindings* infer request

Public Functions

hailo_status *set_buffer* (MemoryView view)

Sets the edge's buffer to a new one, of type *MemoryView*.

For best performance and to avoid memory copies, buffers should be aligned to the system *PAGE_SIZE*.

On output streams, the actual buffer allocated size must be aligned to *PAGE_SIZE* as well - otherwise some memory corruption might occur at the end of the last page. For example, if the buffer size is 4000 bytes, the actual buffer size should be at least 4096 bytes. To fill all requirements, it is recommended to allocate the buffer with standard page allocation function provided by the os (mmap on linux, VirtualAlloc in windows).

Parameters view - **[in]** The new buffer to be set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<*MemoryView*> *get_buffer* () const

Note: If buffer type is not *MemoryView*, will return *HAILO_INVALID_OPERATION*.

Returns Upon success, returns *Expected* of the *MemoryView* buffer of the edge. Otherwise, returns *Unexpected* of *hailo_status* error.

hailo_status *set_pix_buffer* (const *hailo_pix_buffer_t* &pix_buffer)

Sets the edge's buffer to a new one, of type *hailo_pix_buffer_t*.

Each plane in the *hailo_pix_buffer_t* must meet the requirements listed in *set_buffer*.

Note: Supported only for inputs.

Note: Currently only support *memory_type* field of buffer to be *HAILO_PIX_BUFFER_MEMORY_TYPE_USERPTR*.

Parameters pix_buffer - **[in]** The new buffer to be set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<hailo_pix_buffer_t> `get_pix_buffer() const`

Note: If buffer type is not *hailo_pix_buffer_t*, will return *HAILO_INVALID_OPERATION*.

Returns Upon success, returns *Expected* of the *hailo_pix_buffer_t* buffer of the edge. Otherwise, returns *Unexpected* of *hailo_status* error.

hailo_status `set_dma_buffer(hailo_dma_buffer_t dma_buffer)`

Sets the edge's buffer from a DMA buffer.

Note: Supported on Linux only.

Parameters `dma_buffer - [in]` The new buffer to be set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<hailo_dma_buffer_t> `get_dma_buffer() const`

Note: If buffer type is not *hailo_dma_buffer_t*, will return *HAILO_INVALID_OPERATION*.

Note: Supported on Linux only.

Returns Upon success, returns *Expected* of the *hailo_dma_buffer_t* buffer of the edge. Otherwise, returns *Unexpected* of *hailo_status* error.

class *hailort* :: AsyncInferJob

Asynchronous inference job representation is used to manage and control an inference job that is running asynchronously.

Public Functions

hailo_status `wait(std::chrono::milliseconds timeout)`

Waits for the asynchronous inference job to finish.

Parameters `timeout - [in]` The maximum time to wait.

Returns A *hailo_status* indicating the status of the operation. If the job finishes successfully within the timeout, *HAILO_SUCCESS* is returned. Otherwise, returns a *hailo_status* error

`void detach()`

Detaches the job. Without detaching, the job's destructor will block until the job finishes.

struct *hailort* :: AsyncInferCompletionInfo

Context passed to the callback function after the asynchronous inference operation was completed or has failed.

Public Functions

inline *AsyncInferCompletionInfo*(*hailo_status* _status)

Constructor for *AsyncInferCompletionInfo*.

Parameters `_status - [in]` The status of the inference operation.

Public Members

`hailo_status` `status`

Status of the asynchronous inference operation.

- `HAILO_SUCCESS` - When the inference operation is complete successfully.
- Any other `hailo_status` on unexpected errors.

15.8. Virtual Stream API Functions

```
class hailort :: InputVStream
```

Public Functions

`hailo_status` `write`(const MemoryView &buffer)

Writes `buffer` to hailo device.

Parameters `buffer` - **[in]** The buffer containing the data to be sent to device. The buffer's format can be obtained by `get_user_buffer_format()`, and the buffer's shape can be obtained by calling `get_info().shape`.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `write`(const `hailo_pix_buffer_t` &buffer)

Writes `buffer` to hailo device.

Note: Currently only support `memory_type` field of buffer to be `HAILO_PIX_BUFFER_MEMORY_TYPE_USERPTR`.

Parameters `buffer` - **[in]** The buffer containing pointers to the planes where the data to be sent to the device is stored.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `flush`()

Flushes the vstream pipeline buffers. This will block until the vstream pipeline is clear.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `abort`()

Aborts vstream until its resumed.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status` `resume`()

Resumes vstream after it was aborted.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`size_t` `get_frame_size`() const

Note: The size could be affected by the format type - using `UINT16`, or by the data having not yet been quantized

Returns the size of a virtual stream's frame on the host side in bytes.

```
const hailo_vstream_info_t &get_info() const  
  
    Returns hailo_vstream_info_t object containing the vstream info.  
  
const std::vector<hailo_quant_info_t> &get_quant_infos() const  
  
    Returns the stream's vector of quantization infos.  
  
const hailo_format_t &get_user_buffer_format() const  
  
    Returns hailo_format_t object containing the user buffer format.  
  
std::string name() const  
  
    Returns the virtual stream's name.  
  
std::string network_name() const  
  
    Returns the virtual stream's network name.  
  
const std::map<std::string, AccumulatorPtr> &get_fps_accumulators() const  
  
Gets a reference to a map between pipeline element names to their respective fps accumulators. These accumulators measure the net throughput of each pipeline element. This means that the effects of queuing in the vstream pipeline (between elements) are not accounted for by these accumulators.
```

Note: FPS accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_FPS* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective fps accumulators.

```
const std::map<std::string, AccumulatorPtr> &get_latency_accumulators() const  
  
Gets a reference to a map between pipeline element names to their respective latency accumulators. These accumulators measure the net latency of each pipeline element. This means that the effects of queuing in the vstream pipeline (between elements) are not accounted for by these accumulators.
```

Note: Latency accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_LATENCY* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective latency accumulators.

```
const std::map<std::string, std::vector<AccumulatorPtr>> &get_queue_size_accumulators() const  
  
Gets a reference to a map between pipeline element names to their respective queue size accumulators. These accumulators measure the number of free buffers in the queue, right before a buffer is removed from the queue to be used.
```

Note: Queue size accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_QUEUE_SIZE* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective queue size accumulators.

```
AccumulatorPtr get_pipeline_latency_accumulator() const
```

Gets a shared_ptr to the vstream's latency accumulator. This accumulator measures the time it takes for a frame to pass through an entire vstream pipeline. Specifically:

- For *InputVStreams*: The time it takes a frame from the call to *InputVStream::write*, until the frame is written to the HW.
- For *OutputVStreams*: The time it takes a frame from being read from the HW, until it's returned to the user via *OutputVStream::read*.

Note: A pipeline-wide latency accumulator is created for the vstream, if the vstream is created with the flag *HALO_VSTREAM_STATS_MEASURE_LATENCY* set under the *vstream_stats_flags* field of *hailo_vstream_params_t*.

Returns A shared pointer to the vstream's latency accumulator.

```
const std::vector<std::shared_ptr<PipelineElement>> &get_pipeline() const
```

Returns A const reference to the *PipelineElements* of which this vstream is comprised of.

Public Static Functions

```
static halo_status clear(std::vector<InputVStream> &vstreams)
```

Clears the vstreams' pipeline buffers.

Parameters *vstreams* - [in] The vstreams to be cleared.

Returns Upon success, returns *HALO_SUCCESS*. Otherwise, returns a *halo_status* error.

```
static halo_status clear(std::vector<std::reference_wrapper<InputVStream>> &vstreams)
```

Clears the vstreams' pipeline buffers.

Parameters *vstreams* - [in] The vstreams to be cleared.

Returns Upon success, returns *HALO_SUCCESS*. Otherwise, returns a *halo_status* error.

```
class hailort::OutputVStream
```

Public Functions

```
halo_status read(MemoryView buffer)
```

Reads data from hailo device into *buffer*.

Parameters *buffer* - [in] The buffer to read data into. The buffer's format can be obtained by *get_user_buffer_format()*, and the buffer's shape can be obtained by calling *get_info().shape*.

Returns Upon success, returns *HALO_SUCCESS*. Otherwise, returns a *halo_status* error.

```
halo_status abort()
```

Aborts vstream until its resumed.

Returns Upon success, returns *HALO_SUCCESS*. Otherwise, returns a *halo_status* error.

```
halo_status resume()
```

Resumes vstream after it was aborted.

Returns Upon success, returns *HALO_SUCCESS*. Otherwise, returns a *halo_status* error.

```
size_t get_frame_size() const
```

Note: The size could be affected by the format type - using *UINT16*, or by the data having not yet been quantized

Returns the size of a virtual stream's frame on the host side in bytes.

```
const hailo_vstream_info_t &get_info() const
```

Returns *hailo_vstream_info_t* object containing the vstream info.

```
const std::vector<hailo_quant_info_t> &get_quant_infos() const
```

Returns the stream's vector of quantization infos.

```
const hailo_format_t &get_user_buffer_format() const
```

Returns *hailo_format_t* object containing the user buffer format.

```
std::string name() const
```

Returns the virtual stream's name.

```
std::string network_name() const
```

Returns the virtual stream's network name.

```
const std::map<std::string, AccumulatorPtr> &get_fps_accumulators() const
```

Gets a reference to a map between pipeline element names to their respective fps accumulators. These accumulators measure the net throughput of each pipeline element. This means that the effects of queuing in the vstream pipeline (between elements) are not accounted for by these accumulators.

Note: FPS accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_FPS* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective fps accumulators.

```
const std::map<std::string, AccumulatorPtr> &get_latency_accumulators() const
```

Gets a reference to a map between pipeline element names to their respective latency accumulators. These accumulators measure the net latency of each pipeline element. This means that the effects of queuing in the vstream pipeline (between elements) are not accounted for by these accumulators.

Note: Latency accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_LATENCY* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective latency accumulators.

```
const std::map<std::string, std::vector<AccumulatorPtr>> &get_queue_size_accumulators() const
```

Gets a reference to a map between pipeline element names to their respective queue size accumulators. These accumulators measure the number of buffers in the queue, waiting to be processed downstream. The measurements take place right before we try to enqueue the next buffer.

Note: Queue size accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_QUEUE_SIZE* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective queue size accumulators.

```
AccumulatorPtr get_pipeline_latency_accumulator() const
```

Gets a shared_ptr to the vstream's latency accumulator. This accumulator measures the time it takes for a frame to pass through an entire vstream pipeline. Specifically:

- For *InputVStreams*: The time it takes a frame from the call to *InputVStream::write*, until the frame is written to the HW.
- For *OutputVStreams*: The time it takes a frame from being read from the HW, until it's returned to the user via *OutputVStream::read*.

Note: A pipeline-wide latency accumulator is created for the vstream, if the vstream is created with the flag *HAIGO_VSTREAM_STATS_MEASURE_LATENCY* set under the *vstream_stats_flags* field of *hailo_vstream_params_t*.

Returns A shared pointer to the vstream's latency accumulator.

```
const std::vector<std::shared_ptr<PipelineElement>> &get_pipeline() const
```

Returns A const reference to the *PipelineElements* of which this vstream is comprised of.

```
halilo_status set_nms_score_threshold(float32_t threshold)
```

Set NMS score threshold, used for filtering out candidates. Any box with score<TH is suppressed.

Note: This function will fail in cases where the output vstream has no NMS operations on the CPU.

Parameters *threshold* - [in] NMS score threshold to set.

Returns Upon success, returns *HAIGO_SUCCESS*. Otherwise, returns a *halilo_status* error.

```
halilo_status set_nms_iou_threshold(float32_t threshold)
```

Set NMS intersection over union overlap Threshold, used in the NMS iterative elimination process where potential duplicates of detected items are suppressed.

Note: This function will fail in cases where the output vstream has no NMS operations on the CPU.

Parameters *threshold* - [in] NMS IoU threshold to set.

Returns Upon success, returns *HAIGO_SUCCESS*. Otherwise, returns a *halilo_status* error.

```
halilo_status set_nms_max_proposals_per_class(uint32_t max_proposals_per_class)
```

Set a limit for the maximum number of boxes per class.

Note: This function must be called before starting inference! This function will fail in cases where the output vstream has no NMS operations on the CPU.

Parameters *max_proposals_per_class* - [in] NMS max proposals per class to set.

Returns Upon success, returns *HAIGO_SUCCESS*. Otherwise, returns a *halilo_status* error.

```
halilo_status set_nms_max_accumulated_mask_size(uint32_t max_accumulated_mask_size)
```

Set maximum accumulated mask size for all the detections in a frame.

Note: Used in order to change the output buffer frame size, in cases where the output buffer is too small for all the segmentation detections.

Note: This function must be called before starting inference! This function will fail in cases where the output vstream has no NMS operations on the CPU.

Parameters `max_accumulated_mask_size - [in]` NMS max accumulated mask size.

Public Static Functions

`static hailo_status clear(std::vector<OutputVStream> &vstreams)`

Clears the vstreams' pipeline buffers.

Parameters `vstreams - [in]` The vstreams to be cleared.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`static hailo_status clear(std::vector<std::reference_wrapper<OutputVStream>> &vstreams)`

Clears the vstreams' pipeline buffers.

Parameters `vstreams - [in]` The vstreams to be cleared.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`class hailort :: VStreamsBuilder`

Contains the virtual streams creation functions

Public Static Functions

`static Expected<std::pair<std::vector<InputVStream>, std::vector<OutputVStream>> create_vstreams(ConfiguredNetworkGroup)`

Creates input virtual streams and output virtual streams.

Parameters

- `net_group - [in]` Configured network group that owns the streams.
- `unused - [in]` Unused.
- `format_type - [in]` The default format type for all virtual streams.
- `network_name - [in]` Request to create vstreams of specific network inside the configured network group. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns `Expected` of a pair of input vstreams and output vstreams. Otherwise, returns `Unexpected` of `hailo_status` error.

`static Expected<std::pair<std::vector<InputVStream>, std::vector<OutputVStream>> create_vstreams(ConfiguredNetworkGroup)`

Creates input virtual streams and output virtual streams.

Parameters

- `net_group - [in]` Configured network group that owns the streams.
- `vstreams_params - [in]` A `hailo_vstream_params_t` containing default params for all virtual streams.
- `network_name - [in]` Request to create vstreams of specific network inside the configured network group. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns `Expected` of a vector of input virtual streams. Otherwise, returns `Unexpected` of `hailo_status` error.

`static Expected<std::vector<InputVStream>> create_input_vstreams(ConfiguredNetworkGroup ...)`

Creates input virtual streams.

Parameters

- `net_group` - **[in]** Configured network group that owns the streams.
- `inputs_params` - **[in]** Map of input vstreams <name, params> to create input vstreams from.

Returns Upon success, returns *Expected* of a vector of input virtual streams. Otherwise, returns *Unexpected* of `hailo_status` error.

```
static Expected<std::vector<OutputVStream>> create_output_vstreams(ConfiguredNetworkGroup ...)
```

Creates output virtual streams.

Note: If not creating all output vstreams together, one should make sure all vstreams from the same group are created together. See `ConfiguredNetworkGroup::make_output_vstream_params_groups`

Parameters

- `net_group` - **[in]** Configured network group that owns the streams.
- `outputs_params` - **[in]** Map of output vstreams <name, params> to create output vstreams from.

Returns Upon success, returns *Expected* of a vector of output virtual streams. Otherwise, returns *Unexpected* of `hailo_status` error.

```
class hailort :: InferVStreams
```

Pipeline used to run inference

Public Functions

```
hailo_status infer(const std::map<std::string, MemoryView> &input_data, std::map<std::string, ...>)
```

Run inference on dataset `input_data`.

Note: `ConfiguredNetworkGroup` must be activated before calling this function.

Note: The size of each element in `input_data` and `output_data` must match the frame size of the matching vstream name multiplied by `frames_count`.

Note: If at least one input/output of some network is present, all inputs and outputs of that network must also be present.

Parameters

- `input_data` - **[in]** A mapping of vstream name to MemoryView containing input dataset for inference.
- `output_data` - **[out]** A mapping of vstream name to MemoryView containing the inference output data.
- `frames_count` - **[in]** The amount of inferred frames.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

```
Expected<std::reference_wrapper<InputVStream>> get_input_by_name(const std::string &name)
```

Get `InputVStream` by name.

Parameters `name` - **[in]** The vstream's name.

Returns Upon success, returns *Expected* of *InputVStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::reference_wrapper<*OutputVStream*>> **get_output_by_name**(const std::string &name)
Get *OutputVStream* by name.

Parameters name – [in] The vstream's name.

Returns Upon success, returns *Expected* of *OutputVStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

std::vector<std::reference_wrapper<*InputVStream*>> **get_input_vstreams**()

Returns Returns a vector of all *InputVStreams*.

std::vector<std::reference_wrapper<*OutputVStream*>> **get_output_vstreams**()

Returns Returns a vector of all *OutputVStreams*.

hailo_status **set_nms_score_threshold**(*float32_t* threshold)

Set NMS score threshold, used for filtering out candidates. Any box with score<TH is suppressed.

Note: This function will fail in cases where there is no output with NMS operations on the CPU.

Parameters threshold – [in] NMS score threshold to set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status **set_nms_iou_threshold**(*float32_t* threshold)

Set NMS intersection over union overlap Threshold, used in the NMS iterative elimination process where potential duplicates of detected items are suppressed.

Note: This function will fail in cases where there is no output with NMS operations on the CPU.

Parameters threshold – [in] NMS IoU threshold to set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status **set_nms_max_proposals_per_class**(*uint32_t* max_proposals_per_class)

Set a limit for the maximum number of boxes per class.

Note: This function must be called before starting inference! This function will fail in cases where there is no output with NMS operations on the CPU.

Parameters max_proposals_per_class – [in] NMS max proposals per class to set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status **set_nms_max_accumulated_mask_size**(*uint32_t* max_accumulated_mask_size)

Set maximum accumulated mask size for all the detections in a frame.

Note: Used in order to change the output buffer frame size, in cases where the output buffer is too small for all the segmentation detections.

Note: This function must be called before starting inference! This function will fail in cases where the output vstream has no NMS operations on the CPU.

Parameters `max_accumulated_mask_size - [in]` NMS max accumulated mask size.

Public Static Functions

static `Expected<InferVStreams>` `create(ConfiguredNetworkGroup &net_group, const std::map<std::string, ... >`
Creates vstreams pipelines to be used later for inference by calling the `InferVStreams::infer()` function.

Note: If at least one input/output of some network is present, all inputs and outputs of that network must also be present.

Parameters

- `net_group - [in]` A `ConfiguredNetworkGroup` to run the inference on.
- `input_params - [in]` A mapping of input vstream name to its' params.
Can be achieved by calling `ConfiguredNetworkGroup::make_input_vstream_params()` or `Hef::make_input_vstream_params` functions.
- `output_params - [in]` A mapping of output vstream name to its' params.
Can be achieved by calling `ConfiguredNetworkGroup::make_output_vstream_params()` or `Hef::make_output_vstream_params` functions.

Returns Upon success, returns `Expected` of `InferVStreams`. Otherwise, returns `Unexpected` of `hailo_status` error.

15.9. DMA Buffer Mapping API Functions

class `hailort` :: `DmaMappedBuffer`

A wrapper class for mapping and unmapping buffers using `VDevice::dma_map` and `VDevice::dma_unmap` (or their variants for `Device`).

The `DmaMappedBuffer` class provides a convenient way to keep a DMA mapping on a buffer active. It encapsulates the functionality of mapping and unmapping buffers using `VDevice::dma_map` and `VDevice::dma_unmap`, as well as their variants for `Device`.

Example:

```
// Create a DmaMappedBuffer object for a VDevice
void* user_address = ...;
size_t size = ...;
hailo_dma_buffer_direction_t direction = ...;
Expected<DmaMappedBuffer> mapped_buffer = DmaMappedBuffer::create(vdevice, user_address, size, direction);
if (!mapped_buffer.has_value()) {
    // Handle error
} else {
    // Use the mapped buffer
}
```

Note: The buffer pointed to by address cannot be released until this object is destroyed.

Public Functions

```
~DmaMappedBuffer()
```

The destructor automatically unmaps the buffer.

Public Static Functions

```
static Expected<DmaMappedBuffer> create(VDevice &vdevice, void *user_address, size_t size, ...)
```

Creates a *DmaMappedBuffer* object for a *VDevice*.

Parameters

- *vdevice* – The *VDevice* object to use for mapping the buffer.
- *user_address* – The user address of the buffer to be mapped.
- *size* – The size of the buffer to be mapped.
- *direction* – The direction of the DMA transfer.

Returns An *Expected* object containing the created *DmaMappedBuffer* on success, or an error on failure.

```
static Expected<DmaMappedBuffer> create(Device &device, void *user_address, size_t size, ...)
```

Creates a *DmaMappedBuffer* object for a *Device*.

Parameters

- *device* – The *Device* object to use for mapping the buffer.
- *user_address* – The user address of the buffer to be mapped.
- *size* – The size of the buffer to be mapped.
- *direction* – The direction of the DMA transfer.

Returns An *Expected* object containing the created *DmaMappedBuffer* on success, or an error on failure.

15.10. Common HailoRT Utility Functions

```
namespace hailort
```

```
class HailoRTCommon
```

```
#include <hailort_common.hpp>
```

Public Static Functions

```
static uint32_t get_nms_host_shape_size(const hailo_nms_info_t &nms_info)
```

Deprecated: use *get_nms_by_class_host_shape_size* instead

```
static uint32_t get_nms_host_shape_size(const hailo_nms_shape_t &nms_shape)
```

Deprecated: use *get_nms_by_class_host_shape_size* instead

```
static inline constexpr uint32_t get_nms_by_class_host_shape_size(const ...)
```

Gets the NMS host shape size (number of elements) from NMS info.

Note: The size in bytes can be calculated using *get_nms_by_class_host_frame_size(const hailo_nms_info_t &nms_info, const hailo_format_t &format)*.

Parameters *nms_info* – [in] The NMS info to get shape size from.

Returns The host shape size (number of elements).

```
static inline constexpr uint32_t get_nms_by_class_host_shape_size(const ...)  
Gets the NMS host shape size (number of elements) from NMS shape.
```

Note: The size in bytes can be calculated using `get_nms_host_frame_size(const hailo_nms_shape_t &nms_shape, const hailo_format_t &format)`.

Parameters `nms_shape` - **[in]** The NMS shape to get size from.
Returns The host shape size (number of elements).

```
template<typename T, typename U>  
static inline constexpr T align_to(T num, U alignment)
```

Rounds an integer value up to the next multiple of a specified size.

Parameters

- `num` - **[in]** Original number.
- `alignment` - **[in]** Returned number should be aligned to this parameter.

Returns aligned number

```
static inline void *align_to(void *addr, size_t alignment)
```

```
static inline constexpr uint32_t get_shape_size(const hailo_3d_image_shape_t &shape, ...)
```

Gets the shape size.

Parameters

- `shape` - **[in]** The shape to get size from.
- `row_alignment` - **[in]** The size the shape row is aligned to.

Returns The shape size.

```
static inline constexpr uint8_t get_data_bytes(hailo_format_type_t type)
```

Gets the size of each element in bytes from buffer's format type.

Parameters `type` - **[in]** A `hailo_format_type_t` object.
Returns The data bytes.

```
static inline Expected<hailo_format_type_t> get_format_type(uint32_t hw_data_bytes)
```

Gets the format type of a stream by the hw_data_bytes parameter.

Parameters `hw_data_bytes` - **[in]** The stream's info's hw_data_bytes parameter.
Returns Upon success, returns `Expected` of `hailo_format_type_t`, The format type that the hw_data_type correlates to. Otherwise, returns `Unexpected` of `hailo_status` error.

```
static inline std::string get_format_type_str(const hailo_format_type_t &type)
```

Gets a string representation of the given format type.

Parameters `type` - **[in]** A `hailo_format_type_t` object.
Returns The string representation of the format type.

```
static inline std::string get_power_mode_str(const hailo_power_mode_t &mode)
```

Gets a string representation of the given power mode.

Parameters `mode` - **[in]** A `hailo_power_mode_t` object.
Returns The string representation of the power mode.

```
static inline std::string get_latency_measurement_str(const ...)
```

Gets a string representation of the given latency measurement flags.

Parameters `flags` - **[in]** A `hailo_latency_measurement_flags_t` object.
Returns The string representation of the latency measurement flags.

```
static inline std::string get_scheduling_algorithm_str(const ...)
```

Gets a string representation of the given scheduling algorithm.

Parameters `scheduling_algo` - **[in]** A `hailo_scheduling_algorithm_t` object.
Returns The string representation of the scheduling algorithm.

```
static inline std::string get_device_arch_str(const hailo_device_architecture_t &arch)
```

Gets a string representation of the given device architecture.

Parameters `arch` - **[in]** A `hailo_device_architecture_t` object.

Returns The string representation of the device architecture.

```
static inline std::string get_format_order_str(const hailo_format_order_t &order)
```

Gets a string representation of the given format order.

Parameters *order* - [in] A *hailo_format_order_t* object.

Returns The string representation of the format order.

```
static inline std::string get_log_type_str(const hailo_log_type_t &type)
```

Gets a string representation of the given log type.

Parameters *type* - [in] A *hailo_log_type_t* object.

Returns The string representation of the log type.

```
static inline constexpr uint8_t get_format_data_bytes(const hailo_format_t &format)
```

Gets the size of each element in bytes from buffer's format.

Parameters *format* - [in] A *hailo_format_t* object.

Returns The format's data bytes.

```
static inline constexpr uint32_t get_nms_by_class_host_frame_size(const ...)
```

Gets NMS host frame size in bytes by nms info and buffer format.

Parameters

- *nms_info* - [in] A *hailo_nms_info_t* object.
- *format* - [in] A *hailo_format_t* object.

Returns The NMS host frame size in bytes.

```
static uint32_t get_nms_host_frame_size(const hailo_nms_shape_t &nms_shape, const ...)
```

Gets NMS host frame size in bytes by nms shape and buffer format.

Parameters

- *nms_shape* - [in] A *hailo_nms_shape_t* object.
- *format* - [in] A *hailo_format_t* object.

Returns The NMS host frame size in bytes.

```
static inline constexpr uint32_t get_nms_with_byte_mask_host_frame_size(const ...)
```

Gets HAILO_NMS_WITH_BYTE_MASK host frame size in bytes by nms_shape.

Parameters *nms_shape* - [in] The NMS shape to get size from.

Returns The HAILO_NMS_WITH_BYTE_MASK host frame size.

```
static inline constexpr uint32_t get_nms_by_score_host_frame_size(const ...)
```

Gets HAILO_NMS_BY_SCORE host frame size in bytes by nms_shape.

Parameters *nms_shape* - [in] The NMS shape to get size from.

Returns The HAILO_NMS_BY_SCORE host frame size.

```
static inline constexpr uint32_t get_nms_hw_frame_size(const hailo_nms_info_t &nms_info)
```

Gets NMS hw frame size in bytes by nms info.

Parameters *nms_info* - [in] A *hailo_nms_info_t* object.

Returns The NMS hw frame size in bytes.

```
static inline constexpr uint32_t get_frame_size(const hailo_3d_image_shape_t &shape, const ...)
```

Gets frame size in bytes by image shape and format.

Parameters

- *shape* - [in] A *hailo_3d_image_shape_t* object.
- *format* - [in] A *hailo_format_t* object.

Returns The frame's size in bytes.

```
static inline constexpr uint32_t get_frame_size(const hailo_stream_info_t &stream_info, ...)
```

Gets frame size in bytes by stream info and transformation params.

Parameters

- *stream_info* - [in] A *hailo_stream_info_t* object.
- *trans_params* - [in] A *hailo_transform_params_t* object.

Returns The frame's size in bytes.

```
static inline constexpr uint32_t get_frame_size(const hailo_vstream_info_t &vstream_info, ...)
```

Gets frame size in bytes by stream info and transformation params.

Parameters

- *vstream_info* - **[in]** A *hailo_vstream_info_t* object.
- *format* - **[in]** A *hailo_format_t* object.

Returns The frame's size in bytes.

```
static inline constexpr uint32_t get_periph_frame_size(const hailo_3d_image_shape_t ...)
```

Gets periph frame size in bytes by image shape and format - periph frame size is amount of bytes transferred through peripherals which must be aligned to HW_DATA_ALIGNMENT (8). Note: this function always aligns to next largest HW_DATA_ALIGNMENT

Parameters

- *shape* - **[in]** A *hailo_3d_image_shape_t* object.
- *format* - **[in]** A *hailo_format_t* object.

Returns The periph frame's size in bytes.

```
static inline constexpr bool is_vdma_stream_interface(hailo_stream_interface_t ...)
```

```
static inline constexpr bool is_nms(const hailo_vstream_info_t &vstream_info)
```

```
static inline constexpr bool is_nms(const hailo_stream_info_t &stream_info)
```

```
static inline constexpr bool is_nms(const hailo_format_order_t &order)
```

```
static inline constexpr bool is_nms_by_class(const hailo_format_order_t &order)
```

```
static inline constexpr bool is_nms_by_score(const hailo_format_order_t &order)
```

```
static inline bool is_hailo1x_device_type(const hailo_device_architecture_t dev_arch)
```

```
static Expected<hailo_device_id_t> to_device_id(const std::string &device_id)
```

```
static Expected<std::vector<hailo_device_id_t>> to_device_ids_vector(const ...)
```

```
static Expected<hailo_pix_buffer_t> as_hailo_pix_buffer(MemoryView memory_view, ...)
```

Public Static Attributes

```
static const uint32_t BBOX_PARAMS = sizeof(hailo_bbox_t) / sizeof(uint16_t)
```

```
static const uint32_t DETECTION_BY_SCORE_SIZE = sizeof(hailo_detection_t)
```

```
static const uint32_t DETECTION_WITH_BYTE_MASK_SIZE =
sizeof(hailo_detection_with_byte_mask_t)
```

```
static const uint32_t DETECTION_COUNT_SIZE = sizeof(uint16_t)
```

```
static const uint32_t MAX_DEFUSED_LAYER_COUNT = 9
```

```
static const size_t HW_DATA_ALIGNMENT = 8
```

```
static const uint32_t MUX_INFO_COUNT = 32
```

```
static const uint32_t MAX_MUX_PREDECESSORS = 4
```

```
static const uint16_t ETH_INPUT_BASE_PORT = 32401
```

```
static const uint16_t ETH_OUTPUT_BASE_PORT = 32501  
  
static const uint32_t MAX_NMS_BURST_SIZE = 65536  
  
static const size_t DMA_ABLE_ALIGNMENT_WRITE_HW_LIMITATION = 64  
  
static const size_t DMA_ABLE_ALIGNMENT_READ_HW_LIMITATION = 4096
```

15.11. Runtime Statistics

```
class hailort::AccumulatorResults
```

Results obtained by an [Accumulator](#) at a given point in time via [Accumulator::get_and_clear](#) or [Accumulator::get](#)
Public Functions

```
inline Expected<size_t> count() const
```

Returns Returns [Expected](#) of the number of datapoints added to the [Accumulator](#).

```
inline Expected<double> min() const
```

Returns Returns [Expected](#) of the minimal value added to the [Accumulator](#), or [Unexpected](#) of [HALO_UNINITIALIZED](#) if no data has been added.

```
inline Expected<double> max() const
```

Returns Returns [Expected](#) of the maximal value added to the [Accumulator](#), or [Unexpected](#) of [HALO_UNINITIALIZED](#) if no data has been added.

```
inline Expected<double> mean() const
```

Returns Returns [Expected](#) of the mean of the values added to the [Accumulator](#), or [Unexpected](#) of [HALO_UNINITIALIZED](#) if no data has been added.

```
inline Expected<double> var() const
```

Returns Returns [Expected](#) of the sample variance of the values added to the [Accumulator](#), or [Unexpected](#) of [HALO_UNINITIALIZED](#) if no data has been added.

```
inline Expected<double> sd() const
```

Returns Returns [Expected](#) of the sample standard deviation of the values added to the [Accumulator](#), or [Unexpected](#) of [HALO_UNINITIALIZED](#) if no data has been added.

```
inline Expected<double> mean_sd() const
```

Returns Returns [Expected](#) of the sample standard deviation of the mean of values added to the [Accumulator](#), or [Unexpected](#) of [HALO_UNINITIALIZED](#) if no data has been added.

```
template<typename T, std::enable_if_t<std::is_arithmetic<T>::value, int> = 0>
```

```
class hailort::Accumulator
```

The [Accumulator](#) interface supports the measurement of various statistics incrementally. I.e. upon each addition of a measurement to the [Accumulator](#), via [Accumulator::add_data_point](#), all the statistics are updated. Implementations of this interface are to be thread-safe, meaning that adding measurements in one thread, while another thread reads the current statistics (via the various getters provided by the interface) will produce correct values.

Public Functions

```
inline Accumulator(const std::string &data_type)  
Constructs a new Accumulator with a given data_type
```

Parameters *data_type* – The type of data that will be measured by the *Accumulator*. Used to differentiate between different types of measurements (e.g. fps, latency).

```
inline std::string get_data_type() const
```

Returns The *data_type* of the *Accumulator*.

```
virtual void add_data_point(T data, uint32_t samples_count = 1) = 0
```

Add a new measurement to the *Accumulator*, updating the statistics measured.

Note: Implementations of this interface are to update the statistics in constant time

Parameters

- *data* – The measurement to be added.
- *samples_count* – The weight of the measurement to be considered in average calculations.

```
virtual AccumulatorResults get_and_clear() = 0
```

Gets the current statistics of the data added to the *Accumulator*, clearing the statistics afterwards.

Returns The current statistics of the data added to the *Accumulator*

```
virtual AccumulatorResults get() const = 0
```

Returns The current statistics of the data added to the *Accumulator*

```
virtual Expected<size_t> count() const = 0
```

Returns The number of datapoints added to the *Accumulator*.

```
virtual Expected<double> min() const = 0
```

Returns Returns *Expected* of the minimal value added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

```
virtual Expected<double> max() const = 0
```

Returns Returns *Expected* of the maximal value added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

```
virtual Expected<double> mean() const = 0
```

Returns Returns *Expected* of the mean of the values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

```
virtual Expected<double> var() const = 0
```

Returns Returns *Expected* of the sample variance of the values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

```
virtual Expected<double> sd() const = 0
```

Returns Returns *Expected* of the sample standard deviation of the values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

```
virtual Expected<double> mean_sd() const = 0
```

Returns Returns *Expected* of the sample standard deviation of the mean of values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

15.12. Error Handling

```
class Unexpected
```

Unexpected is an object containing *hailo_status* error, used when an unexpected outcome occurred.

```
template<typename T>
```

```
class hailort : :Expected
```

Expected<T> is either a T or the *hailo_status* preventing T to be created.

Public Functions

```
inline bool has_value() const
```

Checks whether the object contains a value.

```
inline T &value() &
```

Returns the contained value.

Note: This method must be called with a valid value inside! otherwise it can lead to undefined behavior.

```
inline const T &value() const &
```

Returns the contained value.

Note: This method must be called with a valid value inside! otherwise it can lead to undefined behavior.

```
inline hailo_status status() const
```

Returns the status.

```
inline T release()
```

Releases ownership of its stored value, by returning its value and making this object *Unexpected*.

Note: This method must be called with a valid value inside! otherwise it can lead to undefined behavior.

```
template<typename ...Args>
```

```
inline T expect(Args&&...)
```

If the object contains a value, releases ownership of the stored value by returning its value and making this object *Unexpected*. If the object is *Unexpected*, throws an exception of type *hailort_error*.

Note: Using this method requires compilation with exceptions.

```
class hailort::hailort_error
```

hailort_error is an Exception object that inherits from std::runtime_error. Using this class requires compilation with exceptions

Public Functions

inline *hailo_status* status() const

Returns the error status that caused this exception.

15.13. Type Definitions

```
using NotificationCallback = std::function<void(Device &device, const hailo_notification_t &notification,  
void *opaque)>
```

16. HailoRT GenAI C++ API Reference

HailoRT is Hailo's runtime library. The GenAI-C++ API is used for running inference over Generative-AI compiled models (HEFs) in a C++ program.

Note: (for C++ API users) In order to maintain ABI Integrity between libhailort and your application that uses the C++ API, you should [compile libhailort](#) instead of using the pre-compiled binaries.

16.1. LLM API

```
class hailort::genai::LLMParams
```

Parameters to configure the [LLM](#) model

Public Functions

```
LLMParams(const std::string &hef_path, const std::string &lora_name = "", bool ...)
```

Creates [LLMParams](#) with optional model configuration.

Parameters

- `hef_path` - **[in]** The path of the [Hef](#) file. If empty, [set_model\(\)](#) must be called later.
- `lora_name` - **[in]** The name of the chosen LoRA. Default is empty string.
- `optimize_memory_on_device` - **[in]** Whether to optimize memory usage on device by enabling client-side tokenization. When true, tokenization is performed on the host, reducing device memory usage. Requires libhailort to be compiled with `HAILO_BUILD_CLIENT_TOKENIZER=ON`. Default is false.

```
hailo\_status set_model(const std::string &hef_path, const std::string &lora_name = "")
```

Sets [LLM](#) model.

Parameters

- `hef_path` - **[in]** The path of the [Hef](#) file.
- `lora_name` - **[in]** The name of the chosen LoRA.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

```
const std::string &hef() const
```

Returns The [Hef](#) path of the [LLM](#) model.

```
const std::string &lora() const
```

Returns The LoRA name of the [LLM](#) model.

```
bool optimize_memory_on_device() const
```

Returns Whether memory optimization on device is enabled.

```
void set_optimize_memory_on_device(bool optimize_memory_on_device)
```

Sets whether to optimize memory usage on device.

Parameters `optimize_memory_on_device` - **[in]** Whether to enable client-side tokenization for memory optimization. When true, tokenization is performed on the host, reducing device memory usage. Requires libhailort to be compiled with `HAILO_BUILD_CLIENT_TOKENIZER=ON`.

```
class hailort::genai::LLM
```

Represents the [LLM](#) Model. Manages the lifecycle and configuration of a large language model instance. Provides methods to generate text, and manage the conversation context that is stored on the Hailo device. The entire [LLM](#) pipeline is offloaded to the Hailo device (tokenization, pre and post-process, etc), allowing for efficient processing of large language models.

Public Functions

Expected<LLMGeneratorParams> [create_generator_params\(\)](#)

Creates an [LLMGeneratorParams](#) object with the model's default values.

Returns Upon success, returns *Expected* of [LLMGeneratorParams](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<LLMGenerator> [create_generator\(const LLMGeneratorParams ¶ms\)](#)

Creates an [LLMGenerator](#) object from the provided generation parameters.

Note: Only one generator shall be active at any time to ensure precision in parameter handling. A subsequent generator must only be created once the previous one has been fully destructed. Creating two generators at the same time may lead to invalid usage and undefined behavior.

Parameters `params` - **[in]** The [LLMGeneratorParams](#) used to set the generator parameters.

Returns Upon success, returns *Expected* of [LLMGenerator](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<LLMGenerator> [create_generator\(\)](#)

Creates an [LLMGenerator](#) object using the model's default generator parameters.

Note: Only one generator shall be active at any time to ensure precision in parameter handling. A subsequent generator must only be created once the previous one has been fully destructed. Creating two generators at the same time may lead to invalid usage and undefined behavior.

Returns Upon success, returns *Expected* of [LLMGenerator](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<LLMGeneratorCompletion> [generate\(const std::vector<std::string> &prompt_json_strings, ...\)](#)

Generates text directly using structured JSON prompts and default generation parameters without explicitly creating a generator. This is a convenience method that creates a generator with default parameters, writes the structured prompt, and initiates generation.

Note: This method uses the model's default generation parameters.

Note: This method is equivalent to: [create_generator\(\)](#) -> [write\(prompt_json_strings\)](#) -> [generate\(\)](#).

Note: Since this function creates a generator, and only one generator can be active at a time, no other generator can be created while this function is running.

Note: Only one [LLMGeneratorCompletion](#) instance should exist at a time. Creating multiple instances concurrently may lead to undefined behavior. A subsequent [LLMGeneratorCompletion](#) must

only be created once the previous one has been fully destructed.

Note: Tools and system role messages can only be provided on a fresh context. Providing them on consecutive calls will result in an error.

Parameters

- **prompt_json_strings - [in]** A vector of JSON strings representing the structured prompt.
- **tools_json_strings - [in]** A vector of JSON strings representing the tools, defaults to empty vector. If provided, the tools will be used to generate the response. Each string should be a valid JSON string representing a tool (e.g., [{"name": "tool1", "description": "Tool 1", "parameters": {"param1": "value1", "param2": "value2"}}]).

Returns Upon success, returns *Expected* of [LLMGeneratorCompletion](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<[LLMGeneratorCompletion](#)> **generate**(const [LLMGeneratorParams](#) ¶ms, const ...)

Generates text directly using structured JSON prompts without explicitly creating a generator. This is a convenience method that creates a generator, writes the structured prompt, and initiates generation.

Note: This method is equivalent to: `create_generator(params) -> write(prompt_json_strings) -> generate()`.

Note: Since this function creates a generator, and only one generator can be active at a time, no other generator can be created while this function is running.

Note: Only one [LLMGeneratorCompletion](#) instance should exist at a time. Creating multiple instances concurrently may lead to undefined behavior. A subsequent [LLMGeneratorCompletion](#) must only be created once the previous one has been fully destructed.

Note: Tools and system role messages can only be provided on a fresh context. Providing them on consecutive calls will result in an error.

Parameters

- **params - [in]** The [LLMGeneratorParams](#) used to configure generation.
- **prompt_json_strings - [in]** A vector of JSON strings representing the structured prompt.
- **tools_json_strings - [in]** A vector of JSON strings representing the tools, defaults to empty vector. If provided, the tools will be used to generate the response. Each string should be a valid JSON string representing a tool (e.g., [{"name": "tool1", "description": "Tool 1", "parameters": {"param1": "value1", "param2": "value2"}}]).

Returns Upon success, returns *Expected* of [LLMGeneratorCompletion](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<std::vector<int>> **tokenize**(const std::string &prompt)

Tokenizes a given string into a vector of integers representing the tokens.

Parameters `prompt` - [in] The input string to tokenize.

Returns Upon success, returns *Expected* of vector of integers. Otherwise, returns *Unexpected* of `hailo_status` error.

Expected<size_t> `get_context_usage_size()`

Gets the current context usage of the *LLM* model.

Note: This function returns the total number of tokens currently stored in the model's context, including both input tokens and generated tokens from previous interactions.

Note: The context usage includes the conversation history and any tokens that have been processed but not yet cleared by `clear_context()`.

Returns Upon success, returns *Expected* of size_t representing the current number of tokens in the context. Otherwise, returns *Unexpected* of `hailo_status` error.

Expected<size_t> `max_context_capacity()`

Obtains the maximum context capacity of the *LLM* model.

Note: This function returns the maximum number of tokens that can be stored in the model's context, which is determined in the model compilation, and is not configurable.

Returns Upon success, returns *Expected* of size_t representing the maximum number of tokens that can be stored in the model's context. Otherwise, returns *Unexpected* of `hailo_status` error.

`hailo_status clear_context()`

Clears the conversation context of the *LLM*.

Note: This function must not be called while a generation is in progress

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

Expected<BufferPtr> `save_context()`

Saves the current context of the *LLM*, as a binary blob that can be used to load the context later on.

Note: The context is unique for a specific *LLM* model, and can be used to set the context using `load_context()`. Trying to set this context on a different *LLM* (e.g. QWEN context on a LLAMA model) will result in an error.

Returns Upon success, returns *Expected* of BufferPtr. Otherwise, returns *Unexpected* of `hailo_status` error.

`hailo_status load_context(const MemoryView &context)`

Loads a context to the *LLM*.

Note: The context is unique for a specific *LLM* model. It is retrieved using `save_context()`. Trying to set a context from a different *LLM* (e.g. QWEN context on a LLAMA model) will result in an error.

Parameters context - [in] The context to set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status set_generation_recovery_sequence(const std::string &sequence)

Sets a custom generation recovery sequence to be injected into the model when generation ends abruptly (i.e., due to max-token termination or abort, but not when reaching a stop-token). This sequence helps maintain model usability after abrupt terminations and is not counted towards the max-generated-tokens limit.

Note: If not set, the model's default sequence will be used. See also [get_generation_recovery_sequence\(\)](#).

Note: This sequence is NOT applied for LOGICAL_END_OF_GENERATION completions.

Parameters sequence - The recovery sequence to inject after abrupt generation endings.
Can be empty string.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<std::string> get_generation_recovery_sequence()

Gets the generation recovery sequence of the *LLM* model.

Returns Upon success, returns *Expected* of std::string, representing the generation recovery sequence of the *LLM* model. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::string> prompt_template()

Returns the prompt template used by the *LLM*.

Returns Upon success, returns *Expected* of std::string containing the prompt template. Otherwise, returns *Unexpected* of *hailo_status* error.

hailo_status set_stop_tokens(const std::vector<std::string> &stop_tokens)

Sets custom stop token sequences for generation. The generation process will stop when any of the provided sequences is encountered.

Note: This replaces any previously set stop tokens, including the built-in end-of-sentence tokens.

Note: Stop tokens are checked after each generated token, so very long sequences may impact performance.

Parameters stop_tokens - [in] A vector of strings, where each string represents a stop sequence. If a string is encoded to multiple tokens, the full sequence must be matched. An empty vector clears all custom stop tokens.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<std::vector<std::string>> get_stop_tokens()

Gets the currently configured stop token sequences.

Returns Upon success, returns *Expected* of vector of strings representing the current stop token sequences. Otherwise, returns *Unexpected* of *hailo_status* error.

Public Static Functions

static *Expected<LLM>* create (std::shared_ptr<*VDevice*> vdevice, const *LLMParams* &llm_params)

Creates an *LLM* model instance configured with the specified parameters.

Parameters

- *vdevice* - [in] The *VDevice* object used to communicate with the Hailo device.
- *llm_params* - [in] The *LLMParams* object used to configure the *LLM* model.

Returns Upon success, returns *Expected* of *LLM*. Otherwise, returns *Unexpected* of *halo_status* error.

```
class hailort::genai::LLMGeneratorParams
```

The *LLMGeneratorParams* represents the parameters for text generation, which can be changed during runtime for each generator.

Public Functions

halo_status set_temperature(*float32_t* temperature)

Sets the sampling temperature of the *LLM* model.

Parameters *temperature* - [in] The sampling temperature.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *halo_status* error.

float32_t temperature() const

Returns The sampling temperature.

halo_status set_top_p(*float32_t* top_p)

Sets the top_p parameter of the *LLM* model.

Parameters *top_p* - [in] The top_p sampling value.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *halo_status* error.

float32_t top_p() const

Returns The top_p sampling value.

halo_status set_top_k(*uint32_t* top_k)

Sets the top_k parameter of the *LLM* model.

Parameters *top_k* - [in] The top_k sampling value.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *halo_status* error.

uint32_t top_k() const

Returns The top_k sampling value.

halo_status set_frequency_penalty(*float32_t* frequency_penalty)

Sets the frequency_penalty parameter of the *LLM* model.

Parameters *frequency_penalty* - [in] The frequency_penalty for generated tokens.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *halo_status* error.

float32_t frequency_penalty() const

Returns The frequency_penalty for generated tokens.

`hailo_status set_max_generated_tokens(uint32_t max_generated_tokens)`

Sets the max_generated_tokens parameter of the [LLM](#) model.

Note: This number includes all special tokens such as start and end tokens.

Note: This number does not include the tokens of the generation recovery sequence. See also [LLM::set_generation_recovery_sequence\(\)](#).

Parameters `max_generated_tokens` - [\[in\]](#) The maximum number of tokens that can be generated, not including the input tokens.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

`uint32_t max_generated_tokens() const`

Returns The maximum number of tokens that can be generated.

`hailo_status set_do_sample(bool do_sample)`

Whether the [LLM](#) sampling should be statistical or greedy.

Parameters `do_sample` - [\[in\]](#) true meaning the model will perform statistical sampling, false meaning the model will perform greedy sampling.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

`bool do_sample() const`

Returns Whether the [LLM](#) sampling should be statistical or greedy.

`hailo_status set_seed(uint32_t seed)`

Sets the seed for the [LLM](#) model.

Note: If seed is not set, or is set to its default value HAILO_RANDOM_SEED - a random seed will be used.

Parameters `seed` - [\[in\]](#) The seed for the random number generator for statistical sampling.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

`uint32_t seed() const`

Returns The seed for the random number generator.

class `hailort`::`genai`::`LLMGenerator`

The [LLMGenerator](#) object is used to send input prompt to the [LLM](#) model for generating responses

Public Functions

`hailo_status write(const std::string &prompt)`

Writes an input prompt to the [LLM](#) model.

Note: Multiple consecutive writes will concatenate prompts without adding separators.

Note: Writing while the model is actively generating content can result in undefined behavior.

Note: If the [LLMGenerator](#) is destructed before [generate\(\)](#) is called, all inputs written to it will be discarded.

Parameters `prompt` - **[in]** The text prompt to be processed by the [LLM](#) model. The prompt is written as-is without any template formatting.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

`hailo_status write(const std::vector<std::string> &prompt_json_strings, const std::vector<std::string> ...)`

Writes a structured input prompt to the [LLM](#) model using JSON strings.

Note: In case a given key is not supported by the model's template, or if an expected key is missing (e.g., "content" without "role"), the word `None` will replace it.

Note: Multiple consecutive writes will concatenate prompts without adding separators.

Note: Writing while the model is actively generating content can result in undefined behavior.

Note: If the [LLMGenerator](#) is destructed before [generate\(\)](#) is called, all inputs written to it will be discarded.

Note: Tools and system role messages can only be provided on a fresh context. Providing them on consecutive calls will result in an error.

Parameters

- `prompt_json_strings` - **[in]** A vector of JSON strings representing the structured prompt (e.g., `R"({\"role\": \"system\", \"content\": \"You are a helpful assistant\"})"`, `R"({\"role\": \"user\", \"content\": \"Hello\"})"`). This structured input is combined with the model's prompt template (retrievable via [LLM::prompt_template\(\)](#)). Each element in the vector represents a message in JSON format.
- `tools_json_strings` - **[in]** A vector of JSON strings representing the tools, defaults to empty vector. If provided, the tools will be used to generate the response. Each string should be a valid JSON string representing a tool (e.g., `[{"name": "tool1", "description": "Tool 1", "parameters": {"param1": "value1", "param2": "value2"}}]`).

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

Expected<LLMGeneratorCompletion> generate()

Marks the end of input and initiates the generation process. Returns an [LLMGeneratorCompletion](#), which allows fetching generated tokens incrementally.

Note: Only one [LLMGeneratorCompletion](#) instance should exist at a time. Creating multiple instances concurrently may lead to undefined behavior. A subsequent [LLMGeneratorCompletion](#) must only be created once the previous one has been fully destructed.

Note: If the [LLMGenerator](#) is destructed before [generate\(\)](#) is called, all inputs written to it will be discarded.

Returns Upon success, returns *Expected* of *LLMGeneratorCompletion*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
class hailort::genai::LLMGeneratorCompletion
```

The *LLMGeneratorCompletion* object is used to read token completions

Public Functions

Expected<size_t> **read**(char *output, size_t output_size, std::chrono::milliseconds timeout = ...)

Reads the next token completion.

Note: The returned output is a UTF-8 encoded string.

Note: In case the next token completion is the replacement character (U+FFFD), this token will be buffered until the next printable character is received.

Parameters

- **output** - **[out]** The buffer to store the next generated token completion.
- **output_size** - **[in]** The output buffer size.
- **timeout** - **[in]** The timeout for the read operation.

Returns Upon success, returns *Expected* to size_t, representing the number of bytes read. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::string> **read**(std::chrono::milliseconds timeout = DEFAULT_READ_TIMEOUT)

Reads the next token completion.

Note: The returned output is a UTF-8 encoded string.

Note: In case the next token completion is the replacement character (U+FFFD), this token will be buffered until the next printable character is received.

Parameters

timeout - **[in]** The timeout for the read operation.

Returns Upon success, returns *Expected* of std::string, representing the next token completion. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::string> **read_all**(std::chrono::milliseconds timeout = HAILO_INFINITE_TIMEOUT)

Reads all remaining token completions and returns them as a complete string.

Note: This function will read tokens until the generation is complete (status is not GENERATING).

Note: The returned output is a UTF-8 encoded string.

Note: After calling this function, no further reads should be attempted on this *LLMGeneratorCompletion* object.

Parameters `timeout` - **[in]** The timeout for the entire read operation.

Returns Upon success, returns *Expected* of `std::string` containing all remaining generated text. Otherwise, returns *Unexpected* of `hailo_status` error.

`Status generation_status() const`

Returns the current generation status.

Note: Once this function returns a status indicating the end of generation (e.g., `MAX_TOKENS_REACHED`, `LOGICAL_END_OF_GENERATION` or `ABORTED`), no further reads should be attempted on this `LLMGeneratorCompletion` object, as it indicates that all token completions have been provided.

Returns Status - The current generation status, which can indicate:

- `GENERATING`: The generation process is ongoing.
- `MAX_TOKENS_REACHED`: The maximum number of tokens has been generated.
- `LOGICAL_END_OF_GENERATION`: The generation reached its logical end.
- `ABORTED`: The generation was aborted by the user.

`hailo_status abort()`

Aborts the current generation by injecting the model's generation recovery sequence (from `LLM::get_generation_recovery_sequence()` or `VLM::get_generation_recovery_sequence()`).

The function will:

- Stop the generation process.
- Flush any tokens that were generated but not yet read.
- Set the generation status to `ABORTED`.
- Allow a new generation to start once this function returns.

Note: This function blocks until the generation is fully aborted.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

16.2. VLM API

class `hailort::genai::VLMPParams`

Parameters to configure the `VLM` model

Public Functions

inline `VLMPParams` (const `std::string &hef_path`, `bool optimize_memory_on_device = false`)

Constructs a `VLMPParams` object with the specified `HeF` path.

Parameters

- `hef_path` - **[in]** The path of the `HeF` file.
- `optimize_memory_on_device` - **[in]** Whether to optimize memory usage on device by enabling client-side tokenization. When true, tokenization is performed on the host, reducing device memory usage. Requires libhailort to be compiled with `HAILO_BUILD_CLIENT_TOKENIZER=ON`. Default is false.

```
hailo_status set_model(const std::string &hef_path)
    Sets VLM model.

Parameters hef_path - [in] The path of the Hef file.

Returns Upon success, returns HAILO_SUCCESS. Otherwise, returns a hailo_status error.

const std::string &hef() const

Returns The Hef path of the VLM model.

bool optimize_memory_on_device() const

Returns Whether memory optimization on device is enabled.

void set_optimize_memory_on_device(bool optimize_memory_on_device)
    Sets whether to optimize memory usage on device.

Parameters optimize_memory_on_device - [in] Whether to enable client-side tokenization for memory optimization. When true, tokenization is performed on the host, reducing device memory usage. Requires libhailort to be compiled with HAILO_BUILD_CLIENT_TOKENIZER=ON.
```

class *hailort*::genai::VLM

Represents the *VLM* Model. Manages the lifecycle and configuration of a vision language model instance. Provides methods to generate text, and manage the conversation context that is stored on the Hailo device. The entire *VLM* pipeline is offloaded to the Hailo device (tokenization, pre and post-process, etc), allowing for efficient processing of vision language models.

Public Functions

Expected<LLMGeneratorParams> create_generator_params()

Creates an *LLMGeneratorParams* object with the model's default values.

Returns Upon success, returns *Expected* of *LLMGeneratorParams*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<VLMGenerator> create_generator(const *LLMGeneratorParams* ¶ms)

Creates an *VLMGenerator* object from the provided generation parameters.

Parameters params - [in] The *LLMGeneratorParams* used to set the generator parameters.

Returns Upon success, returns *Expected* of *VLMGenerator*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<VLMGenerator> create_generator()

Creates an *VLMGenerator* object using the model's default generator parameters.

Returns Upon success, returns *Expected* of *VLMGenerator*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<LLMGeneratorCompletion> generate(const std::vector<std::string> &messages_json_strings, ...)

Generates text directly using structured JSON messages and input frames with default generation parameters without explicitly creating a generator. This is a convenience method that creates a generator with default parameters, writes the structured prompt, and initiates generation.

Note: This method uses the model's default generation parameters.

Note: This method is equivalent to: *create_generator()* -> *generate(messages_json_strings, input_frames)*.

Note: Since this function creates a generator, and only one generator can be active at a time, no other generator can be created while this function is running.

Note: The number of input_frames must equal the total number of image entries across all messages; the same requirement applies to input_videos.

Note: Only one [LLMGeneratorCompletion](#) instance should exist at a time. Creating multiple instances concurrently may lead to undefined behavior. A subsequent [LLMGeneratorCompletion](#) must only be created once the previous one has been fully destructed.

Note: System role messages can only be provided on a fresh context. Providing them on consecutive calls will result in an error.

Note: Each frame or video frame should be in the format expected by the [VLM](#) model.

Parameters

- `messages_json_strings` - **[in]** A vector of JSON strings representing structured messages.
- `input_frames` - **[in]** The input frames to be sent to the [VLM](#) model, corresponding to image placeholders in the messages. The number of frames must match the number of image references in the messages.
- `input_videos` - **[in]** The input videos to be sent to the [VLM](#) model, can be empty. Each video is represented as a vector of raw frames. The number of videos must match the number of video references in the messages.

Returns Upon success, returns [Expected](#) of [LLMGeneratorCompletion](#). Otherwise, returns [Unexpected](#) of [hailo_status](#) error.

Expected<LLMGeneratorCompletion> `generate(const LLMGeneratorParams ¶ms, const ...)`

Generates text directly using structured JSON messages and input frames without explicitly creating a generator. This is a convenience method that creates a generator, writes the structured prompt, and initiates generation.

Note: This method is equivalent to: `create_generator(params) -> generate(messages_json_strings, input_frames)`.

Note: Since this function creates a generator, and only one generator can be active at a time, no other generator can be created while this function is running.

Note: The number of input_frames must equal the total number of image entries across all messages; the same requirement applies to input_videos.

Note: Only one [LLMGeneratorCompletion](#) instance should exist at a time. Creating multiple instances concurrently may lead to undefined behavior. A subsequent [LLMGeneratorCompletion](#) must

only be created once the previous one has been fully destructed.

Note: System role messages can only be provided on a fresh context. Providing them on consecutive calls will result in an error.

Note: Each frame or video frame should be in the format expected by the [VLM](#) model.

Parameters

- **params** - **[in]** The [LLMGeneratorParams](#) used to configure generation.
- **messages_json_strings** - **[in]** A vector of JSON strings representing structured messages.
- **input_frames** - **[in]** The input frames to be sent to the [VLM](#) model, corresponding to image placeholders in the messages. The number of frames must match the number of image references in the messages.
- **input_videos** - **[in]** The input videos to be sent to the [VLM](#) model, can be empty. Each video is represented as a vector of raw frames. The number of videos must match the number of video references in the messages.

Returns Upon success, returns [Expected](#) of [LLMGeneratorCompletion](#). Otherwise, returns [Unexpected](#) of [hailo_status](#) error.

[Expected](#)<std::vector<int> tokenization(const std::string &prompt)

Tokenizes a given string into a vector of integers representing the tokens.

Parameters **prompt** - **[in]** The input string to tokenize.

Returns Upon success, returns [Expected](#) of vector of integers. Otherwise, returns [Unexpected](#) of [hailo_status](#) error.

[Expected](#)<size_t> get_context_usage_size()

Gets the current context usage of the [VLM](#) model.

Note: This function returns the total number of tokens currently stored in the model's context, including both input tokens and generated tokens from previous interactions.

Note: The context usage includes the conversation history and any tokens that have been processed but not yet cleared by [clear_context\(\)](#).

Returns Upon success, returns [Expected](#) of size_t representing the current number of tokens in the context. Otherwise, returns [Unexpected](#) of [hailo_status](#) error.

[Expected](#)<size_t> max_context_capacity()

Obtains the maximum context capacity of the [VLM](#) model.

Note: This function returns the maximum number of tokens that can be stored in the model's context, which is determined in the model compilation, and is not configurable.

Returns Upon success, returns *Expected* of size_t representing the maximum number of tokens that can be stored in the model's context. Otherwise, returns *Unexpected* of *hailo_status* error.

hailo_status `clear_context()`

Clears the conversation context of the *VLM*.

Note: This function must not be called while a generation is in progress

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<BufferPtr> `save_context()`

Saves the current context of the *VLM*, as a binary blob that can be used to load the context later on.

Note: The context is unique for a specific *VLM* model, and can be used to set the context using *load_context()*. Trying to set this context on a different *VLM* (e.g. QWEN context on a LLAMA model) will result in an error.

Returns Upon success, returns *Expected* of BufferPtr. Otherwise, returns *Unexpected* of *hailo_status* error.

hailo_status `load_context(const MemoryView &context)`

Loads a context to the *VLM*.

Note: The context is unique for a specific *VLM* model. It is retrieved using *save_context()*. Trying to set a context from a different *VLM* (e.g. QWEN2.5-VL context on a QWEN2-VL model) will result in an error.

Parameters `context - [in]` The context to set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `set_generation_recovery_sequence(const std::string &sequence)`

Sets a custom generation recovery sequence to be injected into the model when generation ends abruptly (i.e., due to max-token termination or abort, but not when reaching a stop-token). This sequence helps maintain model usability after abrupt terminations and is not counted towards the max-generated-tokens limit.

Note: If not set, the model's default sequence will be used. See also *get_generation_recovery_sequence()*.

Note: This sequence is NOT applied for LOGICAL_END_OF_GENERATION completions.

Parameters `sequence -` The recovery sequence to inject after abrupt generation endings.
Can be empty string.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<std::string> `get_generation_recovery_sequence()`

Gets the generation recovery sequence of the *VLM* model.

Returns Upon success, returns *Expected* of std::string, representing the generation recovery sequence of the *VLM* model. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::string> **prompt_template()**

Returns the prompt template used by the [VLM](#).

Returns Upon success, returns *Expected* of std::string containing the prompt template. Otherwise, returns *Unexpected* of [hailo_status](#) error.

hailo_status **set_stop_tokens**(const std::vector<std::string> &stop_tokens)

Sets custom stop token sequences for generation. The generation process will stop when any of the provided sequences is encountered.

Note: This replaces any previously set stop tokens.

Note: The built-in end-of-sentence token from the model is always active regardless of this setting.

Note: Stop tokens are checked after each generated token, so very long sequences may impact performance.

Parameters **stop_tokens** - **[in]** A vector of strings, where each string represents a stop sequence. If a string is encoded to multiple tokens, the full sequence must be matched. An empty vector clears all custom stop tokens.

Returns Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

Expected<std::vector<std::string>> **get_stop_tokens()**

Gets the currently configured custom stop token sequences.

Note: This returns only the custom stop tokens set by the user, not the built-in model stop tokens.

Returns Upon success, returns *Expected* of vector of strings representing the current stop token sequences. Otherwise, returns *Unexpected* of [hailo_status](#) error.

uint32_t **input_frame_size()** const

Returns The size of the input frame.

const [hailo_3d_image_shape_t](#) &**input_frame_shape()** const

Returns The shape of the input frame.

const [hailo_format_type_t](#) &**input_frame_format_type()** const

Returns The format type of the input frame.

const [hailo_format_order_t](#) &**input_frame_format_order()** const

Returns The format order of the input frame.

Public Static Functions

static *Expected<VLM>* create (std::shared_ptr<*VDevice*> vdevice, const *VLMParams* &vlm_params)

Creates an *VLM* model instance configured with the specified parameters.

Parameters

- *vdevice* - **[in]** The *VDevice* object used to communicate with the Hailo device.
- *vlm_params* - **[in]** The *VLMParams* object used to configure the *VLM* model.

Returns Upon success, returns *Expected* of *VLM*. Otherwise, returns *Unexpected* of *hailo_status* error.

```
class hailort::genai::VLMGenerator
```

The *VLMGenerator* object is used to send input prompt and input frame to the *VLM* model for generating responses

Public Functions

Expected<LLMGeneratorCompletion> generate(const std::string &prompt, const ...)

Initiates generation from the input prompt and input frame to the *VLM* model. Returns an *LLMGeneratorCompletion*, which allows fetching generated tokens incrementally.

Note: Calling this function while the model is already generating may lead to undefined behavior.

Note: Only one *LLMGeneratorCompletion* instance should exist at a time. Creating multiple instances concurrently may lead to undefined behavior. A subsequent *LLMGeneratorCompletion* must only be created once the previous one has been fully destructed.

Note: Each frame or video frame should be in the format expected by the *VLM* model.

Parameters

- *prompt* - **[in]** The prompt to be processed by the *VLM* model. The prompt is written as-is without any template formatting.
- *input_frames* - **[in]** The input standalone-frames to be sent to the *VLM* model, Can be empty.
- *input_videos* - **[in]** The input videos to be sent to the *VLM* model, can be empty. Each video is represented as a vector of raw frames.

Returns Upon success, returns *Expected* of *LLMGeneratorCompletion*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<LLMGeneratorCompletion> generate(const std::vector<std::string> &messages_json_strings, ...)

Initiates generation from structured messages in JSON format and input frames to the *VLM* model. Returns an *LLMGeneratorCompletion*, which allows fetching generated tokens incrementally.

Note: Currently only supported content type is "image" and "text".

Note: In case a given key is not supported by the model's template, or if an expected key is missing (e.g., "content" without "role"), the word **None** will replace it.

Note: Image and video references in the JSON messages are replaced with appropriate vision tokens during template processing.

Note: The number of input_frames must equal the total number of image entries across all messages; the same requirement applies to input_videos.

Note: Calling this function while the model is already generating may lead to undefined behavior.

Note: Only one [LLMGeneratorCompletion](#) instance should exist at a time. Creating multiple instances concurrently may lead to undefined behavior. A subsequent [LLMGeneratorCompletion](#) must only be created once the previous one has been fully destructed.

Note: System role messages can only be provided on a fresh context. Providing them on consecutive calls will result in an error.

Note: Each frame or video frame should be in the format expected by the [VLM](#) model.

Parameters

- `messages_json_strings` - **[in]** A vector of JSON strings representing structured messages (e.g., `{R"({\"role\": \"user\", \"content\": [{"type\": \"text\", \"text\": \"Describe this image\"}, {"type\": \"image\"}]})\"}`). This structured input is combined with the model's prompt template (retrievable via [VLM::prompt_template\(\)](#)). Each element in the vector represents a message in JSON format following the [VLM](#) message structure.
- `input_frames` - **[in]** The input frames to be sent to the [VLM](#) model, corresponding to image placeholders in the messages. The number of frames must match the number of image references in the messages.
- `input_videos` - **[in]** The input videos to be sent to the [VLM](#) model, can be empty. Each video is represented as a vector of raw frames. The number of videos must match the number of video references in the messages.

Returns Upon success, returns [Expected](#) of [LLMGeneratorCompletion](#). Otherwise, returns [Unexpected](#) of [hailo_status](#) error.

16.3. Speech2Text API

```
class hailort ::genai::Speech2TextParams
```

Parameters to configure the [Speech2Text](#) model

Public Functions

`Speech2TextParams(std::string_view hef_path)`

Creates `Speech2TextParams` with optional model configuration.

Parameters `hef_path` - **[in]** The path of the `Hef` file. If empty, `set_model()` must be called later.

`halo_status set_model(std::string_view hef_path)`

Sets `Speech2Text` model.

Parameters `hef_path` - **[in]** The path of the `Hef` file.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `halo_status` error.

`std::string_view hef() const`

Returns The `Hef` path of the `Speech2Text` model.

class `hailort::genai::Speech2Text`

Represents the `Speech2Text` Model. Manages the lifecycle and configuration of a speech-to-text model instance. Provides methods to transcribe and translate speech to text. The entire speech-to-text pipeline is offloaded to the Hailo device (tokenization, pre and post-process, etc), allowing for efficient processing of speech-to-text.

Public Functions

`Expected<Speech2TextGeneratorParams> create_generator_params()`

Creates an `Speech2TextGeneratorParams` object with the model's default values.

Returns Upon success, returns `Expected` of `Speech2TextGeneratorParams`. Otherwise, returns `Unexpected` of `halo_status` error.

`Expected<std::string> generate_all_text(MemoryView audio_buffer, const ...)`

Generates a full transcription from the given audio data and returns it as a single string.

Parameters

- `audio_buffer` - **[in]** The audio data to process. Must be PCM float32 normalized to [-1.0, 1.0], mono, little-endian, at 16 kHz.
- `generator_params` - **[in]** The generator parameters to use for the generation.
- `timeout` - **[in]** The timeout for the operation.

Returns Upon success, returns `Expected` of `std::string`. Otherwise, returns `Unexpected` of `halo_status` error.

`Expected<std::string> generate_all_text(MemoryView audio_buffer, std::chrono::milliseconds ...)`

Generates a full transcription from the given audio data and returns it as a single string.

Note: The default generator parameters are used in this method, as returned by `create_generator_params()`.

Parameters

- `audio_buffer` - **[in]** The audio data to process. Must be PCM float32 normalized to [-1.0, 1.0], mono, little-endian, at 16 kHz.
- `timeout` - **[in]** The timeout for the operation.

Returns Upon success, returns `Expected` of `std::string`. Otherwise, returns `Unexpected` of `halo_status` error.

Expected<std::vector<[SegmentInfo](#)>> generate_all_segments (MemoryView audio_buffer, const ...)

Generates transcription segments from audio data, and returns them as a vector of [SegmentInfo](#).

Parameters

- `audio_buffer - [in]` The audio data to process. Must be PCM float32 normalized to [-1.0, 1.0), mono, little-endian, at 16 kHz.
- `generator_params - [in]` The generator parameters to use for the generation.
- `timeout - [in]` The timeout for the operation.

Returns Upon success, returns *Expected* of std::vector<Speech2Text::SegmentInfo> containing the transcription segments. Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<std::vector<[SegmentInfo](#)>> generate_all_segments (MemoryView audio_buffer, ...)

Generates transcription segments from audio data, and returns them as a vector of [SegmentInfo](#).

Note: The default generator parameters are used in this method, as returned by [create_generator_params\(\)](#).

Parameters

- `audio_buffer - [in]` The audio data to process. Must be PCM float32 normalized to [-1.0, 1.0), mono, little-endian, at 16 kHz.
- `timeout - [in]` The timeout for the operation.

Returns Upon success, returns *Expected* of std::vector<Speech2Text::SegmentInfo> containing the transcription segments. Otherwise, returns *Unexpected* of [hailo_status](#) error.

Expected<std::vector<int>> tokenize (const std::string &text)

Tokenizes a given text into a vector of integers representing the tokens.

Parameters `text - [in]` The input text to tokenize.

Returns Upon success, returns *Expected* of vector of integers. Otherwise, returns *Unexpected* of [hailo_status](#) error.

Public Static Functions

static *Expected*<Speech2Text> create (std::shared_ptr<[VDevice](#)> vdevice, const [Speech2TextParams](#) ...)

Creates an [Speech2Text](#) model instance configured with the specified parameters.

Parameters

- `vdevice - [in]` The [VDevice](#) object used to communicate with the Hailo device.
- `speech2text_params - [in]` The [Speech2TextParams](#) object used to configure the [Speech2Text](#) model.

Returns Upon success, returns *Expected* of [Speech2Text](#). Otherwise, returns *Unexpected* of [hailo_status](#) error.

struct SegmentInfo

Holds information about a single segment of transcribed audio.

Public Members

`float32_t start_sec`

Start timestamp of the segment relative to the start of the audio, in seconds

`float32_t end_sec`

End timestamp of the segment relative to the start of the audio, in seconds

`std::string text`

Transcribed text corresponding to this segment

enum `hailort::genai::Speech2TextTask`

The task to perform in the `Speech2Text` model.

Values:

enumerator TRANSCRIBE

enumerator TRANSLATE

class `hailort::genai::Speech2TextGeneratorParams`

The `Speech2TextGeneratorParams` represents the parameters for speech-to-text generation, which can be changed during runtime in each generation.

Public Functions

`hailo_status set_task(Speech2TextTask task)`

Sets the task for the `Speech2Text` model to perform.

Parameters `task` - [in] The task to perform.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status set_language(std::string_view language)`

Sets the language to use in the generation process.

Note: If the language is not set, the model will automatically detect it.

Parameters `language` - [in] The language to use in the generation process. In the format of ISO-639-1 two-letter code, for example: "en", "fr", etc.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`hailo_status set_repetition_penalty(float32_t repetition_penalty)`

Sets the repetition penalty used during the generation process.

Parameters `repetition_penalty` - [in] The repetition penalty value.

Returns Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

`Speech2TextTask task() const`

Returns The task for the `Speech2Text` model to perform.

`std::string_view language() const`

Note: An empty string means that the model will automatically detect the language.

Returns The language to use in the generation process.

float32_t repetition_penalty() const

Returns The repetition penalty for the generation process.

17. HailoRT Python API Reference

17.1. `hailo_platform.pyhailort.pyhailort`

```
class hailo_platform.pyhailort.pyhailort.HailoRTException
    Bases: Exception

class hailo_platform.pyhailort.pyhailort.InvalidProtocolVersionException
    Bases: hailo\_platform.pyhailort.\_pyhailort.HailoRTException

class
hailo_platform.pyhailort.pyhailort.HailoRTFirmwareControlFailedException
    Bases: hailo\_platform.pyhailort.\_pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTInvalidFrameException
    Bases: hailo\_platform.pyhailort.\_pyhailort.HailoRTException

class
hailo_platform.pyhailort.pyhailort.HailoRTUnsupportedOpcodeException
    Bases: hailo\_platform.pyhailort.\_pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTTimeout
    Bases: hailo\_platform.pyhailort.\_pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTStreamAborted
    Bases: hailo\_platform.pyhailort.\_pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTStreamAbortedByUser
    Bases: hailo\_platform.pyhailort.\_pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HEF(hef_source)
    Bases: object

    Python representation of the Hailo Executable Format, which contains one or more compiled models.

    __init__(hef_source)
        Constructor for the HEF class.

        Parameters hef_source (str or bytes) - The source from which the HEF object will be created. If the source type is str, it is treated as a path to an hef file. If the source type is bytes, it is treated as a buffer. Any other type will raise a ValueError.

    get_networks_names(network_group_name=None)
        Gets the names of all networks in a specific network group.

        Parameters network_group_name (str, optional) - The name of the network group to access. If not given, first network_group is addressed.

        Returns The names of the networks.

        Return type list of str

    property path
        HEF file path.

    get_network_group_names()
        Get the names of the network groups in this HEF.

    get_network_groups_infos()
        Get information about the network groups in this HEF.
```

`get_input_vstream_infos(name=None)`

Get input vstreams information.

Parameters name (str, optional) - The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with all the information objects of all input vstreams.

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

`get_output_vstream_infos(name=None)`

Get output vstreams information.

Parameters name (str, optional) - The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with all the information objects of all output vstreams

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

`get_all_vstream_infos(name=None)`

Get input and output vstreams information.

Parameters name (str, optional) - The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with all the information objects of all input and output vstreams

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

`get_input_stream_infos(name=None)`

Get the input low-level streams information.

Parameters name (str, optional) - The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with information objects of all input low-level streams.

Return type List of `hailo_platform.pyhailort._pyhailort.StreamInfo`

`get_output_stream_infos(name=None)`

Get the output low-level streams information of a specific network group.

Parameters name (str, optional) - The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with information objects of all output low-level streams.

Return type List of `hailo_platform.pyhailort._pyhailort.StreamInfo`

`get_all_stream_infos(name=None)`

Get input and output streams information of a specific network group.

Parameters name (str, optional) - The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with all the information objects of all input and output streams

Return type list of `hailo_platform.pyhailort._pyhailort.StreamInfo`

`get_sorted_output_names(network_group_name=None)`

Get the names of the outputs in a network group. The order of names is determined by the SDK. If the network group is not given, the first one is used.

`get_vstream_name_from_original_name(original_name, network_group_name=None)`

Get vstream name from original layer name for a specific network group.

Parameters

- `original_name(str)` - The original layer name.
- `network_group_name(str, optional)` - The name of the network group to access. If not given, first network_group is addressed.

Returns the matching vstream name for the provided original name.

Return type str

`get_original_names_from_vstream_name(vstream_name, network_group_name=None)`

Get original names list from vstream name for a specific network group.

Parameters

- `vstream_name(str)` - The stream name.
- `network_group_name(str, optional)` - The name of the network group to access. If not given, first network_group is addressed.

Returns all the matching original layers names for the provided vstream name.

Return type list of str

`get_vstream_names_from_stream_name(stream_name, network_group_name=None)`

Get vstream names list from their underlying stream name for a specific network group.

Parameters

- `stream_name(str)` - The underlying stream name.
- `network_group_name(str, optional)` - The name of the network group to access. If not given, first network_group is addressed.

Returns All the matching vstream names for the provided stream name.

Return type list of str

`get_stream_names_from_vstream_name(vstream_name, network_group_name=None)`

Get stream name from vstream name for a specific network group.

Parameters

- `vstream_name(str)` - The name of the vstreams.
- `network_group_name(str, optional)` - The name of the network group to access. If not given, first network_group is addressed.

Returns All the underlying streams names for the provided vstream name.

Return type list of str

`class hailo_platform.pyhailort.pyhailort.PcieDeviceInfo(bus, device, func, ...)`

Bases: `hailo_platform.pyhailort._pyhailort.PcieDeviceInfo`

Represents pcie device info, includeing domain, bus, device and function.

`BOARD_LOCATION_HELP_STRING = 'Board location in the format of the command: "lspci -d 1e60: | cut -d\' \' -f1" ([<domain>]:<bus>:<device>.<func>). If not specified the first board is taken.'`

```
__init__(self: hailo_platform.pyhailort._pyhailort.PcieDeviceInfo) → None
classmethod from_string(board_location_str)
    Parse pcie device info BDF from string. The format is [<domain>]:<bus>:<device>. <func>
classmethod argument_type(board_location_str)
    PcieDeviceInfo Argument type for argparse parsers
class hailo_platform.pyhailort.pyhailort.ConfiguredNetwork(configured_network)
    Bases: object
    Represents a network group loaded to the device.
    __init__(configured_network)
    get_networks_names()
    activate(network_group_params=None)
        Activate this network group in order to infer data through it.

        Parameters network_group_params      (hailo_platform.pyhailort._pyhailort.ActivateNetworkGroupParams, optional) – Network group activation params. If not given, default params will be applied,
        Returns Context manager that returns the activated network group.
        Return type ActivatedNetworkContextManager
```

Note: Usage of *activate* when scheduler enabled is deprecated. On this case, this function will return None and print deprecation warning.

```
wait_for_activation(timeout_ms=None)
    Block until activated, or until timeout_ms is passed.

    Parameters timeout_ms(int, optional) – Timeout value in milliseconds to wait for activation. Defaults to HAILO_INFINITE.

    Raises HailoRTTimeout – In case of timeout.

static create_params()
    Create activation params for network_group.

    Returns hailo_platform.pyhailort._pyhailort.ActivateNetworkGroupParams.

property name

get_output_shapes()

get_sorted_output_names()

get_input_vstream_infos(network_name=None)
    Get input vstreams information.

    Parameters network_name(str, optional) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

    Returns with all the information objects of all input vstreams

    Return type list of hailo_platform.pyhailort._pyhailort.VStreamInfo
```

```
get_output_vstream_infos(network_name=None)
```

Get output vstreams information.

Parameters `network_name(str, optional)` - The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with all the information objects of all output vstreams

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

```
get_all_vstream_infos(network_name=None)
```

Get input and output vstreams information.

Parameters `network_name(str, optional)` - The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with all the information objects of all input and output vstreams

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

```
get_input_stream_infos(network_name=None)
```

Get the input low-level streams information of a specific network group.

Parameters `network_name(str, optional)` - The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with information objects of all input low-level streams.

Return type List of `hailo_platform.pyhailort._pyhailort.StreamInfo`

```
get_output_stream_infos(network_name=None)
```

Get the output low-level streams information of a specific network group.

Parameters `network_name(str, optional)` - The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with information objects of all output low-level streams.

Return type List of `hailo_platform.pyhailort._pyhailort.StreamInfo`

```
get_all_stream_infos(network_name=None)
```

Get input and output streams information of a specific network group.

Parameters `network_name(str, optional)` - The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with all the information objects of all input and output streams

Return type list of `hailo_platform.pyhailort._pyhailort.StreamInfo`

```
get_stream_names_from_vstream_name(vstream_name)
```

Get stream name from vstream name for a specific network group.

Parameters `vstream_name(str)` - The name of the vstreams.

Returns All the underlying streams names for the provided vstream name.

Return type list of str

```
get_vstream_names_from_stream_name(stream_name)
```

Get vstream names list from their underlying stream name for a specific network group.

Parameters `stream_name(str)` - The underlying stream name.

Returns All the matching vstream names for the provided stream name.

Return type list of str

```
set_scheduler_timeout(timeout_ms, network_name=None)
```

Sets the maximum time period that may pass before receiving run time from the scheduler.

This will occur providing at least one send request has been sent, there is no minimum requirement for send requests, (e.g. threshold - see [ConfiguredNetwork.set_scheduler_threshold\(\)](#)).

Parameters timeout_ms (int) - Timeout in milliseconds.

```
set_scheduler_threshold(threshold)
```

Sets the scheduler threshold. This threshold sets the minimum number of send requests required before the network is considered ready to get run time from the scheduler. If at least one send request has been sent, but the threshold is not reached within a set time period (e.g. timeout - see [ConfiguredNetwork.set_scheduler_timeout\(\)](#)), the scheduler will consider the network ready regardless.

Parameters threshold (int) - Threshold in number of frames.

```
set_scheduler_priority(priority)
```

Sets the priority of the network. When the model scheduler will choose the next network, networks with higher priority will be prioritized in the selection. Larger number represents higher priority.

Parameters priority (int) - Priority as a number between HAILO_SCHEDULER_PRIORITY_MIN - HAILO_SCHEDULER_PRIORITY_MAX.

```
init_cache(read_offset)
```

```
update_cache_offset(offset_delta_entries)
```

```
get_cache_ids() → List[int]
```

```
read_cache_buffer(cache_id: int) → bytes
```

```
write_cache_buffer(cache_id: int, buffer: bytes)
```

```
class hailo_platform.pyhailort.pyhailort.ActivatedNetworkContextManager(configured_network, ...)
```

Bases: object

A context manager that returns the activated network group upon enter.

```
__init__(configured_network, activated_network)
```

```
class hailo_platform.pyhailort.pyhailort.ActivatedNetwork(configured_network, ...)
```

Bases: object

The network group that is currently activated for inference.

```
__init__(configured_network, activated_network)
```

```
get_number_of_invalid_frames(clear=True)
```

Returns number of invalid frames.

Parameters clear (bool) - If set, the returned value will be the number of invalid frames read since the last call to this function.

Returns Number of invalid frames.

Return type int

```
validate_all_frames_are_valid()
```

Validates that all of the frames so far are valid (no invalid frames).

```
class hailo_platform.pyhailort.pyhailort.FormatType
Bases: pybind11_builtins.pybind11_object

Data formats accepted by HailoRT.

Members:

    AUTO : Chosen automatically to match the format expected by the device, usually UINT8.

    UINT8

    UINT16

    FLOAT32

    AUTO = <FormatType.AUTO: 0>

    FLOAT32 = <FormatType.FLOAT32: 3>

    UINT16 = <FormatType.UINT16: 2>

    UINT8 = <FormatType.UINT8: 1>

    __init__(self: hailo_platform.pyhailort._pyhailort.FormatType, value: int) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.PowerMeasurementData
Bases: pybind11_builtins.pybind11_object

__init__(*args, **kwargs)

property average_time_value_milliseconds
    float, Average time in milliseconds between samples

property average_value
    float, The average value of the samples that were sampled

equals(self: hailo_platform.pyhailort._pyhailort.PowerMeasurementData, arg0: ...)

property max_value
    float, The maximum value of the samples that were sampled

property min_value
    float, The minimum value of the samples that were sampled

property total_number_of_samples
    uint, The number of samples that were sampled

class hailo_platform.pyhailort.pyhailort.MeasurementBufferIndex
Bases: pybind11_builtins.pybind11_object

Enum-like class representing all FW buffers for power measurements storing.

Members:

    MEASUREMENT_BUFFER_INDEX_0

    MEASUREMENT_BUFFER_INDEX_1

    MEASUREMENT_BUFFER_INDEX_2

    MEASUREMENT_BUFFER_INDEX_3

MEASUREMENT_BUFFER_INDEX_0 =
<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_0: 0>
```

```
MEASUREMENT_BUFFER_INDEX_1 =
<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_1: 1>

MEASUREMENT_BUFFER_INDEX_2 =
<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_2: 2>

MEASUREMENT_BUFFER_INDEX_3 =
<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_3: 3>

__init__(self: hailo_platform.pyhailort._pyhailort.MeasurementBufferIndex, value: int) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.PowerMeasurementTypes
Bases: pybind11_builtins.pybind11_object

Enum-like class representing the different power measurement types. This determines what would be measured on the device.

Members:
    AUTO : Choose the default value according to the supported features.
    SHUNT_VOLTAGE : Measure the shunt voltage. Unit is mV
    BUS_VOLTAGE : Measure the bus voltage. Unit is mV
    POWER : Measure the power. Unit is W
    CURRENT : Measure the current. Unit is mA

    AUTO = <PowerMeasurementTypes.AUTO: 2147483647>
    BUS_VOLTAGE = <PowerMeasurementTypes.BUS_VOLTAGE: 1>
    CURRENT = <PowerMeasurementTypes.CURRENT: 3>
    POWER = <PowerMeasurementTypes.POWER: 2>
    SHUNT_VOLTAGE = <PowerMeasurementTypes.SHUNT_VOLTAGE: 0>

__init__(self: hailo_platform.pyhailort._pyhailort.PowerMeasurementTypes, value: int) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.DvmTypes
Bases: pybind11_builtins.pybind11_object

Enum-like class representing the different DVMs that can be measured. This determines the device that would be measured.

Members:
    AUTO : Choose the default value according to the supported features.
    VDD_CORE : Perform measurements over the core. Exists only in Hailo-8 EVB.
    VDD_IO : Perform measurements over the IO. Exists only in Hailo-8 EVB.
    MIPI_AVDD : Perform measurements over the MIPI avdd. Exists only in Hailo-8 EVB.
    MIPI_AVDD_H : Perform measurements over the MIPI avdd_h. Exists only in Hailo-8 EVB.
    USB_AVDD_IO : Perform measurements over the IO. Exists only in Hailo-8 EVB.
    VDD_TOP : Perform measurements over the top. Exists only in Hailo-8 EVB.
```

USB_AVDD_IO_HV : Perform measurements over the USB_AVDD_IO_HV. Exists only in Hailo-8 EVB.

AVDD_H : Perform measurements over the AVDD_H. Exists only in Hailo-8 EVB.

SDIO_VDD_IO : Perform measurements over the SDIO_VDDIO. Exists only in Hailo-8 EVB.

OVERCURRENT_PROTECTION : Perform measurements over the OVERCURRENT_PROTECTION dvm. Exists only for Hailo-8 platforms supporting current monitoring (such as M.2 and mPCIe).

```
AUTO = <DvmTypes.AUTO: 2147483647>
AVDD_H = <DvmTypes.AVDD_H: 7>
MIPI_AVDD = <DvmTypes.MIPI_AVDD: 2>
MIPI_AVDD_H = <DvmTypes.MIPI_AVDD_H: 3>
OVERCURRENT_PROTECTION = <DvmTypes.OVERCURRENT_PROTECTION: 9>
SDIO_VDD_IO = <DvmTypes.SDIO_VDD_IO: 8>
USB_AVDD_IO = <DvmTypes.USB_AVDD_IO: 4>
USB_AVDD_IO_HV = <DvmTypes.USB_AVDD_IO_HV: 6>
VDD_CORE = <DvmTypes.VDD_CORE: 0>
VDD_IO = <DvmTypes.VDD_IO: 1>
VDD_TOP = <DvmTypes.VDD_TOP: 5>
__init__(self: hailo_platform.pyhailort._pyhailort.DvmTypes, value: int) → None
property name
property value

class hailo_platform.pyhailort.pyhailort.SamplingPeriod
Bases: pybind11_builtins.pybind11_object
Enum-like class representing all bit options and related conversion times for each bit setting for Bus Voltage and Shunt Voltage.

Members:
PERIOD_140us : The sensor provides a new sampling every 140us.
PERIOD_204us : The sensor provides a new sampling every 204us.
PERIOD_332us : The sensor provides a new sampling every 332us.
PERIOD_588us : The sensor provides a new sampling every 588us.
PERIOD_1100us : The sensor provides a new sampling every 1100us.
PERIOD_2116us : The sensor provides a new sampling every 2116us.
PERIOD_4156us : The sensor provides a new sampling every 4156us.
PERIOD_8244us : The sensor provides a new sampling every 8244us.
PERIOD_1100us = <SamplingPeriod.PERIOD_1100us: 4>
PERIOD_140us = <SamplingPeriod.PERIOD_140us: 0>
PERIOD_204us = <SamplingPeriod.PERIOD_204us: 1>
PERIOD_2116us = <SamplingPeriod.PERIOD_2116us: 5>
```

```
PERIOD_332us = <SamplingPeriod.PERIOD_332us: 2>
PERIOD_4156us = <SamplingPeriod.PERIOD_4156us: 6>
PERIOD_588us = <SamplingPeriod.PERIOD_588us: 3>
PERIOD_8244us = <SamplingPeriod.PERIOD_8244us: 7>

__init__(self: hailo_platform.pyhailort._pyhailort.SamplingPeriod, value: int) → None
property name
property value

class hailo_platform.pyhailort.pyhailort.AveragingFactor
    Bases: pybind11_builtins.pybind11_object
    Enum-like class representing all the AVG bit settings and related number of averages for each bit setting.

    Members:
        AVERAGE_1 : Each sample reflects a value of 1 sub-samples.
        AVERAGE_4 : Each sample reflects a value of 4 sub-samples.
        AVERAGE_16 : Each sample reflects a value of 16 sub-samples.
        AVERAGE_64 : Each sample reflects a value of 64 sub-samples.
        AVERAGE_128 : Each sample reflects a value of 128 sub-samples.
        AVERAGE_256 : Each sample reflects a value of 256 sub-samples.
        AVERAGE_512 : Each sample reflects a value of 512 sub-samples.
        AVERAGE_1024 : Each sample reflects a value of 1024 sub-samples.

        AVERAGE_1 = <AveragingFactor.AVERAGE_1: 0>
        AVERAGE_1024 = <AveragingFactor.AVERAGE_1024: 7>
        AVERAGE_128 = <AveragingFactor.AVERAGE_128: 4>
        AVERAGE_16 = <AveragingFactor.AVERAGE_16: 2>
        AVERAGE_256 = <AveragingFactor.AVERAGE_256: 5>
        AVERAGE_4 = <AveragingFactor.AVERAGE_4: 1>
        AVERAGE_512 = <AveragingFactor.AVERAGE_512: 6>
        AVERAGE_64 = <AveragingFactor.AVERAGE_64: 3>

        __init__(self: hailo_platform.pyhailort._pyhailort.AveragingFactor, value: int) → None
        property name
        property value

class hailo_platform.pyhailort.pyhailort.Endianness
    Bases: pybind11_builtins.pybind11_object
    Members:
        BIG_ENDIAN
        LITTLE_ENDIAN
        BIG_ENDIAN = <Endianness.BIG_ENDIAN: 0>
```

```
LITTLE_ENDIAN = <Endianness.LITTLE_ENDIAN: 1>

__init__(self: hailo_platform.pyhailort._pyhailort.Endianness, value: int) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.InputVStreamParams
Bases: object

Parameters of an input virtual stream (host to device).

static make(configured_network, quantized=None, format_type=None, timeout_ms=None, ...)

Create input virtual stream params from a configured network group. These params determine the format of the data that will be fed into the network group.
```

Parameters

- `configured_network` ([ConfiguredNetwork](#)) – The configured network group for which the params are created.
- `quantized` – Unused.
- `format_type` ([FormatType](#)) – The default format type of the data for all input virtual streams. The default is [AUTO](#), which means the data is fed in the same format expected by the device (usually uint8).
- `timeout_ms(int)` – The default timeout in milliseconds for all input virtual streams. Defaults to [DEFAULT_VSTREAM_TIMEOUT_MS](#). In case of timeout, [HailoRTTimeout](#) will be raised.
- `queue_size (int)` – The pipeline queue size. Defaults to [DEFAULT_VSTREAM_QUEUE_SIZE](#).
- `network_name (str)` – Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns The created virtual streams params. The keys are the vstreams names. The values are the params.

Return type dict

```
static make_from_network_group(configured_network, quantized=None, format_type=None, ...)

Create input virtual stream params from a configured network group. These params determine the format of the data that will be fed into the network group.
```

Parameters

- `configured_network` ([ConfiguredNetwork](#)) – The configured network group for which the params are created.
- `quantized` – Unused.
- `format_type` ([FormatType](#)) – The default format type of the data for all input virtual streams. The default is [AUTO](#), which means the data is fed in the same format expected by the device (usually uint8).
- `timeout_ms(int)` – The default timeout in milliseconds for all input virtual streams. Defaults to [DEFAULT_VSTREAM_TIMEOUT_MS](#). In case of timeout, [HailoRTTimeout](#) will be raised.
- `queue_size (int)` – The pipeline queue size. Defaults to [DEFAULT_VSTREAM_QUEUE_SIZE](#).
- `network_name (str)` – Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns The created virtual streams params. The keys are the vstreams names. The values are the params.

Return type dict

```
class hailo_platform.pyhailort.pyhailort.OutputVStreamParams
```

Bases: object

Parameters of an output virtual stream (device to host).

```
static make(configured_network, quantized=None, format_type=None, timeout_ms=None, ...)
```

Create output virtual stream params from a configured network group. These params determine the format of the data that will be returned from the network group.

Parameters

- `configured_network` ([ConfiguredNetwork](#)) – The configured network group for which the params are created.
- `quantized` – Unused.
- `format_type` ([FormatType](#)) – The default format type of the data for all output virtual streams. The default is [AUTO](#), which means the returned data is in the same format returned from the device (usually uint8).
- `timeout_ms(int)` – The default timeout in milliseconds for all output virtual streams. Defaults to `DEFAULT_VSTREAM_TIMEOUT_MS`. In case of timeout, [HailoRTTimeout](#) will be raised.
- `queue_size (int)` – The pipeline queue size. Defaults to `DEFAULT_VSTREAM_QUEUE_SIZE`.
- `network_name (str)` – Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns The created virtual streams params. The keys are the vstreams names. The values are the params.

Return type dict

```
static make_from_network_group(configured_network, quantized=None, format_type=None, ...)
```

Create output virtual stream params from a configured network group. These params determine the format of the data that will be returned from the network group.

Parameters

- `configured_network` ([ConfiguredNetwork](#)) – The configured network group for which the params are created.
- `quantized` – Unused.
- `format_type` ([FormatType](#)) – The default format type of the data for all output virtual streams. The default is [AUTO](#), which means the returned data is in the same format returned from the device (usually uint8).
- `timeout_ms(int)` – The default timeout in milliseconds for all output virtual streams. Defaults to `DEFAULT_VSTREAM_TIMEOUT_MS`. In case of timeout, [HailoRTTimeout](#) will be raised.
- `queue_size (int)` – The pipeline queue size. Defaults to `DEFAULT_VSTREAM_QUEUE_SIZE`.
- `network_name (str)` – Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns The created virtual streams params. The keys are the vstreams names. The values are the params.

Return type dict

```
static make_groups(configured_network, quantized=None, format_type=None, timeout_ms=None, ...)
```

Create output virtual stream params from a configured network group. These params determine the format of the data that will be returned from the network group. The params groups are splitted with respect to their underlying streams for multi process usges.

Parameters

- `configured_network` ([ConfiguredNetwork](#)) – The configured network group for which the params are created.
- `quantized` – Unused.
- `format_type` ([FormatType](#)) – The default format type of the data for all output virtual streams. The default is [AUTO](#), which means the returned data is in the same format returned from the device (usually uint8).
- `timeout_ms(int)` – The default timeout in milliseconds for all output virtual streams. Defaults to [DEFAULT_VSTREAM_TIMEOUT_MS](#). In case of timeout, [HailoRTTimeout](#) will be raised.
- `queue_size (int)` – The pipeline queue size. Defaults to [DEFAULT_VSTREAM_QUEUE_SIZE](#).

Returns Each element in the list represent a group of params, where the keys are the vstreams names, and the values are the params. The params groups are splitted with respect to their underlying streams for multi process usges.

Return type list of dicts

```
class hailo_platform.pyhailort.pyhailort.InputVStreams(configured_network, ...)
```

Bases: object

Input vstreams pipelines that allows to send data, to be used as a context manager.

```
__init__(configured_network, input_vstreams_params)
```

Constructor for the InputVStreams class.

Parameters

- `configured_network` ([ConfiguredNetwork](#)) – The configured network group for which the pipeline is created.
- `input_vstreams_params` (dict from str to [InputVStreamParams](#)) – Params for the input vstreams in the pipeline.

```
get(name=None)
```

Return a single input vstream by its name.

Parameters `name (str)` – The vstream name. If `name=None` and there is a single input vstream, that single ([InputVStream](#)) will be returned. Otherwise, if `name=None` and there are multiple input vstreams, an exception will be thrown.

Returns The ([InputVStream](#)) that corresponds to the given name.

Return type [InputVStream](#)

```
clear()
```

Clears the vstreams' pipeline buffers.

```
class hailo_platform.pyhailort.pyhailort.OutputVStreams(configured_network, ...)
```

Bases: object

Output virtual streams pipelines that allows to receive data, to be used as a context manager.

```
__init__(configured_network, output_vstreams_params, tf_nms_format=False)
```

Constructor for the OutputVStreams class.

Parameters

- `configured_network` ([ConfiguredNetwork](#)) - The configured network group for which the pipeline is created.
- `output_vstreams_params` (dict from str to [OutputVStreamParams](#)) - Params for the output vstreams in the pipeline.
- `tf_nms_format` (bool, optional) - indicates whether the returned nms outputs should be in Hailo format or TensorFlow format. Default is False (using Hailo format).
 - Hailo format - list of `numpy.ndarray`. Each element represents the detections (bboxes) for the class, and its shape is `[number_of_detections, BBOX_PARAMS]`
 - TensorFlow format - `numpy.ndarray` of shape `[class_count, BBOX_PARAMS, detections_count]` padded with empty bboxes.

`get(name=None)`

Return a single output vstream by its name.

Parameters `name` (str) - The vstream name. If `name=None` and there is a single output vstream, that single ([OutputVStream](#)) will be returned. Otherwise, if `name=None` and there are multiple output vstreams, an exception will be thrown.

Returns The ([OutputVStream](#)) that corresponds to the given name.

Return type [OutputVStream](#)

`clear()`

Clears the vstreams' pipeline buffers.

`class hailo_platform.pyhailort.pyhailort.InputVStream(send_object)`

Bases: object

Represents a single virtual stream in the host to device direction.

`__init__(send_object)`

`property shape`

`property dtype`

`property name`

`property network_name`

`send(input_data)`

Send frames to inference.

Parameters `input_data` (`numpy.ndarray`) - Data to run inference on.

`flush()`

Blocks until there are no buffers in the input VStream pipeline.

`property info`

`class hailo_platform.pyhailort.pyhailort.OutputVStream(configured_network, ...)`

Bases: object

Represents a single output virtual stream in the device to host direction.

`__init__(configured_network, recv_object, name, tf_nms_format=False, net_group_name="")`

`property output_order`

`property shape`

```
property dtype  
property name  
property network_name  
recv()  
Receive frames after inference.
```

Returns The output of the inference for a single frame. The returned tensor does not include the batch dimension. In case of nms output and tf_nms_format=False, returns list of numpy.ndarray.

Return type numpy.ndarray

```
property info
```

```
set_nms_score_threshold(threshold)
```

Set NMS score threshold, used for filtering out candidates. Any box with score<TH is suppressed.

Parameters *threshold*(float) - NMS score threshold to set.

Note: This function will fail in cases where the output vstream has no NMS operations on the CPU.

```
set_nms_iou_threshold(threshold)
```

Set NMS intersection over union overlap Threshold, used in the NMS iterative elimination process where potential duplicates of detected items are suppressed.

Parameters *threshold*(float) - NMS IoU threshold to set.

Note: This function will fail in cases where the output vstream has no NMS operations on the CPU.

```
set_nms_max_proposals_per_class(max_proposals_per_class)
```

Set a limit for the maximum number of boxes per class.

Parameters *max_proposals_per_class*(int) - NMS max proposals per class to set.

Note: This function will fail in cases where the output vstream has no NMS operations on the CPU.

```
set_nms_max_accumulated_mask_size(max_accumulated_mask_size)
```

Set maximum accumulated mask size for all the detections in a frame. Used in order to change the output buffer frame size, in cases where the output buffer is too small for all the segmentation detections.

Parameters *max_accumulated_mask_size*(int) - NMS max accumulated mask size.

Note: This function must be called before starting inference! This function will fail in cases where there is no output with NMS operations on the CPU.

```
class hailo_platform.pyhailort.pyhailort.InferVStreams(configured_net_group, ...)
```

Bases: object

Pipeline that allows to call blocking inference, to be used as a context manager.

```
__init__(configured_net_group, input_vstreams_params, output_vstreams_params, tf_nms_format=False)
```

Constructor for the InferVStreams class.

Parameters

- `configured_net_group` ([ConfiguredNetwork](#)) – The configured network group for which the pipeline is created.
- `input_vstreams_params` (dict from str to [InputVStreamParams](#)) – Params for the input vstreams in the pipeline. Only members of this dict will take part in the inference.
- `output_vstreams_params` (dict from str to [OutputVStreamParams](#)) – Params for the output vstreams in the pipeline. Only members of this dict will take part in the inference.
- `tf_nms_format` (bool, optional) – indicates whether the returned nms outputs should be in Hailo format or TensorFlow format. Default is False (using Hailo format).
 - Hailo format – list of `numpy.ndarray`. Each element represents the detections (bboxes) for the class, and its shape is [number_of_detections, BBOX_PARAMS]
 - TensorFlow format – `numpy.ndarray` of shape [class_count, BBOX_PARAMS, detections_count] padded with empty bboxes.

```
infer(input_data)
```

Run inference on the hardware device.

Parameters `input_data` (dict of `numpy.ndarray`) – Where the key is the name of the input_layer, and the value is the data to run inference on.

Returns Output tensors of all output layers. The keys are outputs names and the values are output data tensors as `numpy.ndarray` (or list of `numpy.ndarray` in case of nms output and `tf_nms_format=False`).

Return type dict

```
get_hw_time()
```

Get the hardware device operation time it took to run inference over the last batch.

Returns Time in seconds.

Return type float

```
get_total_time()
```

Get the total time it took to run inference over the last batch.

Returns Time in seconds.

Return type float

```
set_nms_score_threshold(threshold)
```

Set NMS score threshold, used for filtering out candidates. Any box with score<TH is suppressed.

Parameters `threshold` (float) – NMS score threshold to set.

Note: This function will fail in cases where there is no output with NMS operations on the CPU.

```
set_nms_iou_threshold(threshold)
```

Set NMS intersection over union overlap Threshold, used in the NMS iterative elimination process where potential duplicates of detected items are suppressed.

Parameters `threshold` (float) – NMS IoU threshold to set.

Note: This function will fail in cases where there is no output with NMS operations on the CPU.

`set_nms_max_proposals_per_class(max_proposals_per_class)`

Set a limit for the maximum number of boxes per class.

Parameters `max_proposals_per_class(int)` - NMS max proposals per class to set.

Note: This function must be called before starting inference! This function will fail in cases where there is no output with NMS operations on the CPU.

`set_nms_max_accumulated_mask_size(max_accumulated_mask_size)`

Set maximum accumulated mask size for all the detections in a frame. Used in order to change the output buffer frame size, in cases where the output buffer is too small for all the segmentation detections.

Parameters `max_accumulated_mask_size(int)` - NMS max accumulated mask size.

Note: This function must be called before starting inference! This function will fail in cases where there is no output with NMS operations on the CPU.

`class hailo_platform.pyhailort.pyhailort.BoardInformation(protocol_version, ...)`

Bases: `object`

`__init__(protocol_version, fw_version_major, fw_version_minor, fw_version_revision, logger_version, ...)`

`static get_hw_arch_str(device_arch)`

`class hailo_platform.pyhailort.pyhailort.CoreInformation(fw_version_major, ...)`

Bases: `object`

`__init__(fw_version_major, fw_version_minor, fw_version_revision, is_release, extended_context_switch_buffer)`

`class hailo_platform.pyhailort.pyhailort.ExtendedDeviceInformation(neural_network_core_clock_rate, ...)`

Bases: `object`

`__init__(neural_network_core_clock_rate, supported_features, boot_source, lcs, soc_id, eth_mac_address, ...)`

`class hailo_platform.pyhailort.pyhailort.TemperatureInfo`

Bases: `pybind11_builtins.pybind11_object`

`__init__(*args, **kwargs)`

`equals(self: hailo_platform.pyhailort._pyhailort.TemperatureInfo, arg0: ...)`

`property sample_count`

`property ts0_temperature`

`property ts1_temperature`

`class hailo_platform.pyhailort.pyhailort.Control(device: ...)`

Bases: `object`

The control object of this device, which implements the control API of the Hailo device. Should be used only from `Device.control`

```
WORD_SIZE = 4

__init__(device: hailo_platform.pyhailort._pyhailort.Device)

property device_id
    Getter for the device_id.

Returns A string ID of the device. BDF for PCIe devices, IP address for Ethernet devices, "Integrated" for integrated nnc devices.

Return type str

open()
    Initializes the resources needed for using a control device.

close()
    Releases the resources that were allocated for the control device.

chip_reset()
    Resets the device (chip reset).

nn_core_reset()
    Resets the nn_core.

soft_reset()
    reloads the device firmware (soft reset)

forced_soft_reset()
    reloads the device firmware (forced soft reset)

read_memory(address, data_length)
    Reads memory from the Hailo chip. Byte order isn't changed. The core uses little-endian byte order.
```

Parameters

- `address (int)` – Physical address to read from.
- `data_length (int)` – Size to read in bytes.

Returns Memory read from the chip, each index in the list is a byte

Return type list of str

```
write_memory(address, data_buffer)
    Write memory to Hailo chip. Byte order isn't changed. The core uses little-endian byte order.
```

Parameters

- `address (int)` – Physical address to write to.
- `data_buffer (list of str)` – Data to write.

```
power_measurement(dvm=<DvmTypes.AUTO: 2147483647>, ...)
```

Perform a single power measurement on an Hailo chip. It works with the default settings where the sensor returns a new value every 2.2 ms without averaging the values.

Parameters

- `dvm (DvmTypes)` – Which DVM will be measured. Default (`AUTO`) will be different according to the board:

Default (`AUTO`) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums `VDD_CORE`, `MIPI_AVDD` and `AVDD_H`. Only `POWER` can be measured with this option.

Default (`AUTO`) for platforms supporting current monitoring (such as M.2 and mPCIe): `OVERRCURRENT_PROTECTION`

- `measurement_type` - ([PowerMeasurementTypes](#)): The type of the measurement.

Returns

The measured power.

For [PowerMeasurementTypes](#):

- [SHUNT_VOLTAGE](#): Unit is mV.
- [BUS_VOLTAGE](#): Unit is mV.
- [POWER](#): Unit is W.
- [CURRENT](#): Unit is mA.

Return type float

Note: This function can perform measurements for more than just power. For all supported measurement types, please look at [PowerMeasurementTypes](#).

`start_power_measurement (averaging_factor=<AveragingFactor.AVERAGE_256: 5>, ...)`

Start performing a long power measurement.

Parameters

- `averaging_factor` ([AveragingFactor](#)) - Number of samples per time period, sensor configuration value.
- `sampling_period` ([SamplingPeriod](#)) - Related conversion time, sensor configuration value. The sensor samples the power every `sampling_period` [ms] and averages every `averaging_factor` samples. The sensor provides a new value every: $2 * \text{sampling_period} * \text{averaging_factor}$ [ms]. The firmware wakes up every `delay` [ms] and checks the sensor. If there is a new value to read from the sensor, the firmware reads it. Note that the average calculated by the firmware is "average of averages", because it averages values that have already been averaged by the sensor.

`stop_power_measurement()`

Stop performing a long power measurement. Calling the function eliminates the start function settings for the averaging the samples, and returns to the default values, so the sensor will return a new value every 2.2 ms without averaging values.

`set_power_measurement (buffer_index=<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_0: ...)`

Set parameters for long power measurement on an Hailo chip.

Parameters

- `buffer_index` ([MeasurementBufferIndex](#)) - Index of the buffer on the firmware the data would be saved at. Default is [MEASUREMENT_BUFFER_INDEX_0](#)
- `dvm` ([DvmTypes](#)) - Which DVM will be measured. Default ([AUTO](#)) will be different according to the board:

Default ([AUTO](#)) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums [VDD_CORE](#), [MIPI_AVDD](#) and [AVDD_H](#). Only [POWER](#) can be measured with this option.

Default ([AUTO](#)) for platforms supporting current monitoring (such as M.2 and mPCIe): [OVERCURRENT_PROTECTION](#)

- `measurement_type` - ([PowerMeasurementTypes](#)): The type of the measurement.

Note: This function can perform measurements for more than just power. For all supported measurement types view [PowerMeasurementTypes](#)

`get_power_measurement(buffer_index=<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_0: ...>)`

Read measured power from a long power measurement

Parameters

- `buffer_index` ([MeasurementBufferIndex](#)) – deprecated.
- `should_clear` (bool) – Flag indicating if the results saved at the firmware will be deleted after reading.

Returns

Object containing measurement data

For [PowerMeasurementTypes](#):

- `SHUNT_VOLTAGE`: Unit is mV.
- `BUS_VOLTAGE`: Unit is mV.
- `POWER`: Unit is W.
- `CURRENT`: Unit is mA.

Return type [PowerMeasurementData](#)

Note: This function can perform measurements for more than just power. For all supported measurement types view [PowerMeasurementTypes](#).

`read_user_config()`

Read the user configuration section as binary data.

Returns User config as a binary buffer.

Return type str

`write_user_config(configuration)`

Write the user configuration.

Parameters `configuration(str)` – A binary representation of a Hailo device configuration.

`read_board_config()`

Read the board configuration section as binary data.

Returns Board config as a binary buffer.

Return type str

`write_board_config(configuration)`

Write the static configuration.

Parameters `configuration(str)` – A binary representation of a Hailo device configuration.

`identify()`

Gets the Hailo chip identification.

Returns [BoardInformation](#)

`core_identify()`

Gets the Core Hailo chip identification.

Returns `CoreInformation`

`set_fw_logger(level, interface_mask)`

Configure logger level and interface of sending.

Parameters

- `level (FwLoggerLevel)` - The minimum logger level.
- `interface_mask (int)` - Output interfaces (mix of FwLoggerInterface).

`set_throttling_state(should_activate)`

Change throttling state of temperature protection and overcurrent protection components. In case that change throttling state of temperature protection didn't succeed, the change throttling state of overcurrent protection is executed.

Parameters `should_activate (bool)` - Should be true to enable or false to disable.

`get_throttling_state()`

Get the current throttling state of temperature protection and overcurrent protection components. If any throttling is enabled, the function return true.

Returns true if temperature or overcurrent protection throttling is enabled, false otherwise.

Return type bool

`read_register(address)`

Read the value of a register from a given address.

Parameters `address (int)` - Address to read register from.

Returns Value of the register

Return type int

`set_bit(address, bit_index)`

Set (turn on) a specific bit at a register from a given address.

Parameters

- `address (int)` - Address of the register to modify.
- `bit_index (int)` - Index of the bit that would be set.

`reset_bit(address, bit_index)`

Reset (turn off) a specific bit at a register from a given address.

Parameters

- `address (int)` - Address of the register to modify.
- `bit_index (int)` - Index of the bit that would be reset.

`firmware_update(firmware_binary, should_reset=True)`

Update firmware binary on the flash.

Parameters

- `firmware_binary (bytes)` - firmware binary stream.
- `should_reset (bool)` - Should a reset be performed after the update (to load the new firmware)

`second_stage_update(second_stage_binary)`

Update second stage binary on the flash

Parameters `second_stage_binary (bytes)` – second stage binary stream.

`store_sensor_config(section_index, reset_data_size, sensor_type, config_file_path, config_height=0, ...)`

Store sensor configuration to Hailo chip flash memory.

Parameters

- `section_index (int)` – Flash section index to write to. [0-6]
- `reset_data_size (int)` – Size of reset configuration.
- `sensor_type (SensorConfigTypes)` – Sensor type.
- `config_file_path (str)` – Sensor configuration file path.
- `config_height (int)` – Configuration resolution height.
- `config_width (int)` – Configuration resolution width.
- `config_fps (int)` – Configuration FPS.
- `config_name (str)` – Sensor configuration name.

`store_isp_config(reset_config_size, isp_static_config_file_path, isp_runtime_config_file_path, ...)`

Store sensor isp configuration to Hailo chip flash memory.

Parameters

- `reset_config_size (int)` – Size of reset configuration.
- `isp_static_config_file_path (str)` – Sensor isp static configuration file path.
- `isp_runtime_config_file_path (str)` – Sensor isp runtime configuration file path.
- `config_height (int)` – Configuration resolution height.
- `config_width (int)` – Configuration resolution width.
- `config_fps (int)` – Configuration FPS.
- `config_name (str)` – Sensor configuration name.

`get_sensor_sections_info()`

Get sensor sections info from Hailo chip flash memory.

Returns Sensor sections info read from the chip flash memory.

`sensor_set_generic_i2c_slave(slave_address, register_address_size, bus_index, ...)`

Set a generic I2C slave for sensor usage.

Parameters

- `sequence (int)` – Request/response sequence.
- `slave_address (int)` – The address of the I2C slave.
- `register_address_size (int)` – The size of the offset (in bytes).
- `bus_index (int)` – The number of the bus the I2C slave is behind.
- `should_hold_bus (bool)` – Hold the bus during the read.
- `endianness (Endianness)` – Big or little endian.

`set_sensor_i2c_bus_index(sensor_type, i2c_bus_index)`

Set the I2C bus to which the sensor of the specified type is connected.

Parameters

- `sensor_type (SensorConfigTypes)` – The sensor type.
- `i2c_bus_index (int)` – The I2C bus index of the sensor.

`load_and_start_sensor(section_index)`

Load the configuration with I2C in the section index.

Parameters `section_index (int)` – Flash section index to load config from. [0-6]

`reset_sensor(section_index)`

Reset the sensor that is related to the section index config.

Parameters `section_index (int)` – Flash section index to reset. [0-6]

`wd_enable(cpu_id)`

Enable firmware watchdog.

Parameters `cpu_id (HailoCpuId)` – 0 for App CPU, 1 for Core CPU.

`wd_disable(cpu_id)`

Disable firmware watchdog.

Parameters `cpu_id (HailoCpuId)` – 0 for App CPU, 1 for Core CPU.

`wd_config(cpu_id, wd_cycles, wd_mode)`

Configure a firmware watchdog.

Parameters

- `cpu_id (HailoCpuId)` – 0 for App CPU, 1 for Core CPU.
- `wd_cycles (int)` – number of cycles until watchdog is triggered.
- `wd_mode (int)` – 0 - HW/SW mode, 1 - HW only mode

`previous_system_state(cpu_id)`

Read the FW previous system state.

Parameters `cpu_id (HailoCpuId)` – 0 for App CPU, 1 for Core CPU.

`get_chip_temperature()`

Returns the latest temperature measurements from the two internal temperature sensors of the Hailo chip.

Returns Temperature in celsius of the two internal temperature sensors (TS), and a sample count (a running 16-bit counter)

Return type `TemperatureInfo`

`get_extended_device_information()`

Returns extended information about the device

Return type `ExtendedDeviceInformation`

`set_pause_frames(rx_pause_frames_enable)`

Enable/Disable Pause frames.

Parameters `rx_pause_frames_enable (bool)` – False for disable, True for enable.

`test_chip_memories()`

test all chip memories using smart BIST

```
set_notification_callback(callback_func, notification_id, opaque)
```

Set a callback function to be called when a notification is received.

Parameters

- **callback_func** (*function*) – Callback function with the parameters (device, notification, opaque). Note that throwing exceptions is not supported and will cause the program to terminate with an error!
- **notification_id** (*NotificationId*) – Notification ID to register the callback to.
- **opaque** (*object*) – User defined data.

Note: The notifications thread is started and closed in the `use_device()` context, so notifications can only be received there.

```
remove_notification_callback(notification_id)
```

Remove a notification callback which was already set.

Parameters *notification_id* (*NotificationId*) – Notification ID to remove the callback from.

```
class hailo_platform.pyhailort.pyhailort.Device(device_id=None)
```

Bases: *object*

Hailo device object representation (for inference use `VDevice`)

```
classmethod scan()
```

Scans for all devices on the system.

Returns list of str, device ids.

```
__init__(device_id=None)
```

Create the Hailo device object.

Parameters *device_id* (*str*) – Device id string, can represent several device types: [-] for pcie devices - pcie bdf (XXXX:XX:XX.X) [-] for ethernet devices - ip address (xxx.xxx.xxx.xxx)

```
release()
```

Release the allocated resources of the device. This function should be called when working with the device not as context-manager.

```
property device_id
```

Getter for the `device_id`.

Returns A string ID of the device. BDF for PCIe devices, IP address for Ethernet devices, "Integrated" for integrated nnc devices.

Return type str

```
property control
```

Returns the control object of this device, which implements the control API of the Hailo device.

Return type [Control](#)

Attention: Use the low level control API with care.

```
property loaded_network_groups
```

Getter for the property `_loaded_network_groups`.

Returns List of the the configured network groups loaded on the device.

Return type list of [ConfiguredNetwork](#)

```
class hailo_platform.pyhailort.pyhailort.VDevice(params=None, *, device_ids=None)
```

Bases: object

Hailo virtual device representation.

```
__init__(params=None, *, device_ids=None)
```

Create the Hailo virtual device object.

Parameters

- `params` (`hailo_platform.pyhailort.pyhailort.VDeviceParams`, optional) – VDevice params, call [VDevice.create_params\(\)](#) to get default params. Excludes 'device_ids'.
- `device_ids` (list of str, optional) – devices ids to create VDevice from, call [Device.scan\(\)](#) to get list of all available devices. Excludes 'params'. Cannot be used together with `device_id`.

```
release()
```

Release the allocated resources of the device. This function should be called when working with the device not as context-manager.

```
static create_params()
```

```
configure(hef, configure_params_by_name={})
```

Configures target vdevice from HEF object.

Parameters

- `hef` ([HEF](#)) – HEF to configure the vdevice from
- `configure_params_by_name` (dict, optional) – Maps between each net_group_name to `configure_params`. If not provided, default params will be applied

```
get_physical_devices()
```

Gets the underlying physical devices.

Returns The underlying physical devices.

Return type list of [Device](#)

```
get_physical_devices_ids()
```

Gets the physical devices ids.

Returns The underlying physical devices infos.

Return type list of str

```
create_infer_model(hef_source, name= "")
```

Creates the infer model from an hef.

Parameters

- `hef_source` (str or bytes) – The source from which the HEF object will be created. If the source type is `str`, it is treated as a path to an hef file. If the source type is `bytes`, it is treated as a buffer. Any other type will raise a `ValueError`.
- `name` (str, optional) – The string of the model name.

Returns The infer model object.

Return type [InferModel](#)

Raises [HailoREException](#) – In case the infer model creation failed.

Note: `create_infer_model` must be called from the same process the VDevice is created in, otherwise an [HailoRTException](#) will be raised.

Note: So long as the InferModel object is alive, the VDevice object is alive as well.

property loaded_network_groups

Getter for the property `_loaded_network_groups`.

Returns List of the the configured network groups loaded on the device.

Return type list of [ConfiguredNetwork](#)

`class hailo_platform.pyhailort.pyhailort.InferModel(infer_model, hef_path)`

Bases: `object`

Contains all of the necessary information for configuring the network for inference. This class is used to set up the model for inference and includes methods for setting and getting the model's parameters. By calling the `configure` function, the user can create a [ConfiguredInferModel](#) object, which is used to run inference.

`class InferStream(infer_stream)`

Bases: `object`

Represents the parameters of a stream. In default, the stream's parameters are set to the default values of the model. The user can change the stream's parameters by calling the setter functions.

`__init__(infer_stream)`

property name

Returns: name (str): the name of the edge.

property shape

Returns: shape (list[int]): the shape of the edge.

property format

Returns: format (`_pyhailort.hailo_format_t`): the format of the edge.

`set_format_type(type)`

Set the format type of the stream.

Parameters `type (_pyhailort.hailo_format_type_t)` - the format type

`set_format_order(order)`

Set the format order of the stream.

Parameters `order (_pyhailort.hailo_format_order_t)` - the format order

property quant_infos

Returns: `quant_infos (list[_pyhailort.hailo_quant_info_t])`: List of the quantization information of the edge.

property is_nms

Returns: `is_nms (bool)`: whether the stream is NMS.

`set_nms_score_threshold(threshold)`

Set NMS score threshold, used for filtering out candidates. Any box with score<TH is suppressed.

Parameters `threshold (float)` - NMS score threshold to set.

Note: This function is invalid in cases where the edge has no NMS operations on the CPU. It will not fail, but make the `configure()` function fail.

```
set_nms_iou_threshold(threshold)
```

Set NMS intersection over union overlap Threshold, used in the NMS iterative elimination process where potential duplicates of detected items are suppressed.

Parameters `threshold` (float) – NMS IoU threshold to set.

Note: This function is invalid in cases where the edge has no NMS operations on the CPU. It will not fail, but make the `configure()` function fail.

```
set_nms_max_proposals_per_class(max_proposals)
```

Set a limit for the maximum number of boxes per class.

Parameters `max_proposals` (int) – NMS max proposals per class to set.

Note: This function is invalid in cases where the edge has no NMS operations on the CPU. It will not fail, but make the `configure()` function fail.

```
set_nms_max_accumulated_mask_size(max_accumulated_mask_size)
```

Set maximum accumulated mask size for all the detections in a frame.

Parameters `max_accumulated_mask_size`(int) – NMS max accumulated mask size.

Note: Used in order to change the output buffer frame size in cases where the output buffer is too small for all the segmentation detections.

Note: This function is invalid in cases where the edge has no NMS operations on the CPU. It will not fail, but make the `configure()` function fail.

```
__init__(infer_model, hef_path)
```

```
property hef
```

Returns: [HEF](#): the HEF object of the model

```
set_batch_size(batch_size)
```

Sets the batch size of the InferModel. This parameter determines the number of frames to be sent for inference in a single batch. If a scheduler is enabled, this parameter determines the ‘burst size’: the max number of frames after which the scheduler will attempt to switch to another model.

Note: The default value is `HAILO_DEFAULT_BATCH_SIZE` - which means the batch is determined by HailoRT automatically.

Parameters `batch_size`(int) – The new batch size to be set.

```
set_power_mode(power_mode)
```

Sets the power mode of the InferModel

Parameters `power_mode` (`_pyhailort.hailo_power_mode_t`) – The power mode to set.

```
configure()
```

Configures the InferModel object. Also checks the validity of the configuration’s formats.

Returns The configured [InferModel](#) object.

Return type `configured_infer_model` ([ConfiguredInferModel](#))

Raises `HailoRTException` - In case the configuration is invalid (example: see `InferStream.set_nms_iou_threshold()`).

Note: A `ConfiguredInferModel` should be used inside a context manager, and should not be passed to a different process.

`property input_names`

Returns: names (list[str]): The input names of the `InferModel`.

`property output_names`

Returns: names (list[str]): The output names of the `InferModel`.

`property inputs`

Returns: inputs (list[`InferStream`]): List of input `InferModel.InferStream`.

`property outputs`

Returns: outputs (list[`InferStream`]): List of output `InferModel.InferStream`.

`input(name=“)`

Gets an input's `InferModel.InferStream`.

Parameters name(str, optional) - the name of the input stream. Required in case of multiple inputs.

Returns `ConfiguredInferModel.Bindings.InferStream` - the input infer stream of the configured infer model.

Raises

- `HailoRTNotFoundException` -
- but multiple inputs exist. -

`output(name=“)`

Gets an output's `InferModel.InferStream`.

Parameters name(str, optional) - the name of the output stream. Required in case of multiple outputs.

Returns `ConfiguredInferModel.Bindings.InferStream` - the output infer stream of the configured infer model.

Raises

- `HailoRTNotFoundException` -
- but multiple outputs exist. -

`class hailo_platform.pyhailort.pyhailort.ConfiguredInferModel(configured_infer_model, ...)`

Bases: object

Configured `InferModel` that can be used to perform an asynchronous inference.

Note: Passing an instance of `ConfiguredInferModel` to a different process is not supported and would lead to an undefined behavior.

`class NmsTransformationInfo(format_order: hailo_platform.pyhailort.pyhailort.FormatOrder, ...)`

Bases: object

class for NMS transformation info.

`format_order: hailo_platform.pyhailort._pyhailort.FormatOrder`

```
    input_height: int
    input_width: int
    number_of_classes: int
    max_bboxes_per_class: int
    quant_info: hailo_platform.pyhailort._pyhailort.QuantInfo
    output_dtype: numpy.dtype = dtype('float32')
    __init__(format_order: hailo_platform.pyhailort._pyhailort.FormatOrder, input_height: int, ...)

class NmsHailoTransformationInfo(format_order: ...)
    Bases:      hailo_platform.pyhailort._pyhailort.ConfiguredInferModel.
    NmsTransformationInfo
    class for NMS transformation info when using hailo format
    use_tf_nms_format: bool = False
    __init__(format_order: hailo_platform.pyhailort._pyhailort.FormatOrder, input_height: int, ...)

class NmsTfTransformationInfo(format_order: ...)
    Bases:      hailo_platform.pyhailort._pyhailort.ConfiguredInferModel.
    NmsTransformationInfo
    class for NMS transformation info when using tf format
    use_tf_nms_format: bool = True
    __init__(format_order: hailo_platform.pyhailort._pyhailort.FormatOrder, input_height: int, ...)

class Bindings(bindings, input_names, output_names, nms_infos)
    Bases: object
    Represents an asynchronous infer request - holds the input and output buffers of the request. A request
    represents a single frame.

    class InferStream(infer_stream, nms_info=None)
        Bases: object
        Holds the input and output buffers of the Bindings infer request
        __init__(infer_stream, nms_info=None)

        set_buffer(buffer)
            Sets the edge's buffer to a new one.
            Parameters buffer (numpy.array) - The new buffer to set. The array's shape
            should match the edge's shape.

        get_buffer(tf_format=False)
            Gets the edge's buffer.
            Parameters tf_format (bool, optional) - Whether the output format is tf or
            hailo. Relevant for NMS outputs. The output can be re-formatted into two formats (TF,
            Hailo) and the user through choosing the True/False function parameter, can decide
            which format to receive. Does not support format order HAILO_NMS_BY_SCORE.

            For detection outputs: TF format is an numpy.array with shape [number of classes,
            bounding box params, max bounding boxes per class] where the 2nd dimension (bounding
            box params) is of a fixed length of 5 (y_min, x_min, y_max, x_max, score).

            hailo HAILO_NMS_BY_CLASS format is a list of numpy.array where each array repre-
            sents the detections for a specific class: [cls0_detections, cls1_detections, ...]. The length
```

of the list is the number of classes. Each `numpy.array` shape is (number of detections, bounding box params) where the 2nd dimension (bounding box params) is of a fixed length of 5 (y_min, x_min, y_max, x_max, score).

hailo HAILO_NMS_BY_SCORE format is a list of detections where each detection is an `Detection` object. The detections are sorted decreasingly by score.

For segmentation outputs: TF format is an `numpy.array` with shape [1, image_size + number_of_params, max bounding boxes per class] where the 2nd dimension (image_size + number_of_params) is calculated as: mask (image_width * image_height) + (y_min, x_min, y_max, x_max, score, class_id). The mask is a binary mask of the segmentation output where the ROI (region of interest) is mapped to 1 and the background is mapped to 0.

Hailo format is a list of detections: [detection0, detection1, ... detection_m] where each detection is an `DetectionWithByteMask`. The detections are sorted decreasingly by score.

Returns the buffer of the edge.

Return type buffer (`numpy.array`)

`__init__(bindings, input_names, output_names, nms_infos)`

`input(name=“)`

Gets an input's InferStream object.

Parameters `name(str, optional)`- the name of the input stream. Required in case of multiple inputs.

Returns `ConfiguredInferModel.Bindings.InferStream`- the input infer stream of the configured infer model.

Raises

- `HailoRTNotFoundException` -
- but multiple inputs exist. -

`output(name=“)`

Gets an output's InferStream object.

Parameters `name(str, optional)`- the name of the output stream. Required in case of multiple outputs.

Returns `ConfiguredInferModel.Bindings.InferStream`- the output infer stream of the configured infer model.

Raises

- `HailoRTNotFoundException` -
- but multiple outputs exist. -

`get()`

Gets the internal bindings object.

Returns the internal bindings object.

Return type `_bindings(_pyhailort.ConfiguredInferModelBindingsWrapper)`

`__init__(configured_infer_model, infer_model)`

`activate()`

Activates hailo device inner-resources for inference. Calling this function is invalid in case scheduler is enabled.

Raises `HailoRTException` -

`deactivate()`

Deactivates hailo device inner-resources for inference. Calling this function is invalid in case scheduler is enabled.

Raises `HailoRTException` -

```
create_bindings(input_buffers=None, output_buffers=None)
```

Creates a Bindings object.

Parameters

- (`dict[str (output_buffers) - numpy.array]`, optional): The input buffers for the Bindings object. Keys are the input names, and values are their corresponding buffers. See [get_buffer\(\)](#) for more information.
- (`dict[str - numpy.array]`, optional): The output buffers for the Bindings object. Keys are the output names, and values are their corresponding buffers. See [get_buffer\(\)](#) for more information.

Returns Bindings object

Return type `ConfiguredInferModel.Bindings`

Raises `HailoRTException` -

```
wait_for_async_ready(timeout_ms=1000, frames_count=1)
```

The readiness of the model to launch is determined by the ability to push buffers to the asynchronous inference pipeline. If the model is ready, the method will return immediately. If it's not ready initially, the method will wait for the model to become ready.

Parameters

- `timeout_ms (int, optional)` - Max amount of time to wait until the model is ready in milliseconds. Defaults to 1000
- `frames_count (int, optional)` - The number of buffers you intend to infer in the next request. Useful for batch inference. Defaults to 1

Note: Calling this function with `frames_count` greater than `ConfiguredInferModel.get_async_queue_size()` will timeout.

Raises

- `HailoRTTimeout` -
- `HailoRTException` -

```
run(bindings, timeout)
```

Launches a synchronous inference operation with the provided bindings.

Parameters

- `bindings (list of)` - The bindings for the inputs and outputs of the model. A list with a single binding is valid. Multiple bindings are useful for batch inference.
- `timeout (int)` - The timeout in milliseconds.

Raises

- `HailoRTException` -
- `HailoRTTimeout` -

```
run_async(bindings, callback=None)
```

Launches an asynchronous inference operation with the provided bindings.

Parameters

- `bindings (list of)` - The bindings for the inputs and outputs of the model. A list with a single binding is valid. Multiple bindings are useful for batch inference.
- `callback(Callable, optional)` - A callback that will be called upon completion of the asynchronous inference operation. The function will be called with an info argument (`AsyncInferCompletionInfo`) holding the information about the async job. If the async job was unsuccessful, the info parameter will hold an exception method

that will raise an exception. The callback must accept a ‘completion_info’ keyword argument

Note: As a standard, callbacks should be executed as quickly as possible. In case of an error, the pipeline will be shut down.

Note:

To ensure the inference pipeline can handle new buffers, it is recommended to first call [ConfiguredInferModel.wait_for_async_ready\(\)](#).

Returns The async inference job object.

Return type [AsyncInferJob](#)

Raises [HailoRTException](#) -

set_scheduler_timeout(timeout_ms)

Sets the minimum number of send requests required before the network is considered ready to get run time from the scheduler. Sets the maximum time period that may pass before receiving run time from the scheduler. This will occur providing at least one send request has been sent, there is no minimum requirement for send requests, (e.g. threshold - see [ConfiguredInferModel.set_scheduler_threshold\(\)](#)).

The new time period will be measured after the previous time the scheduler allocated run time to this network group. Using this function is only allowed when scheduling_algorithm is not `HAILO_SCHEDULING_ALGORITHM_NONE`. The default timeout is 0ms.

Parameters `timeout_ms` (int) - The maximum time to wait for the scheduler to provide run time, in milliseconds.

Raises [HailoRTException](#) -

set_scheduler_threshold(threshold)

Sets the minimum number of send requests required before the network is considered ready to get run time from the scheduler.

Parameters `threshold` (int) - Threshold in number of frames.

Using this function is only allowed when scheduling_algorithm is not `HAILO_SCHEDULING_ALGORITHM_NONE`. The default threshold is 1. If at least one send request has been sent, but the threshold is not reached within a set time period (e.g. timeout - see [ConfiguredInferModel.set_scheduler_timeout\(\)](#)), the scheduler will consider the network ready regardless.

Raises [HailoRTException](#) -

set_scheduler_priority(priority)

Sets the priority of the network. When the network group scheduler will choose the next network, networks with higher priority will be prioritized in the selection. Larger number represents higher priority

Using this function is only allowed when scheduling_algorithm is not `HAILO_SCHEDULING_ALGORITHM_NONE`. The default priority is `HAILO_SCHEDULER_PRIORITY_NORMAL`.

Parameters `priority` (int) - Priority as a number between `HAILO_SCHEDULER_PRIORITY_MIN` - `HAILO_SCHEDULER_PRIORITY_MAX`.

Raises [HailoRTException](#) -

get_async_queue_size()

Returns Expected of a the number of inferences that can be queued simultaneously for execution.

Returns the number of inferences that can be queued simultaneously for execution

Return type size (int)

Raises `HailoRTException` -

`shutdown()`

Shuts the inference down. After calling this method, the model is no longer usable.

`class hailo_platform.pyhailort.pyhailort.AsyncInferJob(job)`

Bases: object

Hailo Asynchronous Inference Job Wrapper. It holds the result of the inference job (once ready), and provides an async poll method to check the job status.

`MILLISECOND = 0.001`

`__init__(job)`

`wait(timeout_ms)`

Waits for the asynchronous inference job to finish. If the async job and its callback have not completed within the given timeout, a HailoRTTimeout exception will be raised.

Parameters `timeout_ms (int)` - timeout The maximum time to wait.

Raises `HailoRTTimeout` -

`class hailo_platform.pyhailort.pyhailort.AsyncInferCompletionInfo(exception)`

Bases: object

Holds information about the async infer job

`__init__(exception)`

Parameters `exception (HailoRTException)` - an exception corresponding to the error that happened inside the async infer job.

`property exception`

Returns the exception that was set on this Infer job. if the job finished successfully, returns None.

`class hailo_platform.pyhailort.pyhailort.HailoRTTransformUtils`

Bases: object

`static get_dtype(data_bytes)`

Get data type from the number of bytes.

`static dequantize_output_buffer(src_buffer, dst_buffer, elements_count, quant_info)`

De-quantize the data in input buffer `src_buffer` and output it to the buffer `dst_buffer`

Parameters

- `src_buffer (numpy.ndarray)` - The input buffer containing the data to be de-quantized. The buffer's data type is the source data type.
- `dst_buffer (numpy.ndarray)` - The buffer that will contain the de-quantized data. The buffer's data type is the destination data type.
- `elements_count (int)` - The number of elements to de-quantize. This number must not exceed 'src_buffer' or 'dst_buffer' sizes.
- `quant_info (QuantInfo)` - The quantization info.

`static dequantize_output_buffer_in_place(raw_buffer, dst_dtype, elements_count, ...)`

De-quantize the output buffer `raw_buffer` to data type `dst_dtype`.

Parameters

- `raw_buffer (numpy.ndarray)` - The output buffer to be de-quantized. The buffer's data type is the source data type.
- `dst_dtype (numpy.dtype)` - The data type to de-quantize `raw_buffer` to.
- `elements_count (int)` - The number of elements to de-quantize. This number must not exceed 'raw_buffer' size.
- `quant_info (QuantInfo)` - The quantization info.

```
static quantize_input_buffer(src_buffer, dst_buffer, elements_count, quant_info)
```

Quantize the data in input buffer `src_buffer` and output it to the buffer `dst_buffer`

Parameters

- `src_buffer (numpy.ndarray)` - The input buffer containing the data to be quantized. The buffer's data type is the source data type.
- `dst_buffer (numpy.ndarray)` - The buffer that will contain the quantized data. The buffer's data type is the destination data type.
- `elements_count (int)` - The number of elements to quantize. This number must not exceed 'src_buffer' or 'dst_buffer' sizes.
- `quant_info (QuantInfo)` - The quantization info.

18. HailoRT Python GenAI API Reference

18.1. `hailo_platform.genai`

The GenAI module provides Python bindings for Generative AI functionality, including Large Language Models (LLMs).

18.1.1. LLM Classes

```
class hailo_platform.pyhailort.pyhailort.LLM(vdevice, model_path, lora_name=", ... )
```

Bases: object

Large Language Model inference interface for Hailo devices.

This class provides a high-level Python interface for running Large Language Models on Hailo hardware. It supports both streaming and non-streaming text generation, context management, and various generation parameters.

Features:

- Streaming and non-streaming text generation
- Structured prompts (chat format) and raw text prompts
- Context management for multi-turn conversations
- Flexible generation parameters (temperature, top_p, etc.)
- LoRA (Low-Rank Adaptation) support for fine-tuned models
- Built-in tokenization utilities

Example:

```
# Basic usage
with VDevice() as vd:
    with LLM(vd, "model.hef", optimize_memory_on_device=False) as llm:
        response = llm.generate_all("Hello, how are you?")
        print(response)

# Streaming generation
with VDevice() as vd:
    with LLM(vd, "model.hef", optimize_memory_on_device=False) as llm:
        prompt = [{"role": "user", "content": "Tell me a joke"}]
        with llm.generate(prompt, temperature=0.8) as gen:
            for token in gen:
                print(token, end=' ', flush=True)
```

Note:

- The VDevice must remain active for the lifetime of the LLM instance.
- Using this class within a context manager (`with` statement) is recommended to ensure proper resource cleanup.

```
__init__(vdevice, model_path, lora_name=", optimize_memory_on_device=False)
```

Initialize the LLM with a Hailo device and model file.

Parameters

- `vdevice` (`VDevice`) – An active VDevice instance managing Hailo hardware.
- `model_path` (`str`) – Path to the compiled HEF (Hailo Executable Format) file.

- `lora_name(str, optional)` - Name of LoRA adapter to load. Defaults to "".
- `optimize_memory_on_device(bool, optional)` - Whether to optimize memory usage on device by enabling client-side tokenization. Defaults to False.

Example:

```
with VDevice() as vd:
    llm = LLM(vd, "models/qwen-2.5-1.5b-instruct.hf", lora_name="chat_
        ↪adapter", optimize_memory_on_device=True)

release()
```

Release internal resources and cleanup.

This method is called automatically when exiting the context manager. Manual calls are needed only in case context manager is not used.

`generate(prompt, tools=None, temperature=None, top_p=None, top_k=None, frequency_penalty=None, ...)`

Generate text in streaming mode, yielding tokens as they are produced.

This method returns a context manager that provides an iterator for receiving generated tokens one by one. This is ideal for interactive applications where you want to display text as it's being generated.

Parameters

- `prompt(str or list)` - Input prompt. Can be: - str: Raw text prompt, to be inserted to the model as is. - list: Structured chat format [{"role": "user", "content": "..."}]
- `tools(list, optional)` - List of tool/function definitions for function calling. Each tool should be a dict with tool schema. Only allowed on first prompt. Example: [{"type": "function", "function": {"name": "get_weather", ...}}]
- `temperature(float, optional)` - Sampling temperature (0.0-2.0). Lower values = more deterministic, higher = more creative. Default varies by model.
- `top_p(float, optional)` - Nucleus sampling threshold (0.0-1.0). Only consider tokens with cumulative probability up to this value.
- `top_k(int, optional)` - Top-k sampling limit. Only consider top-k most likely tokens.
- `frequency_penalty(float, optional)` - Repetition penalty. Positive values discourage repetition.
- `max_generated_tokens(int, optional)` - Maximum number of tokens to generate. If None, uses model default.
- `do_sample(bool, optional)` - Whether to use sampling (True) or greedy decoding (False).
- `seed(int, optional)` - Random seed for reproducible outputs. If None, uses random seed.

Returns Generator completion object that can be iterated to receive tokens one by one.

Return type `LLMGeneratorCompletion`

Example:

```
# Streaming with structured prompt
prompt = [{"role": "user", "content": "Tell me a story"}]
with llm.generate(prompt, temperature=0.8, max_generated_tokens=100) as
    ↪gen:
        for token in gen:
            print(token, end=' ', flush=True)

# Streaming with tools (function calling)
```

(continues on next page)

(continued from previous page)

```
tools = [{"type": "function", "function": {"name": "get_weather", "description": "Get weather", "parameters": {...}}}]  
prompt = [{"role": "user", "content": "What's the weather?"}]  
with llm.generate(prompt, tools=tools, temperature=0.7) as gen:  
    for token in gen:  
        print(token, end=' ', flush=True)  
  
    # Streaming with raw prompt  
    with llm.generate("Once upon a time", temperature=0.7) as gen:  
        for token in gen:  
            print(token, end=' ', flush=True)
```

Note: The conversation context is automatically maintained between calls. Use `clear_context()` to reset the conversation history and get a fresh context.

Tools and system-role messages (e.g. `{"role": "system", "content": "..."}) can only be provided on a fresh context. Providing them on consecutive calls will raise an error.`

`generate_all(prompt, tools=None, temperature=None, top_p=None, top_k=None, ...)`

Generate complete text response in non-streaming mode.

This method generates the complete response and returns it as a single string. Use this when you need the full response before proceeding, or for batch processing where streaming is not required.

Parameters

- `prompt(str or list)`-Input prompt. Can be: - str: Raw text prompt, to be inserted to the model as is. - list: Structured chat format `[{"role": "user", "content": "..."}]`
- `tools(list, optional)` – List of tool/function definitions for function calling. Each tool should be a dict with tool schema. Only allowed on first prompt. Example: `[{"type": "function", "function": {"name": "get_weather", ...}}]`
- `temperature(float, optional)` – Sampling temperature (0.0-2.0). Lower values = more deterministic, higher = more creative.
- `top_p(float, optional)` – Nucleus sampling threshold (0.0-1.0). Only consider tokens with cumulative probability up to this value.
- `top_k(int, optional)` – Top-k sampling limit. Only consider top-k most likely tokens.
- `frequency_penalty(float, optional)` – Repetition penalty. Positive values discourage repetition.
- `max_generated_tokens(int, optional)` – Maximum number of tokens to generate. If None, uses model default.
- `do_sample(bool, optional)` – Whether to use sampling (True) or greedy decoding (False).
- `seed(int, optional)` – Random seed for reproducible outputs. If None, uses random seed.
- `timeout_ms(int, optional)` – Timeout in milliseconds.

Returns Complete generated response as a single string.

Return type str

Example:

```
# Simple text generation
response = llm.generate_all("Explain quantum physics")
print(response)

# Chat-style interaction
prompt = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "What is the capital of France?"}
]
print(llm.generate_all(prompt, temperature=0.1))

# With specific parameters
response = llm.generate_all(
    "1+1=",
    temperature=0.8,
    max_generated_tokens=1
)
```

Note: The conversation context is automatically maintained between calls. Use `clear_context()` to reset the conversation history and get a fresh context.

Tools and system-role messages (e.g. `{"role": "system", "content": "...")`) can only be provided on a fresh context. Providing them on consecutive calls will raise an error.

`tokenize(text)`

Tokenize input text using the model's tokenizer.

This method converts text into a list of token IDs that the model uses internally. Useful for understanding how the model processes input, for debugging and for understanding how many tokens are needed for a given prompt.

Parameters `text (str)` – Text to tokenize.

Returns List of token IDs (integers) representing the input text.

Return type list

Example:

```
tokens = llm.tokenize("Hello, world!")
print(f"Token IDs: {tokens}")
print(f"Number of tokens: {len(tokens)}")
```

`get_context_usage_size()`

Get the context usage size of the LLM model.

This method returns the total number of tokens currently stored in the model's context, including both input tokens and generated tokens from previous interactions. The context usage includes the conversation history and any tokens that have been processed but not yet cleared by `clear_context()`.

Returns Current number of tokens in the context.

Return type int

`max_context_capacity()`

Obtain the maximum context capacity of the LLM model.

This method returns the maximum number of tokens that can be stored in the model's context. This value is determined in the model compilation, represents the upper limit for context storage, and is not configurable.

Returns Maximum number of tokens that can be stored in the context.

Return type int

Example:

```
max_capacity = llm.max_context_capacity()
current_usage = llm.get_context_usage_size()
print(f"Context usage: {current_usage}/{max_capacity}")
```

clear_context()

Clear the conversation context/history.

This method resets the model's memory of previous interactions, starting fresh for the next generation. Call this between separate conversations or when you want to start a new topic without prior context.

Example:

```
# First conversation
response1 = llm.generate_all("My name is Alice")
response2 = llm.generate_all("What's my name?") # Will remember "Alice"

# Clear context and start fresh
llm.clear_context()
response3 = llm.generate_all("What's my name?") # Won't remember "Alice"
```

Note: This operation is irreversible. Once context is cleared, previous conversation history cannot be recovered.

get_generation_recovery_sequence()

Get the current generation recovery sequence.

Recovery sequences are used to handle error conditions during generation, or for when a generation is interrupted (e.g. by reaching max_generated_tokens). This is an advanced feature for error handling and debugging.

Returns Recovery sequence string, to be inserted to the model when a generation is interrupted.

Return type str

Note: This is an advanced feature. Most users won't need to use this method.

set_generation_recovery_sequence(*sequence*)

Set a custom generation recovery sequence.

Recovery sequences are used to handle error conditions during generation, or for when a generation is interrupted (e.g. by reaching max_generated_tokens). This is an advanced feature for error handling and debugging.

Parameters *sequence* (str) – Recovery sequence string, to be inserted to the model when a generation is interrupted.

Note: This is an advanced feature. Most users won't need to use this method.

prompt_template()

Get the model's prompt template format.

Different models use different prompt templates (chat template) for translating structured prompts (list of dicts) to raw text prompts. This method returns information about the expected format for this model.

Returns Prompt template as a string (Jinja2 template).

Return type str

Example:

```
template = llm.prompt_template()
print(f"Model expects format: {template}")
```

Note: When using structured prompts (list of dicts), the template is applied automatically. This method is useful when constructing raw text prompts.

`set_stop_tokens(stop_tokens)`

Set custom stop tokens for generation.

Stop tokens are sequences that, when generated, will cause the model to stop generating additional text. This is useful for controlling response length and format.

A stop token can be a string that represents multiple tokens. In this case, the generation will stop when the exact string is generated as a sequence. All stop tokens are checked after each generated token. Setting too many can affect performance. Setting stop tokens overrides current stop tokens. Setting empty list will remove all stop tokens.

Parameters `stop_tokens (list)` – List of strings that when generated will stop the generation.

`get_stop_tokens()`

Get the currently configured stop tokens.

Returns List of strings, where each string is a current stop token.

Return type list

Example:

```
current_stops = llm.get_stop_tokens()
print(f"Current stop tokens: {current_stops}")
```

`save_context()`

Save the current context of the LLM as a binary blob.

This method saves the current conversation context (including all previous interactions) as a binary blob that can be used to restore the context later using `load_context()`. This is useful for saving conversation state between sessions or for implementing context switching.

Returns Binary blob containing the saved context data.

Return type bytes

Example:

```
# Save context after a conversation
context_data = llm.save_context()

# Save to file for later use
with open("conversation_context.bin", "wb") as f:
    f.write(context_data)
```

Note:

- The context is unique for a specific LLM model
 - Trying to load a context from a different model will result in an error
 - The context includes all conversation history up to this point
-

```
load_context(context_data)
```

Load a previously saved context into the LLM.

This method restores a conversation context that was previously saved using `save_context()`. This allows to continue the conversation from where it left off, or to switch between different conversation contexts.

Parameters `context_data (bytes)` – Binary blob containing the saved context data.

Example:

```
# Load context from file
with open("conversation_context.bin", "rb") as f:
    context_data = f.read()

# Restore the conversation context
llm.load_context(context_data)

# Continue the conversation
response = llm.generate_all("What were we talking about?")
```

Note:

- The context must be from the same LLM model
 - Loading a context will replace the current context completely
 - This operation will fail if the context is from a different model
-

```
class hailo_platform.pyhailort.pyhailort.LLMGeneratorCompletion(generator_completion)
```

Bases: `object`

Generator completion object for streaming LLM text generation.

This class provides an iterator interface for receiving generated tokens one by one from a Large Language Model. It is created by the `LLM.generate()` method and should be used within a context manager for proper resource cleanup.

Example:

```
with llm.generate(prompt="Hello", max_generated_tokens=50) as gen_completion:
    for token in gen_completion:
        print(token, end='', flush=True)
```

```
__init__(generator_completion)
```

Initialize the generator completion wrapper.

Parameters `generator_completion` – Internal generator completion object.

```
release()
```

Release internal resources.

This method is called automatically when exiting the context manager. Manual calls are needed only in case context manager is not used.

```
property generation_status
```

Get the current generation status.

Returns Current status of the generation process.

Return type `LLMGeneratorCompletionStatus`

```
read(timeout_ms=10000)
```

```
read_all(timeout_ms=10000)
```

Read all remaining tokens and return complete response.

This method blocks until generation is complete or timeout occurs.

Parameters `timeout_ms` (int, optional) – Timeout in milliseconds. Defaults to 10000.

Returns Complete generated response as a single string.

Return type str

Example:

```
full_response = gen_completion.read_all()  
print(f"Complete response: {full_response}")
```

18.1.2. VLM Classes

```
class hailo_platform.pyhailort.pyhailort.VLM(vdevice, model_path, ...)
```

Bases: object

Vision Language Model inference interface for Hailo devices.

This class provides a high-level Python interface for running Vision Language Models on Hailo hardware. It supports both streaming and non-streaming text generation, context management, and various generation parameters.

Features:

- Streaming and non-streaming text generation, supporting multiple (or 0) images per generation
- Structured prompts (chat format) and raw text prompts
- Context management for multi-turn conversations
- Flexible generation parameters (temperature, top_p, etc.)
- Built-in tokenization utilities

Example:

```
# Basic usage  
with VDevice() as vd:  
    with VLM(vd, "model.hef", optimize_memory_on_device=False) as vlm:  
        response = vlm.generate_all(prompt="Hello, how are you?", frames=[ ])  
        print(response)  
  
# Streaming generation  
with VDevice() as vd:  
    with VLM(vd, "model.hef", optimize_memory_on_device=False) as vlm:  
        prompt = [{"role": "user", "content": [{"type": "text", "text": "Describe this image"}, {"type": "image"}]}]  
        with vlm.generate(prompt, frames=[frame_numpy_array], temperature=0.8) as gen:  
            for token in gen:  
                print(token, end=' ', flush=True)
```

Note:

- The VDevice must remain active for the lifetime of the VLM instance.
- Using this class within a context manager (`with` statement) is recommended to ensure proper resource cleanup.

- The given frames must be in the same order, shape, and dtype as the input frame of the model.
-

`__init__(vdevice, model_path, optimize_memory_on_device=False)`

Initialize the VLM with a Hailo device and model file.

Parameters

- `vdevice (VDevice)` - An active VDevice instance managing Hailo hardware.
- `model_path (str)` - Path to the compiled HEF (Hailo Executable Format) file.
- `optimize_memory_on_device (bool, optional)` - Whether to optimize memory usage on device by enabling client-side tokenization. Defaults to False.

Example:

```
with VDevice() as vd:  
    vlm = VLM(vd, "models/qwen2-vl-2b-instruct.hef", optimize_memory_on_  
    ↴device=True)
```

`release()`

Release internal resources and cleanup.

This method is called automatically when exiting the context manager. Manual calls are needed only in case context manager is not used.

`generate(prompt, frames, videos=None, temperature=None, top_p=None, top_k=None, ...)`

Generate text in streaming mode with vision input, yielding tokens as they are produced.

This method returns a context manager that provides an iterator for receiving generated tokens one by one. This is ideal for interactive applications where you want to display text as it's being generated.

Parameters

- `prompt (str or list)` - Input prompt. Can be: - str: Raw text prompt, to be inserted to the model as is. - list: Structured chat format with or without vision content [{"role": "user", "content": [{"type": "text", "text": "..."}, {"type": "image"}]}]
- `frames (list)` - List of numpy arrays representing input images/frames. Empty list means no vision input. Must match model's expected input format.
- `videos (list, optional)` - List of lists of numpy arrays representing input videos as a list of frames. Each frame must match model's expected input format. An empty list means no video input.
- `temperature (float, optional)` - Sampling temperature (0.0-2.0). Lower values = more deterministic, higher = more creative. Default varies by model.
- `top_p (float, optional)` - Nucleus sampling threshold (0.0-1.0). Only consider tokens with cumulative probability up to this value.
- `top_k (int, optional)` - Top-k sampling limit. Only consider top-k most likely tokens.
- `frequency_penalty (float, optional)` - Repetition penalty. Positive values discourage repetition.
- `max_generated_tokens (int, optional)` - Maximum number of tokens to generate. If None, uses model default.
- `do_sample (bool, optional)` - Whether to use sampling (True) or greedy decoding (False).
- `seed (int, optional)` - Random seed for reproducible outputs. If None, uses random seed.

Returns Generator completion object that can be iterated to receive tokens one by one.

Return type `LLMGeneratorCompletion`

Example:

```
# Streaming with structured prompt and image
prompt = [{"role": "user", "content": [{"type": "text", "text": "Describe this image"}, {"type": "image"}]}]
with vlm.generate(prompt, frames=[image_array], temperature=0.7) as gen:
    for token in gen:
        print(token, end=' ', flush=True)
```

Note: The conversation context is automatically maintained between calls. Use `clear_context()` to reset the conversation history and get a fresh context.

System-role messages (e.g. `{"role": "system", "content": "..."}`) can only be provided on a fresh context. Providing them on consecutive calls will raise an error.

`generate_all(prompt, frames, videos=None, temperature=None, top_p=None, top_k=None, ...)`

Generate complete text response in non-streaming mode with vision input.

This method generates the complete response and returns it as a single string. Use this when you need the full response before proceeding, or for batch processing where streaming is not required.

Parameters

- `prompt` (str or list) - Input prompt. Can be: - str: Raw text prompt, to be inserted to the model as is. - list: Structured chat format with vision content [`{"role": "user", "content": [{"type": "text", "text": "..."}, {"type": "image"}]}`]
- `frames` (list) - List of numpy arrays representing input images/frames. Empty list means no vision input. Must match model's expected input format.
- `videos` (list, optional) - List of lists of numpy arrays representing input videos as a list of frames. Each frame must match model's expected input format. An empty list means no video input.
- `temperature` (float, optional) - Sampling temperature (0.0-2.0). Lower values = more deterministic, higher = more creative.
- `top_p` (float, optional) - Nucleus sampling threshold (0.0-1.0). Only consider tokens with cumulative probability up to this value.
- `top_k` (int, optional) - Top-k sampling limit. Only consider top-k most likely tokens.
- `frequency_penalty` (float, optional) - Repetition penalty. Positive values discourage repetition.
- `max_generated_tokens` (int, optional) - Maximum number of tokens to generate. If None, uses model default.
- `do_sample` (bool, optional) - Whether to use sampling (True) or greedy decoding (False).
- `seed` (int, optional) - Random seed for reproducible outputs. If None, uses random seed.
- `timeout_ms` (int, optional) - Timeout in milliseconds.

Returns Complete generated response as a single string.

Return type str

Example:

```
# Chat-style interaction with image
prompt = [
    {"role": "user", "content": [
        {"type": "text", "text": "What objects do you see?"},
        {"type": "image"}
    ]}
]
print(vlm.generate_all(prompt, frames=[image_array], videos=[video_
    ↪array], temperature=0.1))

# Simple text generation (without vision)
response = vlm.generate_all("Tell me a joke", frames=[ ])
```

Note: The conversation context is automatically maintained between calls. Use `clear_context()` to reset the conversation history and get a fresh context.

System-role messages (e.g. `{"role": "system", "content": "..."}) can only be provided on a fresh context. Providing them on consecutive calls will raise an error.`

`tokenize(text)`

Tokenize input text using the model's tokenizer.

This method converts text into a list of token IDs that the model uses internally. Useful for understanding how the model processes input, for debugging and for understanding how many tokens are needed for a given prompt.

Parameters `text (str)` - Text to tokenize.

Returns List of token IDs (integers) representing the input text.

Return type list

Example:

```
tokens = vlm.tokenize("Describe this image")
print(f"Token IDs: {tokens}")
print(f"Number of tokens: {len(tokens)}")
```

`get_context_usage_size()`

Get the context usage size of the VLM model.

This method returns the total number of tokens currently stored in the model's context, including both input tokens and generated tokens from previous interactions. The context usage includes the conversation history and any tokens that have been processed but not yet cleared by `clear_context()`.

Returns Current number of tokens in the context.

Return type int

`max_context_capacity()`

Obtain the maximum context capacity of the VLM model.

This method returns the maximum number of tokens that can be stored in the model's context. This value is determined in the model compilation, represents the upper limit for context storage, and is not configurable.

Returns Maximum number of tokens that can be stored in the context.

Return type int

Example:

```
max_capacity = vlm.max_context_capacity()
current_usage = vlm.get_context_usage_size()
print(f"Context usage: {current_usage}/{max_capacity}")
```

clear_context()

Clear the conversation context/history.

This method resets the model's memory of previous interactions, starting fresh for the next generation. Call this between separate conversations or when you want to start a new topic without prior context.

Example:

```
# First conversation
response1 = vlm.generate_all("What's in this image?", frames=[image1])
response2 = vlm.generate_all("What about the colors?", frames=[]) # Will
    ↳remember image1

# Clear context and start fresh
vlm.clear_context()
response3 = vlm.generate_all("What about the colors?", frames=[]) # Won't
    ↳remember image1
```

Note: This operation is irreversible. Once context is cleared, previous conversation history cannot be recovered.

get_generation_recovery_sequence()

Get the current generation recovery sequence.

Recovery sequences are used to handle error conditions during generation, or for when a generation is interrupted (e.g. by reaching max_generated_tokens). This is an advanced feature for error handling and debugging.

Returns Recovery sequence string, to be inserted to the model when a generation is interrupted.

Return type str

Note: This is an advanced feature. Most users won't need to use this method.

set_generation_recovery_sequence(*sequence*)

Set a custom generation recovery sequence.

Recovery sequences are used to handle error conditions during generation, or for when a generation is interrupted (e.g. by reaching max_generated_tokens). This is an advanced feature for error handling and debugging.

Parameters *sequence* (str) - Recovery sequence string, to be inserted to the model when a generation is interrupted.

Note: This is an advanced feature. Most users won't need to use this method.

prompt_template()

Get the model's prompt template format.

Different models use different prompt templates (chat template) for translating structured prompts (list of dicts) to raw text prompts. This method returns information about the expected format for this model.

Returns Prompt template as a string (Jinja2 template).

Return type str

Example:

```
template = vlm.prompt_template()
print(f"Model expects format: {template}")
```

Note: When using structured prompts (list of dicts), the template is applied automatically. This method is useful when constructing raw text prompts.

set_stop_tokens(*stop_tokens*)

Set custom stop tokens for generation.

Stop tokens are sequences that, when generated, will cause the model to stop generating additional text. This is useful for controlling response length and format.

Parameters *stop_tokens* (list) – List of strings that when generated will stop the generation.

get_stop_tokens()

Get the currently configured stop tokens.

Returns List of strings, where each string is a current stop token.

Return type list

Example:

```
current_stops = vlm.get_stop_tokens()
print(f"Current stop tokens: {current_stops}")
```

save_context()

Save the current context of the VLM as a binary blob.

This method saves the current conversation context (including all previous interactions) as a binary blob that can be used to restore the context later using `load_context()`. This is useful for saving conversation state between sessions or for implementing context switching.

Returns Binary blob containing the saved context data.

Return type bytes

Example:

```
# Save context after a conversation
context_data = vlm.save_context()

# Save to file for later use
with open("conversation_context.bin", "wb") as f:
    f.write(context_data)
```

Note:

- The context is unique for a specific VLM model
 - Trying to load a context from a different model will result in an error
 - The context includes all conversation history up to this point
-

load_context(*context_data*)

Load a previously saved context into the VLM.

This method restores a conversation context that was previously saved using `save_context()`. This allows to continue the conversation from where it left off, or to switch between different conversation contexts.

Parameters `context_data (bytes)` – Binary blob containing the saved context data.

Example:

```
# Load context from file
with open("conversation_context.bin", "rb") as f:
    context_data = f.read()

# Restore the conversation context
vlm.load_context(context_data)

# Continue the conversation
response = vlm.generate_all("What were we talking about?", frames=[ ])
```

Note:

- The context must be from the same VLM model
 - Loading a context will replace the current context completely
 - This operation will fail if the context is from a different model
-

`input_frame_size()`

Get the expected input frame size in bytes.

This method returns the total size in bytes that each input frame should have. Use this to ensure your numpy arrays have the correct total byte size.

Returns Expected frame size in bytes.

Return type int

Example:

```
frame_size = vlm.input_frame_size()
print(f"Expected frame size: {frame_size} bytes")
```

`input_frame_shape()`

Get the expected input frame shape.

This method returns the expected dimensions for input frames. Use this to ensure your numpy arrays have the correct shape.

Returns Frame shape as (height, width, channels).

Return type list

Example:

```
shape = vlm.input_frame_shape()
print(f"Expected frame shape: {shape}")
# Create a frame with correct shape
frame = np.zeros(shape, dtype=np.uint8)
```

`input_frame_format_type()`

Get the expected input frame data type.

This method returns the expected numpy data type for input frames. Use this to ensure your numpy arrays have the correct dtype.

Returns Expected data type for input frames.

Return type numpy.dtype

Example:

```
dtype = vlm.input_frame_format_type()
print(f"Expected frame dtype: {dtype}")
# Create a frame with correct dtype
frame = np.zeros((224, 224, 3), dtype=dtype)
```

`input_frame_format_order()`

Get the expected input frame format order.

This method returns the expected channel ordering for input frames (Usually NHWC, which means (height, width, channels)).

Returns Expected format order for input frames.

Return type `_pyhailort.hailo_format_order_t`

Example:

```
format_order = vlm.input_frame_format_order()
print(f"Expected format order: {format_order}")
```

18.1.3. Speech2Text Classes

`class hailo_platform.pyhailort.pyhailort.Speech2Text(vdevice, model_path)`

Bases: `object`

Speech-to-Text model inference interface for Hailo devices.

This class provides a high-level Python interface for running Speech-to-Text models on Hailo hardware. It supports audio transcription and translation with configurable language settings and task types.

Features:

- Audio transcription with timestamped segments
- Language translation capabilities
- Support for multiple output formats (segments vs complete text)
- Configurable task types (transcribe/translate)
- Language-specific processing

Example:

```
# Basic transcription
with VDevice() as vd:
    with Speech2Text(vd, "whisper.hef") as s2t:
        print(s2t.generate_all_text(task=Speech2TextTask.TRANSCRIBE, language="en",
                                     audio_data=audio_array))

# Transcription with segments
with VDevice() as vd:
    with Speech2Text(vd, "whisper.hef") as s2t:
        all_segments = s2t.generate_all_segments(task=Speech2TextTask.TRANSCRIBE,
                                                language="en", audio_data=audio_array)
        print(segment for segment in all_segments)
```

Note:

- The VDevice must remain active for the lifetime of the Speech2Text instance.
- Using this class within a context manager (`with` statement) is recommended to ensure proper resource cleanup.

- Audio input must be in PCM float32 format (normalized to [-1.0, 1.0], mono, little-endian, 16 kHz).

`__init__(vdevice, model_path)`

Initialize the Speech2Text model with a Hailo device and model file.

Parameters

- `vdevice (VDevice)` - An active VDevice instance managing Hailo hardware.
- `model_path (str)` - Path to the compiled HEF (Hailo Executable Format) file.

Example:

```
with VDevice() as vd:  
    s2t = Speech2Text(vd, "models/whisper.hef")
```

`release()`

Release internal resources and cleanup.

This method is called automatically when exiting the context manager. Manual calls are needed only in case context manager is not used.

`generate_all_text(audio_data, task=Speech2TextTask.TRANSCRIBE, language=None, ...)`

Generate complete transcription as a single string.

Parameters

- `audio_data (numpy.ndarray)` - Audio data as numpy array in PCM float32 format (normalized to [-1.0, 1.0], mono, little-endian, 16 kHz).
- `task (Speech2TextTask, optional)` - Task to perform (transcribe or translate). Defaults to TRANSCRIBE.
- `language (str, optional)` - Language to use for translation, in the format of ISO-639-1 two-letter code, for example: "en", "fr", etc. Defaults to None, which means that the model will automatically detect the language.
- `repetition_penalty(float, optional)` - Repetition penalty to use for the generation process. Defaults to 1.0, which means that the model will not apply any repetition penalty.
- `timeout_ms(int, optional)` - Timeout in milliseconds. Defaults to 10000.

Returns Complete transcription as a single string.

Return type str

Example:

```
complete_text = s2t.generate_all_text(task=Speech2TextTask.TRANSCRIBE,  
                                      language="en", audio_data=audio_array)  
print(f"Transcription: {complete_text}")
```

`generate_all_segments(audio_data, task=Speech2TextTask.TRANSCRIBE, language=None, ...)`

Generate transcription with timestamped segments.

Parameters

- `audio_data (numpy.ndarray)` - Audio data as numpy array in PCM float32 format (normalized to [-1.0, 1.0], mono, little-endian, 16 kHz).
- `task (Speech2TextTask, optional)` - Task to perform (transcribe or translate). Defaults to TRANSCRIBE.
- `language (str, optional)` - Language to use for translation, in the format of ISO-639-1 two-letter code, for example: "en", "fr", etc. Defaults to None, which means that the model will automatically detect the language.

- `repetition_penalty(float, optional)` - Repetition penalty to use for the generation process. Defaults to 1.0, which means that the model will not apply any repetition penalty.

- `timeout_ms(int, optional)` - Timeout in milliseconds. Defaults to 10000.

Returns List of SegmentInfo objects containing start/end timestamps and transcribed text.

Return type list[[SegmentInfo](#)]

Example:

```
segments = s2t.generate_all_segments(task=Speech2TextTask.TRANSCRIBE,
                                     language="en", audio_data=audio_array)
for segment in segments:
    print(f"[{segment.start_sec:.2f}s-{segment.end_sec:.2f}s] {segment.
                                     text}")
```

`tokenize(text)`

Tokenize input text using the model's tokenizer.

This method converts text into a list of token IDs that the model uses internally. Useful for understanding how the model processes input, for debugging and for understanding what is the tokens count of specific text.

Parameters `text (str)` - Text to tokenize.

Returns List of token IDs (integers) representing the input text.

Return type list

Example:

```
tokens = s2t.tokenize("Hello, world!")
print(f"Token IDs: {tokens}")
print(f"Number of tokens: {len(tokens)}")
```

`class hailo_platform.pyhailort.pyhailort.SegmentInfo(segment_info)`

Bases: object

Represents a transcribed audio segment with timing information.

This class contains the transcribed text along with start and end timestamps for a specific segment of the audio input.

Example:

```
for segment in segments:
    print(f"[{segment.start_sec:.2f}s-{segment.end_sec:.2f}s] {segment.text}")
```

`__init__(segment_info)`

Initialize a SegmentInfo object.

Parameters `segment_info` - Internal segment object containing timing and text data.

`property start_sec`

Get the start time of the segment.

Returns Start time in seconds, relative to the start of the audio.

Return type float

`property end_sec`

Get the end time of the segment.

Returns End time in seconds, relative to the start of the audio.

Return type float

```
property text
```

Get the transcribed text of the segment.

Returns Transcribed text for this segment.

Return type str

```
class hailo_platform.pyhailort.pyhailort.Speech2TextTask(value)
```

Bases: enum.Enum

The available Speech2Text task types.

This enum defines the supported task types for speech processing:

- TRANSCRIBE: Transcribe speech to text in the same language
- TRANSLATE: Translate speech to text in a different language

```
TRANSCRIBE = <Speech2TextTask.TRANSCRIBE: 0>
```

```
TRANSLATE = <Speech2TextTask.TRANSLATE: 1>
```

Python Module Index

h

`hailo_platform.genai`, [338](#)
`hailo_platform.pyhailort.pyhailort`,
 [304](#)