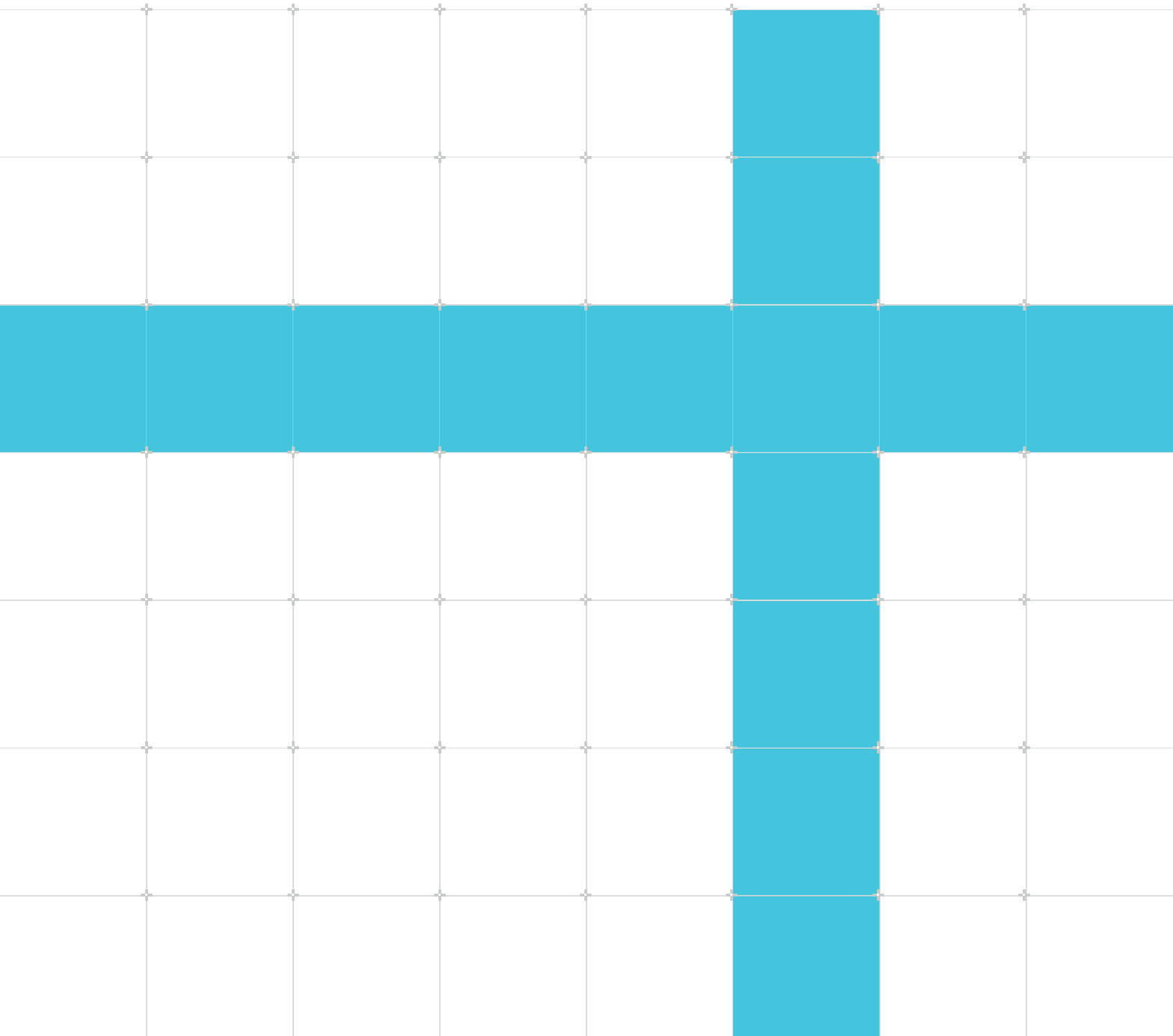
Ethos-U Getting Started Guide

Version 1.0

Issue 01
109267_0100_01_en



Ethos-U Getting Started Guide

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	1 August 2023	Non-Confidential	Initial release

Proprietary Notice

or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

Thank you for using the Arm Ethos-U NPU processor series. To provide you with the best experience for developing Machine Learning (ML) applications with the Arm Ethos-U NPUs we design our tools to make software engineering easy and productive. In addition Arm provides supportive material and collaborates with many AI partners to complement our solution, for example with optimized ML models, MLOps integration, and evaluation boards. The Arm tool and software support is therefore expandable in various ways.

This Overview chapter contains:

1.1 Chapter overview

This Getting Started user's guide starts with this overview of the Arm Ethos-U software and tools support along with an outline of development workflow from starting ML model training up to debugging on hardware. In addition it contains the following chapters:

- Introduction to Arm Ethos-U NPU provides information about the various Ethos-U processors and access to detailed technical documentation.
- Tool Support for the Arm Ethos-U NPU explains the Vela compiler that transforms a ML model for execution on Ethos-U NPU and covers the ML Interference Advisor, Arm Virtual Hardware, and SDS-Framework for analysis, verification, and training.
- The Arm ML Zoo which gives you access to pre-trained models for various types of applications. It explains how to use the information that is provided.
- ML Embedded Evaluation Kit provides ML examples for a range of use cases that help to create your own applications for Cortex-M CPU and Ethos-U NPU.
- CMSIS-Pack Based Machine Learning Examples showing ML software integration using software components in form of CMSIS-Packs, and using of CMSIS-Toolbox to create compilation setup.
- [Integration of TensorFlow Lite Micro] provides an overview of the integration of the TensorFlow Lite Micro runtime and models into new or existing projects.
- Profiling and optimization of ML models describes how to analyze and optimize the ML model execution on Arm Ethos-U processor based systems.
- MLOps Systems describes the integration of the Arm foundation tools into MLOps systems that automate training and help selecting optimal ML models for your applications.
- Resources for Ethos-U gives an overview of the available resources and eco-system partners that support Ethos-U NPUs.

1.2 Target Audience

This Getting Started user's guide assumes some top-level knowledge about Cortex-M software development. It is written for the following audiences:

- Embedded Developers that want to utilize microcontroller devices that incorporate Ethos-U processors and need easy access to development tools, software examples, and additional usage information.
- MLOps system architects that want to extend support to the Ethos-U NPU processor series and need to integrate the various development tools into the development flows.
- Data scientists that analyze data to develop new ML models and require statistics of the model performance by using the various software tools.

1.3 Benefits of Machine Learning

Applying machine learning in edge devices enables a new range of applications. For example:

- Predictive maintenance: where sensors in a system are used to identify likely failures, allowing for proactive maintenance to prevent downtime.
- Speech recognition, making it possible to interact with devices using natural language.
- Image detection in factory automation to improve efficiency, reduce errors, and increase safety in manufacturing environments.
- Medical diagnosis to assist in medical treatment, helping to design personalized treatment plans.
- Enable robots to perceive and understand their environment, such as recognizing objects, detecting obstacles, and tracking people. These are just a few examples. The overall possibilities of machine learning are vast, and the technology is constantly evolving making it possible to innovate in many new areas.

Typically, machine learning models require a high quantity of reliable data for the models to perform accurate predictions. When training a machine learning model, engineers need to target and collect a large and representative sample of data. Data from the training set can be a collection of images, sensor data, and data collected from individual users of a device.

Today, AI and ML algorithms that operate on data from IoT endpoint devices frequently execute on cloud servers. But to meet real-time requirements of embedded systems the execution of the actual AI algorithm should be transferred to the edge device.

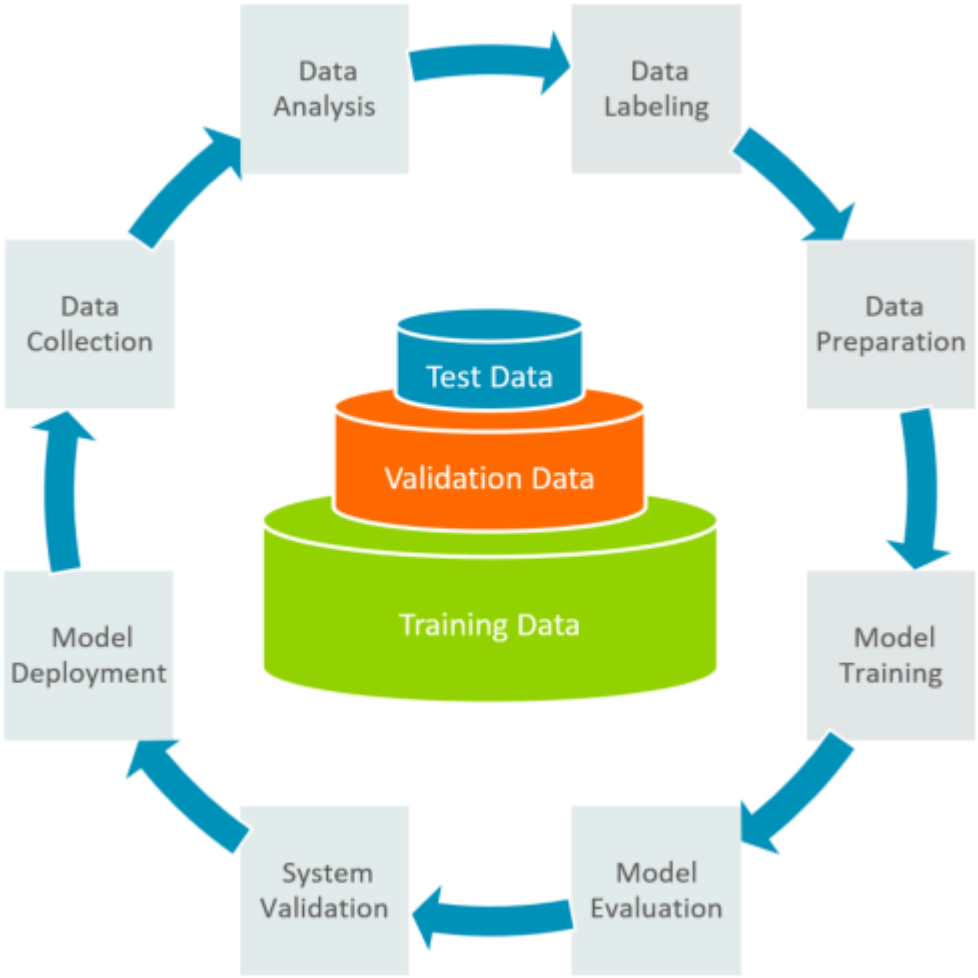
1.4 Overall Development Process

The software and system design of an embedded IoT and ML device can be separated into two parts:

- The classic embedded IoT software that requires efficient device drivers that interface with peripherals, a communication stack with security, and firmware update services.
- The system part that implements the machine learning algorithm. The ML part is frequently designed by utilizing Software-as-a-Service (SaaS) cloud environments that are specialized for ML algorithm development.

The machine learning algorithm is developed using a MLOps workflow. MLOps is a set of practices for developing, deploying, and maintaining machine learning models in production devices. The picture below shows the key phases of a MLOps development process.

Figure 1-1: MLOps Process Steps



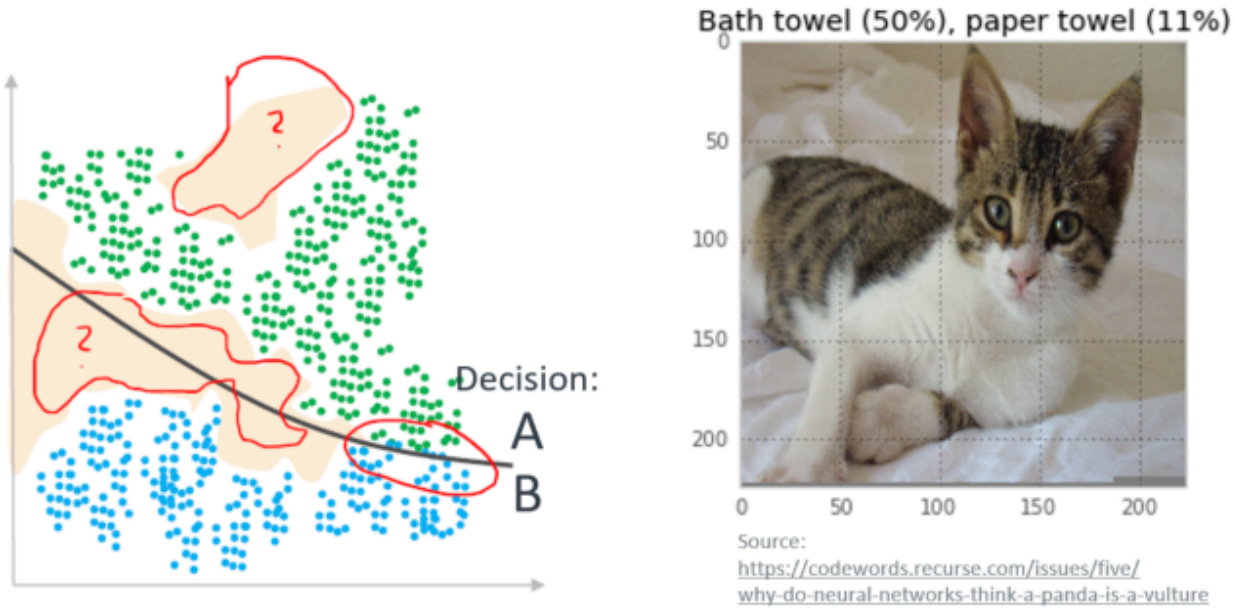
- Data Collection: is the foundation of a machine learning project. It is important that the machine learning model has enough data to learn from, the data covers as much scenarios as possible, and the data collected are accurate as this is critical for the performance of the model.
- Data Analysis: requires understanding of the scenarios that are recorded in the data collection. It may require that the data is cropped or cleaned before progressing to the next step as the collected data may have a mixture of scenarios.
- Data Labelling: is the annotation of the collected and cleaned data. For example, the data for a fitness tracker might be labelled with “walk”, “run”, and “rest” to describe the activity.
- Data Preparation: is making collected data available for the model training. Data might be separated also into test data (for smoke testing), validation data, and training data, which is typically the largest data set.
- Model Training: performs the training of the machine learning model. It is a process in which one or more machine learning algorithms are fed with training dataset from which it can learn.
- Model Evaluation: performs the training of the machine learning model. It is a process in which one or more machine learning algorithms are fed with training dataset from which it can learn.
- Model Evaluation: there are several algorithms to choose from, and it is an iterative process to determine the best ML algorithm for any given problem. In this step the best suited algorithm for the problem will be chosen.
- System Validation: is testing of the ML algorithm along with the model data when running on the final target system. The verification of the ML algorithm may be performed using a reduced set of validation data.
- Model Deployment: is the integration of the ML algorithm along with the model data into the final target system, which is in our case an embedded IoT application.

Machine learning models are typically tested and developed in isolated systems. The training of the ML model will mostly take place in the cloud, because both an extensive data set, and high compute power is a prerequisite. The algorithm execution based on the ML model can then take place directly on the IoT endpoint device.

Like humans learn and improve upon past experiences, the machine learning algorithms adaptively improve their performance as the number of samples available for learning increases. Correct decisions can only be made in areas where training data exist. “Learning” means therefore that ML algorithms are retrained based on new data that delivers new “experience”.

For example, if a picture recognition application never “saw” the picture of a cat, it cannot be correctly qualified (see below). It is therefore expected that IoT endpoint systems that incorporate AI and ML technology require periodic updates.

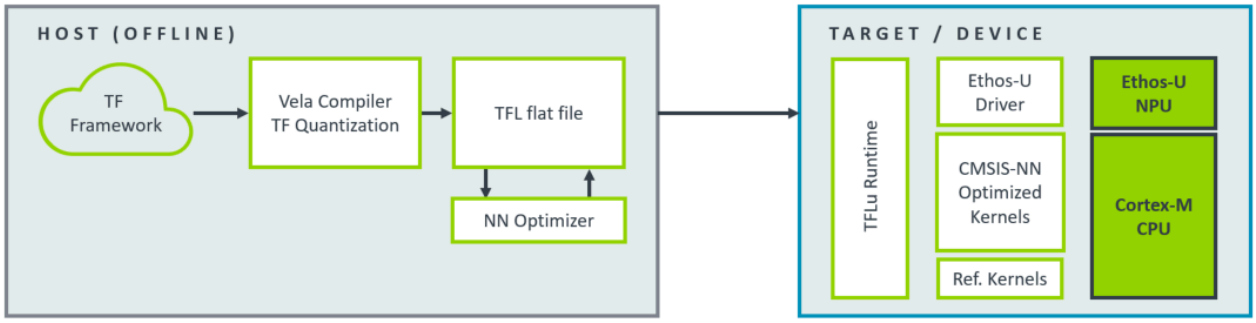
Figure 1-2: Missing Training Data



1.5 Targeting Ethos-U NPU Processors

The picture below outlines the software development flow for ML models utilizing an Ethos-U NPU.

Figure 1-3: Software Development Flow



The Host (Offline) process steps start with a trained ML model using the TensorFlow machine learning framework. Use for ML model conditioning techniques such as collaborative clustering, pruning, and quantization aware training (QAT) to improve model performance on Ethos-U while preserving its accuracy. Use TensorFlow Lite post-training quantization to `int8` data types to speed-up the ML model. The resulting TFL flatbuffer file (`*.tflite` file) is then transformed for execution on Ethos-U NPU using the Vela Compiler. The NN Optimizer identifies graphs to run on Ethos-U and optimizes, schedules, and allocates these graphs. Finally lossless compression reduces size of `*.tflite` file.

The Target / Device uses this `*.tflite` file as ML model for execution with the TFLu runtime system. The Ethos-U driver schedules the operators for execution on Ethos-U, whereas the CMSIS-NN library executes operators that cannot be mapped to Ethos-U using a software implementation on Cortex-M.

2. Introduction to Arm Ethos-U NPU

The majority of Machine Learning (ML) applications contain Neural Network (NN) inference operations. While it is possible to handle this in software running on a processor, we can increase the performance significantly if a hardware accelerator is used. The use of a hardware accelerator in NN inference usually improves the energy efficiency and allows a higher processor bandwidth to handle other tasks. There are many types of NN inference hardware accelerators: E.g. Arm® Ethos™-U series - a family of hardware accelerators designed for microcontrollers / System-on-Chips (SoC) which are known as Neural Processing Units (NPUs).

Currently the Ethos-U family contains two designs:

- Ethos-U55
- Ethos-U65

These two NPUs are already available in commercial products. For example:

- Ethos-U55 is used in the Alif Ensemble™ family from Alif Semiconductor™.
- Ethos-U65 is used in the i.MX 93 family from NXP®.

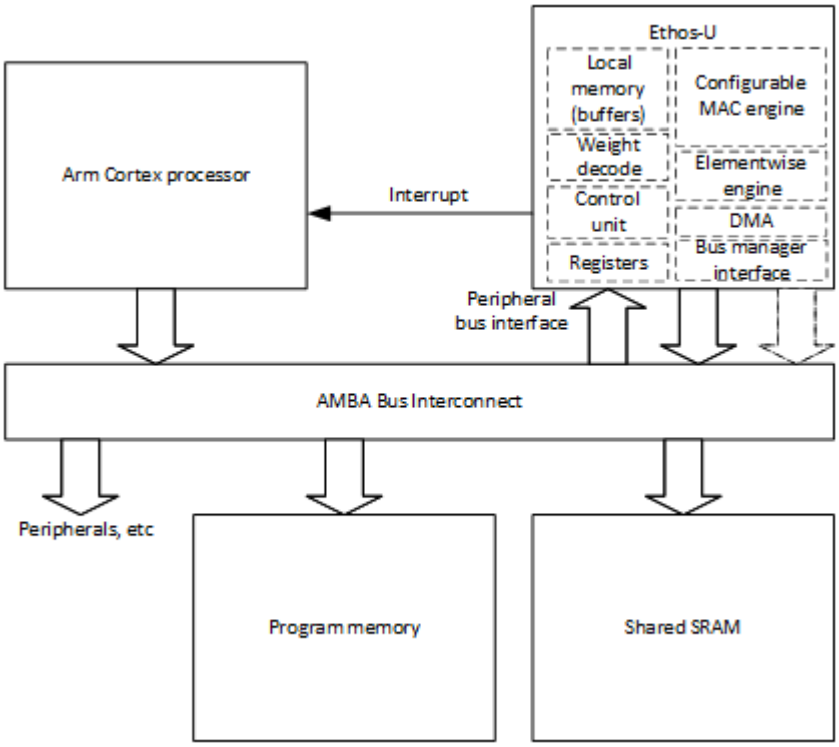
You can try out Ethos-U55/U65 without using real hardware. For example, you can use a simulated environment such as Arm Virtual Hardware (AVH) or Fixed Virtual Platform (FVP). These tools are described in other parts of this Getting Started Guide.

2.1 Ethos-U hardware concept

A Ethos-U NPU requires a host processor to control its operations. It has a range of programmable registers and several hardware interfaces:

- A peripheral bus interface to allow a processor to access the internal registers.
- One or two AMBA® bus manager interfaces to access the system memories.
- An interrupt output for sending an interrupt event to the host processor.

Figure 2-1: Concept of a microcontroller system with a Ethos-U



The majority of the processing in an NN inference is based on Multiply-Accumulate (MAC) computations. Inside the Ethos-U NPU there is a configurable MAC engine to handle MAC operations. The number of MACs that can be carried out per clock cycle is configurable by chip designers, ranging from 32 to 256 for the Ethos-U55 and either 256 or 512 for the Ethos-U65. There is also an element-wise data processing engine for other computations. To enhance the efficiency of NN inferences, the Ethos-U NPUs have a local memory inside for buffering the data they process. The Ethos-U NPUs also include a Direct Memory Access (DMA) engine so that data can be copied, before it is needed, from the shared memory to the local memory.

Because Ethos-U NPUs are designed for embedded systems that, in many instances, have a limited amount of memory, Ethos-U NPUs support NN weight data compression. The weight data is decoded on-the-fly during inference operations.

The operations of the Ethos-U are controlled by a number of registers which are memory mapped on the processor system. When the system needs to carry out a NN inference, the operations are broken down into a number of smaller jobs that can be carried out by the Ethos-U NPU. Under the control of software libraries, the jobs are issued to the Ethos-U NPU via the peripheral bus interface. Each time a job is completed the Ethos-U NPU issues an interrupt request to the processor so that the software library can then issue the next job.

In addition to the aforementioned key interfaces, the Ethos-U NPUs support additional interfaces, e.g. an interface for power management. More detailed information for the Ethos-U NPUs can be found in the:

- Ethos-U55 Technical Reference Manual
- Ethos-U65 Technical Reference Manual

There are a large number of Neural Network models available and in a number of instances some of those network models contain operators that are not supported by the Ethos-U hardware. In such cases, the unsupported operators will need to be handled by the software running on the processor. The ML operators supported in the Ethos-U55 and Ethos-U65 are documented in the following links:

- Ethos-U55:Supported data types and operators
- Ethos-U65:Supported data types and operators

2.2 Processor support requirements

There are two common types of system arrangements when an Ethos-U NPU is integrated into a microcontroller / SoC. These are:

- An Ethos-U NPU controlled by a Cortex-M4/M7/M33/M55/M85 processor, which also runs the application codes.
- An Ethos-U NPU integrated into an ML subsystem together with a Cortex-M processor. In this scenario, the ML subsystem is part of a larger SoC with one or more Cortex-A processors. The Cortex-A processor would dispatch the NN inference workloads to the Cortex-M processor in the ML subsystem, with the Cortex-M processor managing the low level control.

2.3 ML software support for Ethos-U

A ML software project is usually based on a specific ML software framework. The Ethos-U NPUs support the TensorFlow Lite Micro (TFLM), a popular ML software framework which is optimized for microcontrollers. Because the weights in the NN models in TFLM are quantized to 8-bit integers, the memory footprint of the NN models is reduced. Another advantage of using quantized NN models is that generally these models perform very well on hardware with support for vector-dot-product operations (e.g. Ethos-U, as well as a range of modern Arm Cortex processors).

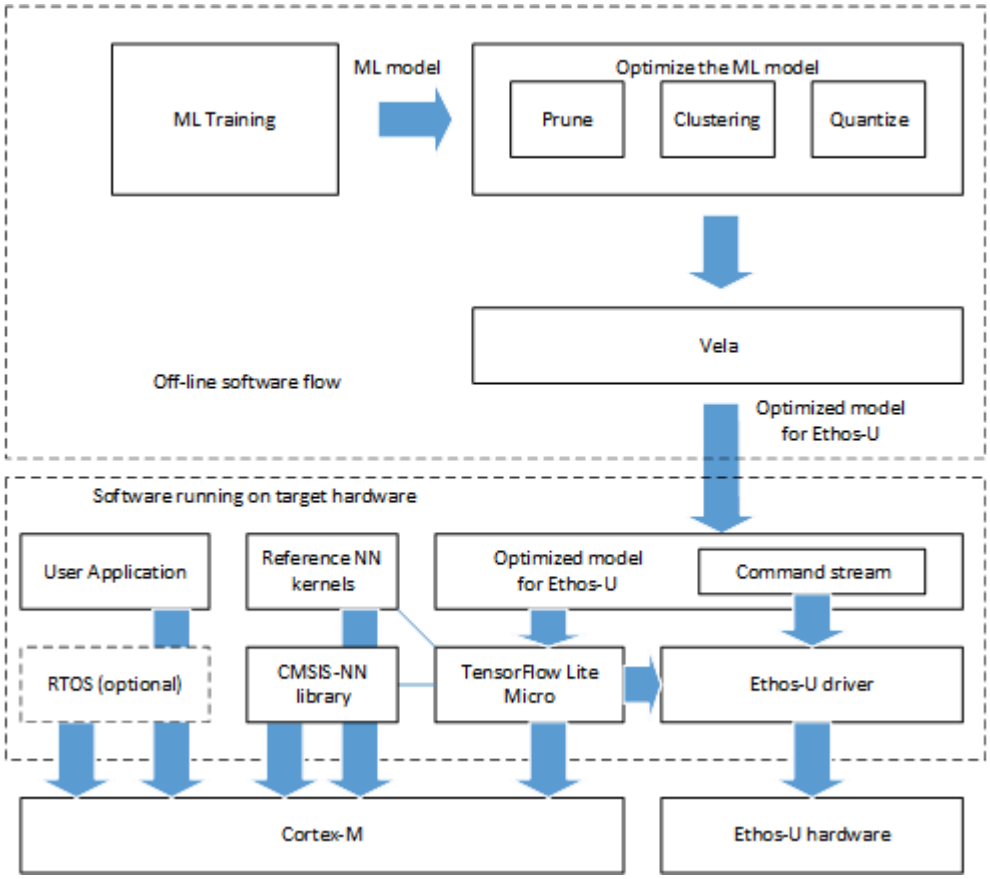
The development of a ML application can be divided into two parts:

- Off-line development flow. This consists of:
  - The preparation of the ML model
  - Quantizing the ML model to use 8-bit weight data (TF Quantization tooling - TOCO)
  - Optimizing the ML model using a tool call Vela. This tool identifies operators inside the model that can be handled by the Ethos-U and replaces them with Ethos-U functions. If an ML operator is not supported by Ethos-U but is supported by an optimized function in the CMSIS-NN library, CMSIS-NN would be used instead. The Vela compiler also handles memory layout optimizations. See Ethos-U Vela Usage for more information.
- Software components running on the target hardware. These include:

- The TensorFlow Lite Micro runtime library that takes the output from Vela (i.e. the modified .tflite file) and interprets the operations.
- The Ethos-U driver which controls the Ethos-U NPU
- The CMSIS-NN library to handle the ML operators that are not supported by Ethos-U.
- The reference C kernel of ML operators that are not supported by the Ethos-U or the CMSIS-NN library.

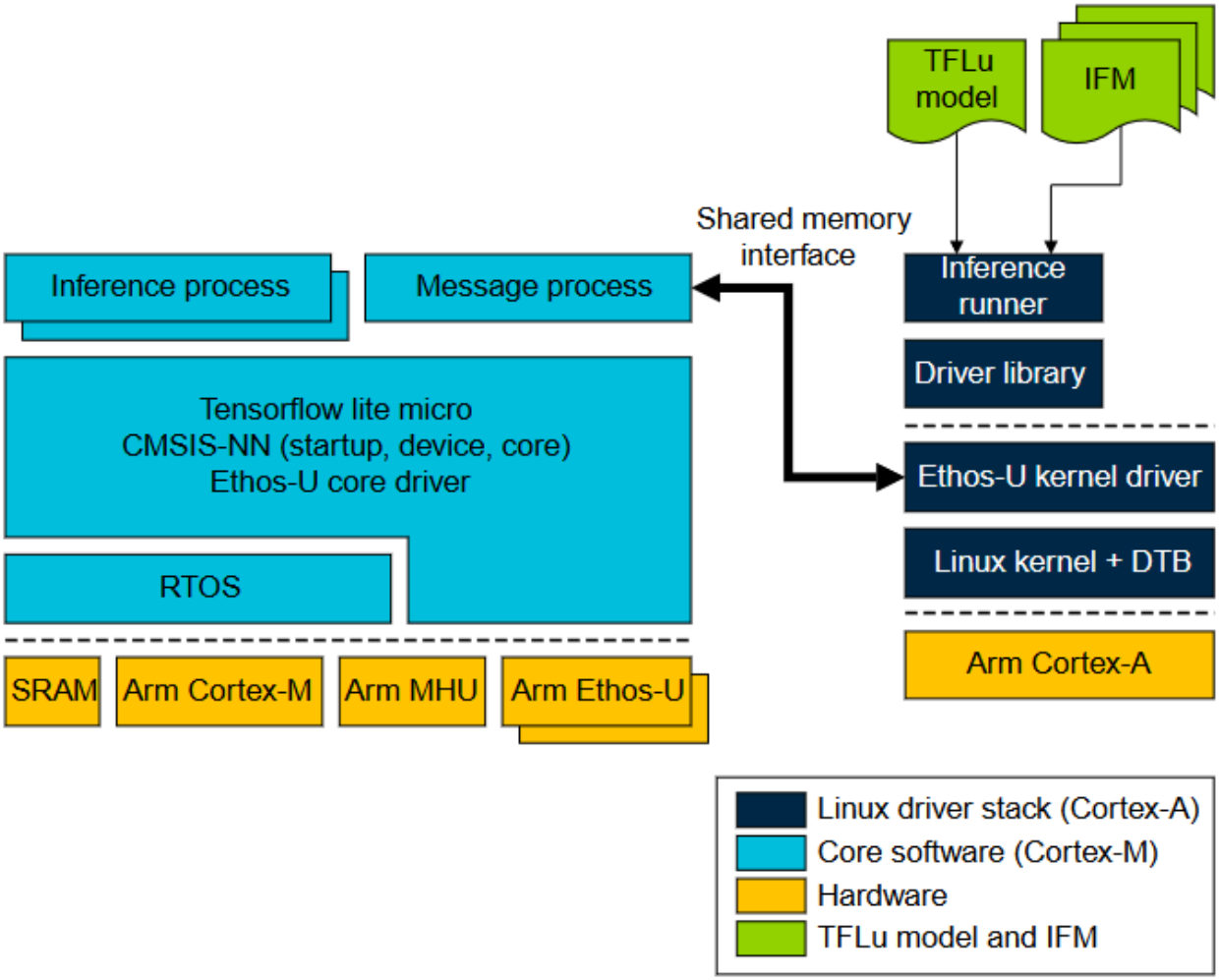
An overview of the software development flow is illustrated in the diagram below.

Figure 2-2: Software development concept



In the scenario where the Ethos-U is used in a ML subsystem within an SoC with Cortex-A processors, the applications running in the Linux environment communicates with the ML subsystem using a Ethos-U kernel driver. This driver then communicates with the Cortex-M processor via a Message Handling Unit (MHU). After receiving the information, the Cortex-M processor controls the Ethos-U NPU that carries out the inference. Additional information on this is available here.

Figure 2-3: ML subsystem software concept



An additional experimental tool called ML Inference Advisor (MLIA) is available to help developers analyze and optimize NN models on a range of Arm based hardware targets such as Ethos-U and Cortex-A processors. For more information about MLIA, see MLIA Usage.

For ML developers that are using PyTorch, there are 3rd party tools available that can convert PyTorch models to TensorFlow Lite. For example, TinyNeuralNetwork from Alibaba. Additional information about converting PyTorch to TensorFlow Lite via ONNX (Open Neural Network Exchange) is available in this blog and this GitHub page.

2.4 Differences between Ethos-U55 and Ethos-U65

There are several key differences between the Ethos-U55 and the Ethos-U65:

Feature	Ethos-U55	Ethos-U65
Number of MACs/ cycle	32/64/128/256	256 or 512
Manager Bus interface	1 or 2 64-bit AXI supporting on-chip SRAM and embedded flash	Two 128-bit AXI supporting on-chip SRAM, DRAM and flash
Host CPU support	Cortex-M85, Cortex-M55, Cortex-M7, Cortex-M4, Cortex-M33	Cortex-M85, Cortex-M55, Cortex-M7

Because Ethos-U65 has a wider bus interface(s) and additional hardware resources, in average it provides around 50% higher performance than the Ethos-U55. (Information source here).

For systems with DRAM/DDR, such as Cortex-A systems running Linux, the Ethos-U65 is more suitable because the bus interface is designed to support memories with longer latency.

2.5 Additional features

To enable Ethos-U NPUs to be used in a wide range of systems, a range of additional features are provided:

- Power management interface: Ethos-U55 and Ethos-U65 provide Q-channels (see AMBA 4 Low Power Interface Specification) for the management of clock and power gating. This interface connects to system level power management hardware (e.g. power control build on Arm CoreLink PCK-600 Power Control Kit).
- Security management: If an Ethos-U NPU is used in a TrustZone enabled system, software running in the Secure state can restrict the access permission from the Non-secure world. Privileged software can also control whether the Ethos-U NPU is privileged access only or can be accessed from both privileged and unprivileged software. Two hardware signals are available to define the access permission when the NPU comes out of reset. Contents of registers inside the NPU are also cleared at reset to prevent data leakage.
- Performance Monitoring Unit (PMU): The PMU supports a 48-bit cycle counter and four 32-bit event counters which can be used to measure activities inside the NPU. This allows software developers to analyze the characteristics of the NN workload and identify potential performance issues.

3. Tool Support for the Arm Ethos-U NPU

Several tools provide support for the Arm Ethos-U NPU:

- Vela

  This tool is used to compile a TensorFlow Lite for Microcontrollers neural network model into an optimised version that can run on an embedded system containing an Arm Ethos-U NPU.

  In order to be accelerated by the Ethos-U NPU the network operators must be quantised to either 8-bit (unsigned or signed) or 16-bit (signed).

  The optimised model will contain TensorFlow Lite Custom operators for those parts of the model that can be accelerated by the Ethos-U NPU. Parts of the model that cannot be accelerated are left unchanged and will instead run on the Cortex-M series CPU using an appropriate kernel (such as the Arm optimised CMSIS-NN kernels).

  For more information, see Ethos-U Vela Usage.
- MLIA - Machine Learning Inference Advisor

  The ML Inference Advisor (MLIA) is used to help AI developers design and optimize neural network models for efficient inference on Arm® targets (see supported targets) by enabling performance analysis and providing actionable advice early in the model development cycle. The final advice can cover supported operators, performance analysis and suggestions for model optimization (e.g. pruning, clustering, etc.).

  For more information, see MLIA Usage.
- Arm Virtual Hardware, VSI Interfaces for Sensors/Audio/Video

  Arm Virtual Hardware (AVH) provides simulation models, software tooling, and infrastructure that can be integrated into CI/CD and MLOps development flows.

  The Virtual Streaming Interface (VSI) is a flexible memory-mapped peripheral that is part of Arm Fixed Virtual Platforms (FVPs). VSI is used to simulate data streaming interfaces like audio, video, and sensors, commonly used in IoT and Machine-Learning applications. The system provides eight independent VSI instances that can function in parallel, allowing for multi-channel input/output interfaces.

  For more information, see Arm Virtual Hardware Usage.
- SDS Framework

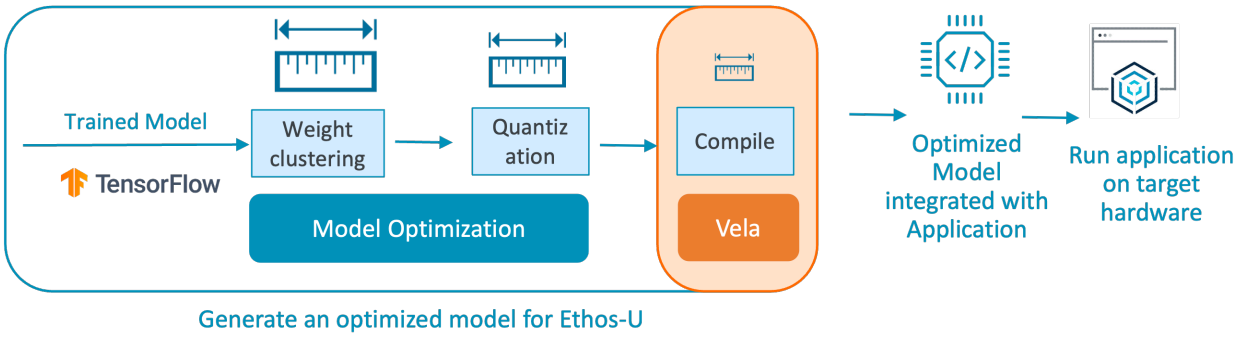
  The Synchronous Data Stream (SDS) Framework implements a data stream management, provides methods and helper tools for developing and optimizing embedded applications that integrate DSP and ML algorithms. This framework relates to the Compute Graph streaming that is part of the CMSIS-DSP compute library.

  For more information, see SDS Framework Usage.

3.1 Ethos-U Vela Usage

Ethos-U Vela is a software tool developed by Arm that serves as a performance analyzer and compiler for the Ethos-U NPU architecture. It accepts TensorFlow Lite models as input, applies quantization and layer fusion techniques, and generates a compiled binary that's specifically optimized for the Ethos-U architecture, maximizing the hardware's features for efficient execution of machine learning workloads.

Figure 3-1: Vela Development Flow



3.1.1 Requirements

The following should be installed prior to the installation of Vela:

- Windows 10 or Linux (amd64)
- Python 3.9 or higher
- Development version containing the Python/C API header files e.g. apt install python3.9-dev or yum install python39-devel
- A C99 capable compiler and associated toolchain
- For Linux operating systems, a GNU toolchain is recommended.
- For Microsoft Windows 10, Microsoft Visual C++ 14.2 Build Tools is recommended. See https://wiki.python.org/moin/WindowsCompilers

3.1.2 Installation

To install ethos-u-vela, use the following command:

```
pip3 install ethos-u-vela
```

In order for ethos-u-vela to work with an older version of NumPy that uses different C APIs, you will need to install the desired NumPy version first, and then build ethos-u-vela with that specific NumPy version. The workaround build instruction will be:

```
pip uninstall ethos-u-vela
pip install numpy==1.21.3 --force
pip install "setuptools_scm[toml]<6" wheel
pip install ethos-u-vela --no-build-isolation --no-cache-dir
```

3.1.3 Invocation

Vela runs with an input `.tflite` or `.tosa` (EXPERIMENTAL) file passed on the command line. This file contains the neural network to be compiled. The tool then outputs an optimised `.tflite` file with a `_vela` suffix in the file name, along with performance estimate (EXPERIMENTAL) CSV files, all to the output directory. It also prints a performance estimation summary back to the console, see Vela Performance Estimation Summary.

```
Neural network model compiler for Arm Ethos-U NPUs

positional arguments:
  NETWORK               Filename of the input TensorFlow Lite for Microcontrollers
 network

options:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  --api-version         Displays the version of the external API.
  --supported-ops-report
                        Generate the SUPPORTED_OPS.md file in the current working
 directory and exit
  --list-config-files   Display all available configurations in the `config_files`
 folder and exit. To select config
                        file, use the --config argument with one of the listed
 config files (For example: --config
                        Arm/vela.ini )
  --output-dir OUTPUT_DIR
                        Output directory to write files to (default: output)
  --enable-debug-db     Enables the calculation and writing of a network debug
 database to output directory
  --config CONFIG       Vela configuration file(s) in Python ConfigParser .ini file
 format
  --verbose-all         Enable all verbose options
  --verbose-config      Verbose system configuration and memory mode
  --verbose-graph       Verbose graph rewriter
  --verbose-quantization
                        Verbose quantization
  --verbose-packing     Verbose pass packing
  --verbose-tensor-purpose
                        Verbose tensor purpose
  --verbose-tensor-format
                        Verbose tensor format
  --verbose-schedule    Verbose schedule
  --verbose-allocation  Verbose tensor allocation
  --verbose-high-level-command-stream
                        Verbose high level command stream
  --verbose-register-command-stream
                        Verbose register command stream
  --verbose-operators   Verbose operator list
  --verbose-weights     Verbose weights information
  --verbose-performance
                        Verbose performance information
  --verbose-progress    Verbose progress information
```

```
  --show-cpu-operations
                        Show the operations that fall back to the CPU
  --timing              Time the compiler doing operations
  --force-symmetric-int-weights
                        Forces all zero points to 0 for signed integer weights
  --accelerator-config {ethos-u55-32,ethos-u55-64,ethos-u55-128,ethos-u55-256,ethos-
u65-256,ethos-u65-512}
                        Accelerator configuration to use (default: ethos-u55-256)
  --system-config SYSTEM_CONFIG
                        System configuration to select from the Vela configuration
 file (default: internal-default)
  --memory-mode MEMORY_MODE
                        Memory mode to select from the Vela configuration file
 (default: internal-default)
  --tensor-allocator {LinearAlloc,Greedy,HillClimb}
                        Tensor Allocator algorithm (default: HillClimb)
  --show-subgraph-io-summary
                        Shows a summary of all the subgraphs and their inputs and
 outputs
  --max-block-dependency {0,1,2,3}
                        Set the maximum value that can be used for the block
 dependency between npu kernel operations
                        (default: 3)
  --optimise {Size,Performance}
                        Set the optimisation strategy. The Size strategy results in
 minimal SRAM usage (does not use
                        arena-cache-size). The Performance strategy results in
 maximal performance (uses the arena-
                        cache-size if specified) (default: Performance)
  --arena-cache-size ARENA_CACHE_SIZE
                        Set the size of the arena cache memory area, in bytes. If
 specified, this option overrides the
                        memory mode attribute with the same name in a Vela
 configuration file
  --cpu-tensor-alignment CPU_TENSOR_ALIGNMENT
                        Controls the allocation byte alignment of cpu tensors
 including Ethos-U Custom operator inputs
                        and outputs (default: 16)
  --recursion-limit RECURSION_LIMIT
                        Set the recursion depth limit, may result in RecursionError
 if too low (default: 1000)
  --hillclimb-max-iterations HILLCLIMB_MAX_ITERATIONS
                        Set the maximum number of iterations the Hill Climb tensor
 allocator will run (default: 99999)
```

Detailed explanation of the parameters can be found in the Configuration Line Interface Reference.

3.1.4 Command Examples

In most cases all parameters required will be configured via the `vela.ini` config file. In this case the call will look like this:

```
vela --config vela.ini my_model.tflite
```

3.1.5 vela.ini File Example

Detailed explanation of the options can be found in the Configuration File Reference

```
; Ethos-U55 High-End Embedded: SRAM (4 GB/s) and Flash (0.5 GB/s)
[System_Config.Ethos_U55_High_End_Embedded]
```

```
core_clock=500e6
axi0_port=Sram
axi1_port=OffChipFlash
Sram_clock_scale=1.0
Sram_burst_length=32
Sram_read_latency=32
Sram_write_latency=32
OffChipFlash_clock_scale=0.125
OffChipFlash_burst_length=128
OffChipFlash_read_latency=64
OffChipFlash_write_latency=64
```

3.2 MLIA Usage

MLIA (Machine Learning Inference Advisor) is currently an experimental software tool and is provided as-is, without any guarantees or warranties of its functionality, reliability, or suitability for any specific purpose

3.2.1 Requirements

It is recommended to use a virtual environment for MLIA installation, and a typical setup for MLIA requires:

- Ubuntu® 20.04.03 LTS
- Python® 3.8.1 or higher
- Ethos™-U Vela

3.2.2 Installation

To install MLIA, run the following command:

```
pip install mlia
```

3.2.3 Invocation

MLIA usage is as follows:

```
usage: mlia [-h] [-v] {check,optimize} ...

ML Inference Advisor 0.6.0

Help the design and optimization of neural network models for efficient inference on
 a target CPU or NPU.

Supported Targets/Backends:
```

Target	Backend(s)	Status	Advice: comp/perf/opt
Cortex-A	Arm NN TensorFlow Lite	BUILTIN	YES/NO/NO

```
| <cortex-a>     | delegate                  |                 |                    |
|                | <armnn-tflite-delegat…    |                 |                    |
|----------------|---------------------------|-----------------|--------------------|
| Ethos-U55      | Vela                      | BUILTIN         | YES/YES/YES        |
| <ethos-u55>    | <vela>                    |                 |                    |
|                | Corstone-310              | NOT INSTALLED   |                    |
|                | <corstone-310>            |                 |                    |
|                | Corstone-300              | NOT INSTALLED   |                    |
|                | <corstone-300>            |                 |                    |
|----------------|---------------------------|-----------------|--------------------|
| Ethos-U65      | Vela                      | BUILTIN         | YES/YES/YES        |
| <ethos-u65>    | <vela>                    |                 |                    |
|                | Corstone-310              | NOT INSTALLED   |                    |
|                | <corstone-310>            |                 |                    |
|                | Corstone-300              | NOT INSTALLED   |                    |
|                | <corstone-300>            |                 |                    |
|----------------|---------------------------|-----------------|--------------------|
| TOSA           | TOSA Checker              | NOT INSTALLED   | YES/NO/NO          |
| <tosa>         | <tosa-checker>            |                 |                    |
+----------------+---------------------------+-----------------+--------------------+
Comp/Perf/Opt: Advice categories compatibility/performance/optimization
Use command 'mlia-backend' to install backends.

options:
  -h, --help            Show this help message and exit
  -v, --version         Show program's version number and exit

Commands:
  {check,optimize}
    check               Generate a full report on the input model
    optimize            Show the performance improvements (if any) after applying
                        the optimizations
```

3.2.4 Command Examples

A very typical tool invocation for Ethos-U on a Corstone-3xx based device is shown here. You can run `mlia -h` first, validating that you have the necessary backend(s) installed.

```
# Explicitly specify the target profile and backend(s) to use
# with --backend option
mlia check ~/models/ds_cnn_large_fully_quantized_int8.tflite \
    --target-profile ethos-u55-256 \
    --performance \
    --backend "vela" \
    --backend "corstone-300"
```

3.3 Arm Virtual Hardware Usage

For detailed information on AVH's capabilities and how to utilize them effectively, refer to the Getting Started Guide. This guide provides a comprehensive walkthrough of the setup process, explains the navigational structure of the AVH platform, and offers essential technical details to maximize the utility of the system.

The Getting Started Guide provides a step-by-step guide through the Continuous Integration (CI) workflow operation and its setup using Arm Virtual Hardware (AVH). Here's an overview of its contents:

Overview: Overview of the common steps in the CI workflow, including local development using a toolchain such as Keil MDK and Arm Fixed Virtual Platforms, setup of a CI pipeline using GitHub Actions, automated program build and testing in the cloud with AVH, and failure analysis and local debugging.

Prerequisites: The guide outlines the prerequisites required to reproduce the operation of the example project.

Develop tests: It introduces the concept of developing unit tests using the Unity Framework. Includes an example project.

- Forking the Getting Started GitHub repository: The guide explains the process of creating a GitHub repository by either creating a new one or forking from the AVH GetStarted repository.
- Setup local project on your PC: It provides instructions on setting up the local project on your PC. This includes cloning the repository onto your local PC and setting up the project in Keil MDK.
- Implement tests: The guide outlines how to implement tests using the Unity Framework and how to redirect standard output to be visible during the debug session1.
- Build and Run the example in Keil MDK: It provides steps on how to build and execute the program in Keil MDK. It also explains how to export the project for use in command-line CI environments.

Setup CI pipeline: The guide provides information on setting up the CI pipeline which is triggered on every code change via push and pull requests. The CI implementation in the example is implemented with GitHub Actions.

AWS setup: The guide explains the process of setting up AWS to enable the execution of the example CI pipeline on cloud-hosted Arm Virtual Hardware instance.

GitHub Actions setup: It introduces the concept of running Arm Virtual Hardware with GitHub Actions.

3.4 SDS Framework Usage

For effective ML algorithm selection, training, and validation a large set of representative and qualified data is a pre-requisite. As explained under section “III. Development Process” correct decisions can only be done when sufficient training data exist. ML algorithms can be introduced in a step-by-step approach where first a classic algorithm is deployed, and scenarios of interest are captured. Having a deployed fleet of IoT endpoint devices that captures such data is therefore invaluable.

To capture real-world data from sensors and audio sources, Arm developed the Synchronous Data Stream (SDS) Framework. It consists of several components:

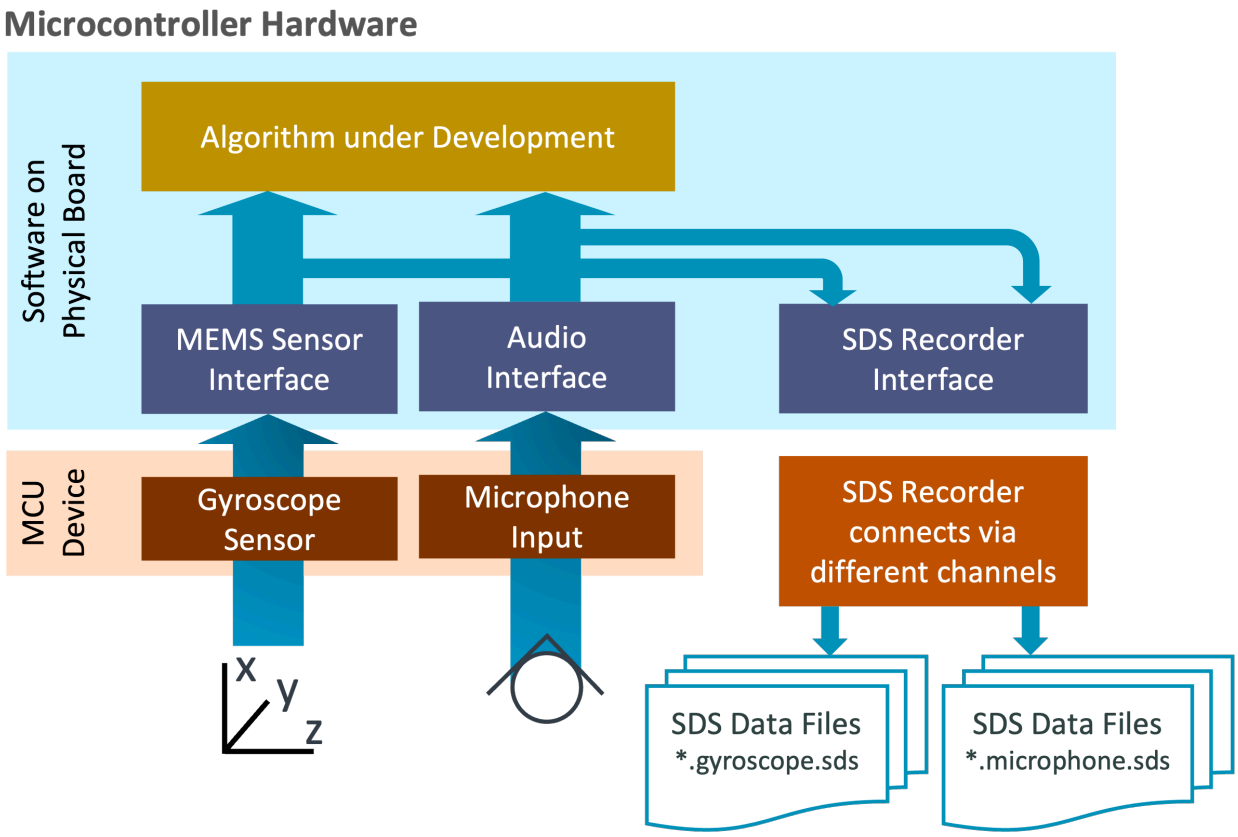
- SDS Recorder Interface: provides methods to record real-world data in SDS data files for analysis and development of DSP and ML algorithms.

- SDS Metadata: describe the content of SDS data files along with scaling and formatting information.
- SDS Utilities: tools to record, convert, and display SDS data files.
- SDS Playback: Using Arm Virtual Hardware with the Virtual Streaming Interfaces (VSI) allows to stimulate an algorithm under development with real-world data.

3.4.1 SDS Recorder Interface

The SDS Recorder Interface is designed for recording real-world data in SDS data files. The SDS Recorder is an integral part of the target application and enables data streaming via various interfaces such as TCP/IP via Ethernet, UART, or USB. It can also be used to capture data in a deployed IoT endpoint device to report situations where the current ML model has gaps in the training data.

Figure 3-2: SDS Recorder Interface



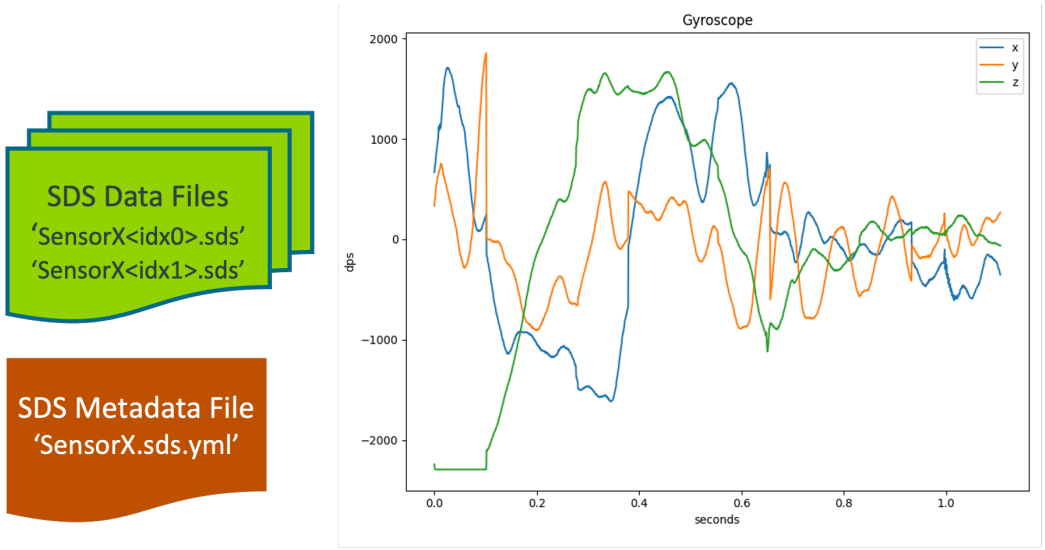
The SDS Recorder Interface is used to caputure real-world data with the sensors of the final hardware target.

SDS connects via different channels and can be also embedded in deployed IoT endpoint devices.

3.4.2 SDS Metadata

The SDS Metadata file provides information about the content of SDS data files. This metadata information is used to display meaningful information to the user. It also identifies the data streams for inputs to DSP design utilities, MLOps development workflows, or AVH data playback.

Figure 3-3: SDS Metadata File

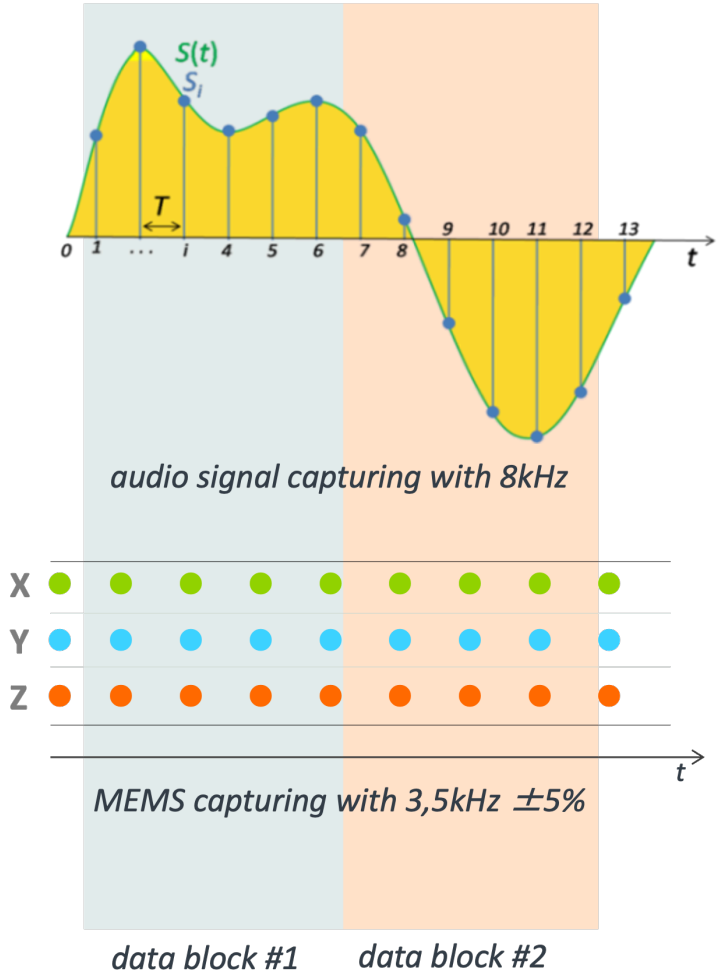


The SDS metadata file provides information about the content of SDS data files; used to diplay meaningful information and provide context for MLOps systems.

The SDS data files have multiple use cases such as:

- Input to Digital Signal Processing (DSP) development tools such as filter designers
- Input to Machine Learning (ML) model classification, training, and performance optimization.
- Verify execution of DSP algorithm on Cortex-M targets with off-line tools.
- Playback real-world SDS data files for algorithm validation using Arm Virtual Hardware.

Figure 3-4: SDS Sample Frequencies



Sensors may have independent clock sources with tolerances which results in different size records for block procession algorithms.

The SDS Recorder is a flexible software component that can be connected to various output channels. The table below shows the different communication speeds that can be achieved depending on the output channel.

Development Board	Output Channel	Measured speed	Comment
NXP IMXRT1050-EVKB	TCP/IP via Ethernet	2 MB/s	
NXP IMXRT1050-EVKB	File System	2.85 MB/s	MicroSD card
NXP IMXRT1050-EVKB	VCOM (High-Speed)	11.8 MB/s	
ST B-U585I-IOT02A	VCOM (Full-Speed)	600 kB/s	
ST B-U585I-IOT02A	UART	80 kB/s	Baud Rate: 921600

Refer to SDS Recorder in the GitHub repository for access to examples.

3.4.3 SDS Utilities

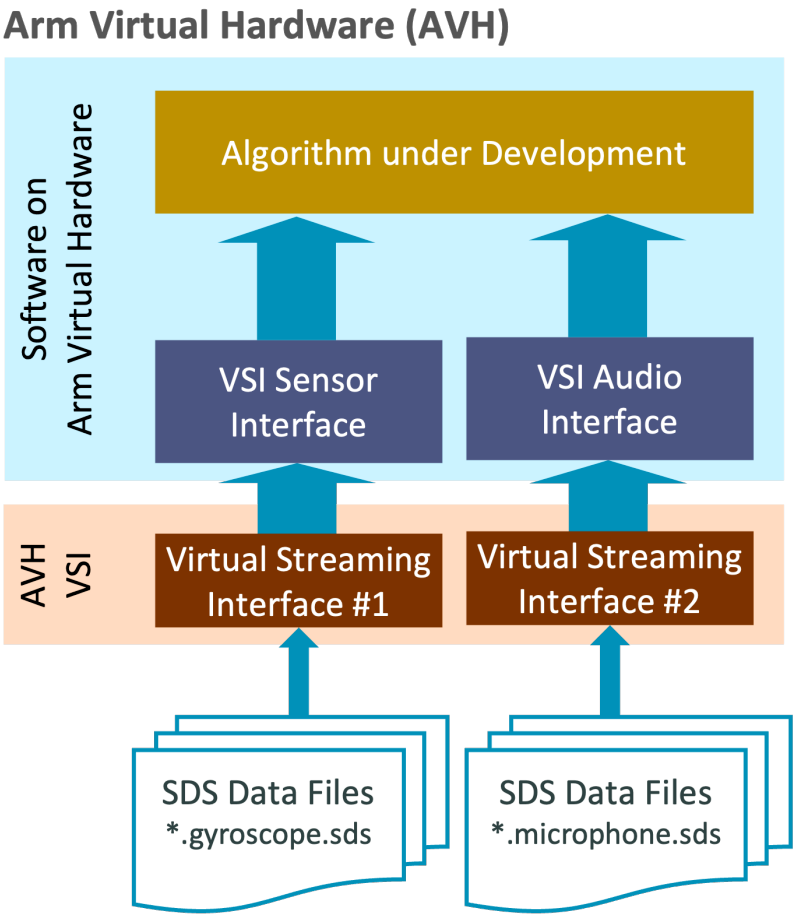
These utilities allow to analyze and convert the SDS data files.

Refer to SDS Utilities in the GitHub repository for more information and access to these utilities.

3.4.4 SDS Playback

Arm Virtual Hardware (AVH) is available in multiple deployments such as GitHub, Qeexo AutoML, Keil Studio Cloud, and AWS AMI for flexible cloud access. In the desktop version of Keil MDK it supports test case development and verification of algorithms. With DevOps systems such as GitHub actions it supports continuous integration workflows for build and test automation. SDS data files can be used as input during validation tests. AVH also supports A/B comparisons of ML algorithms and helps therefore to select the best matching algorithm for an application. As it is part of some MLOps systems, it helps to validate ML models before deploying it to physical hardware devices.

Figure 3-5: SDS Arm Virtual Hardware



AVH provides virtual streaming interfaces that can be used to playback SDS data to an algorithm under development; useful for repeatable validation tests or ML model verification. The SDS framework is designed to support also sensor fusion applications where the data of multiple different sources is combined. It might be useful to combine an audio signal with the information of a MEMS sensor to provide better prediction for machine failures. However, when combining data from multiple sources, tolerances of independent clock sources should be considered. The SDS framework has provisions to cope with such situations and provides independent clock information for multiple data streams.

Refer to SDS Playback in the GitHub repository for more information.

4. The Arm ML Zoo

Creating a new Neural Networks(NN) model for an application requires in depth understanding in Machine Learning (ML), and it can take very long time to optimize the network for specific use case. Fortunately, there are many different types of Neural Networks(NN) already developed for different applications and many of them are available in online databases called Model Zoo. The models in those model zoos can be used as a starting point for further development, so when an application developer creates an ML application, there is no need to create a new NN model from scratch.

The ARM ML Zoo repository hosts a variety of machine learning models optimized for ARM IP. The models cover different applications such as Anomaly Detection, Image Classification, Keyword Spotting, Noise Suppression, Object Detection, Speech Recognition, Superresolution, and Visual Wake Words.

Here's a brief overview:

1. Anomaly Detection: This includes three MicroNet models (Large, Medium, Small) that are optimized for INT8 and are compatible with TensorFlow Lite. They are particularly suitable for Cortex-M, Mali GPU, and Ethos U.
2. Image Classification: This category offers MobileNet v2 models optimized for INT8 and UINT8. They are compatible with TensorFlow Lite and are suitable for Cortex-A, Cortex-M, Mali GPU, and Ethos U.
3. Keyword Spotting: This includes CNN models (Large, Medium, Small), DNN models (Large, Medium, Small), and DS-CNN models (Large Clustered in FP32 and INT8, Large in INT8). They are optimized for INT8 or FP32 and are compatible with TensorFlow Lite. They are suitable for Cortex-A, Cortex-M, Mali GPU, and Ethos U, with some variations depending on the model.
4. Noise Suppression
5. Object Detection
6. Speech Recognition: This includes Wav2letter models (standard and pruned) and Tiny Wav2letter models (standard and pruned). They are optimized for INT8 and are compatible with TensorFlow Lite. They are suitable for Cortex-A, Cortex-M, Mali GPU, and Ethos U.
7. Superresolution: This includes the SESR model optimized for INT8 and compatible with TensorFlow Lite. It is suitable for Cortex-A and Mali GPU.
8. Visual Wake Words: This includes three MicroNet models (VWW-2, VWW-3, VWW-4) optimized for INT8 and compatible with TensorFlow Lite. They are particularly suitable for Cortex-M, Mali GPU, and Ethos U.

In most cases, software developers still need to retrain the ML model to fit the specific needs in their ML applications.

4.1 Integrate a ML-Zoo model

The content for this section is currently under development and will be added in the near future.

5. ML Embedded Evaluation Kit

The Arm ML Evaluation Kit is a tool designed to help developers build and deploy machine learning applications for Arm Cortex-M55 and Arm Ethos-U55 NPU. It provides ready-to-use software applications for Ethos-U55 systems, including image classification, keyword spotting, automated speech recognition, anomaly detection, and person detection.

The kit allows developers to evaluate the performance metrics of networks running on the Cortex-M CPU and Ethos-U NPU. It also includes a generic inference runner that can be used to develop custom ML applications for Ethos-U. The kit is based on the Arm Corstone-300 reference package, which is designed to help SoC designers build secure systems faster. The platform is available as Ecosystem FPGA (MPS3) and Fixed Virtual Platform (FVP) to allow development ahead of hardware availability.

5.1 ML Embedded Evaluation Kit - Quick Start

There are a range of options when trying out the Ethos-U NPUs.

- ML Embedded Evaluation Kit (Ethos-U ML Evaluation Kit examples): This software environment allows you to build and try out Ethos-U projects for a selection of platforms. These include Fixed Virtual Platforms and FPGA. This software environment runs in a Linux environment (it is possible to use Windows Subsystem for Linux).
- CMSIS-Pack based Machine Learning Examples - This contains examples that use CMSIS-Pack to handle software integration. This setup can be used in both Linux and Windows.

5.1.1 Using the ML Embedded Evaluation Kit

The ML Embedded Evaluation Kit provides examples based on the applications listed on this page.

A Quick Start Guide is available for the ML Embedded Evaluation Kit. A summary of the guide is as follows.

5.1.1.1 Supported platforms

In this section, we will discuss the various platforms that are compatible with the Machine Learning Evaluation Kit. These platforms range from physical hardware like the Arm MPS3 FPGA board to virtual environments such as the Fixed Virtual Platform (FVP) and Arm Virtual Hardware (AVH). Each of these platforms offers unique capabilities and advantages, making the ML Evaluation Kit a versatile tool for a variety of machine learning explorations.

- Arm MPS3 FPGA board with one of the following FPGA images:
  - AN552 FPGA image - This is based on the Corstone-300 subsystem containing Arm Cortex-M55 processor and the Ethos-U55 NPU.

    - AN555 FPGA image - This is based on the Corstone-310 subsystem containing Arm Cortex-M85 processor and the Ethos-U55 NPU.

  The FPGA images can be downloaded from here.
- Fixed Virtual Platform (FVP) for Corstone-300
- Arm Virtual Hardware (AVH) for Corstone-300
- Arm Virtual Hardware (AVH) for Corstone-310

5.1.1.2 System and software requirements

The prerequisites required is available here. A summary of the steps required is as follows:

To use the ML embedded evaluation kit, you need to use a x86 Linux system (or Windows Subsystem for Linux) with a recent release of Python3 (currently Python version 3.9 or newer is required). If the Python version on your system is 3.8 or earlier, you can update the Python version by following the instructions here. You can check the version of Python3 in your system using:

```
python3 --version
```

You should then install a number of software tools using the following commands:

```
sudo apt install -y cmake make python3 git curl unzip xxd
sudo apt install -y python3-pip
python3 -m pip install pillow
```

Next, install Python virtual environment. If using python 3.9:

```
sudo apt install -y python3.9-venv
```

If using python 3.10, install Python virtual environment support using:

```
sudo apt install -y python3.10-venv
```

You also need to have a compilation tool chain, either:

- Arm Compiler for Embedded 6.16 or later, or
- GNU Arm Embedded toolchain 10.2.1 or later

Information about the Arm Compiler for Embedded can be found here. To use Arm Compiler for Embedded, you need a valid license.

If you need to install the GNU Arm Embedded toolchain, please download a suitable version from the following links:

- https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads (for versions 11.2, 11.3 12.2 or later)

- https://developer.arm.com/downloads/-/gnu-rm (for version 10.3 or earlier)

Do not install the GNU Arm Embedded toolchain 10.3.1 20210621 using "sudo apt install gcc-arm-none-eabi". There is a known issue for that release that results in a compilation error in the ML embedded evaluation kit.

After the toolchain is installed, you need to add the toolchain binary path to the search path.

For users running Windows Subsystem for Linux (WSL): Windows' paths in the $PATH variable might contain unescaped space characters (e.g. "/mnt/c/Program Files/Microsoft VS Code/bin"). This causes problems to the build script (see details here). To solve the problem, you can use one of the following workarounds:

- create a bash script that removes the offending Windows paths in the path variable, or
- add escape character \ in the path variable, or
- enclose the paths with quotes.

5.1.1.3 Check out the repository

The following commands should be used to check out the repository:

```
git clone "https://review.mlplatform.org/ml/ethos-u/ml-embedded-evaluation-kit"
cd ml-embedded-evaluation-kit
git submodule update --init
```

5.1.1.4 Compiling the default projects

Once the toolchain and utilities are installed and the search paths are in place, the compilation script can then be run:

If using GNU Arm Embedded toolchain (gcc):

```
python3 ./build_default.py
```

If using Arm Compiler 6:

```
python3 ./build_default.py --toolchain arm
```

After the compilation, the executables are located in cmake-build-XXXXXX/bin.

To run the example projects, follow the instructions in this page: Deployment.

5.1.2 Command line options for the default build script

In the previous step we executed a script (build_default.py) to build the tests for the default configuration. The script supports additional command line options. For example, to specify the configuration of the Ethos-U55 hardware to be 32 MAC, use one of the following:

```
./build_default.py --npu-config-name ethos-u55-32 # For gcc
```

or

```
./build_default.py --npu-config-name ethos-u55-32 --toolchain arm # for Arm Compiler
```

The list of valid options for the Ethos-U configurations are:

- ethos-u55-32
- ethos-u55-64
- ethos-u55-128
- ethos-u55-256
- ethos-u65-256
- ethos-u65-512

Additional command line options are described here.

5.1.3 Documentation

There are a number of documents available in the ML Embedded Evaluation Kit git repository:

- Home page
- Quick Start Guide
- Documentation
- Building the ML embedded code sample applications from sources
- Deployment
- Customizing (Implementing custom ML application)
- Arm Virtual Hardware
- FAQ
- Troubleshooting
- Memory considerations
- Testing and benchmark
- Timing adapter
- CMAKE presets

- Coding standards and guidelines
- Appendix

5.1.4 Additional material

Arm blogs

- Optimize a ML model for fast inference on Ethos-U microNPU
- Vela Complier: The first step to deploy your NN model on the Arm Ethos-U microNPU
- Blog: Arm ML Embedded Evaluation Kit

Learning pages

- Navigate Machine Learning development with Ethos-U processors
- Build and run the Arm Machine Learning Evaluation Kit examples

5.2 ML Evaluation kit : Under the hood

In the preceding chapters, we have introduced the ML Evaluation Kit, its purpose, and its potential applications. We have also discussed the fundamental concepts of machine learning that are crucial for understanding the workings of the kit.

Further, we provide an overview of the software components of the kit, including the TensorFlow Lite Micro runtime and the Ethos-U driver.

We explore the structure of the repository, the build process, and the key steps involved in the compilation script. We also discuss the various build options available and how they can be customized according to the user's needs.

5.2.1 More about the build process

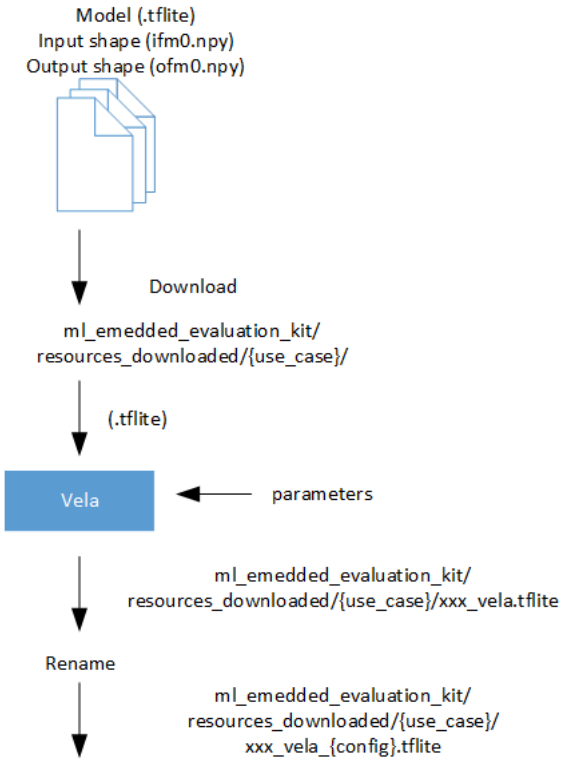
A range of materials are available to describe:

- The structure of the repository
- Build process
  - Preparing Build environment
  - Create a build directory
  - Configuring the build for the platform chosen
    - Configuring the build for MPS3 SSE-300
    - Configuring the build for MPS3 SSE-310
    - Configuring native unit-test build

The compilation script (“build_default.py”) contains the following key steps:

1. Download models for TensorFlow Lite Micro for each “use-case” and use Vela to optimize the models

Figure 5-1: Model download and optimize



2. Download additional software components, unpack them, and if needed, patch them

   The software components include:

   - FlatBuffers library from TensorFlow Lite Micro
   - kissfft (a Fast Fourier Transform library)
   - pigweed (a collection of embedded-targeted libraries)
   - gemmlowp (General Matrix Multiplication)
   - ruy (matrix multiplication library)
3. Setup build environment for each “use-case”

   This process includes setting up directories, creating the MAKEFILEs and carrying out conversion processes to enable the models and data to be included in the C++ compilation:

Input file	Conversion script
Models for TensorFlow Lite Micro (.tflite)	ml-embedded-evaluation-kit/scripts/py/gen_model_cpp.py
ML data labels (e.g. .txt)	ml-embedded-evaluation-kit/scripts/py/gen_labels_cpp.py
Audio input data (e.g. .wav)	ml-embedded-evaluation-kit/scripts/py/gen_audio_cpp.py
Image input data (e.g. .bmp)	ml-embedded-evaluation-kit/scripts/py/gen_rgb_cpp.py

The output of the converted C++ source and header files can be found in ml-embedded-evaluation-kit/cmake-build-{target}-{configs}/generated/{use-case}/

Please note that it is also possible to use the xxd utility to convert .tflite model files to byte array in C/C++.

4. Build the executables

   The source codes are compiled at this stage. If needed, additional software components (e.g. CMSIS-DSP library source) are downloaded.

5.2.2 Build options for build_default.py

The list of build options is available on this page. For example, you can use the following ETHOS_U_NPU_MEMORY_MODEL options to define the memory types used by Ethos-U NPU:

- Shared_Sram (default for Ethos-U55 NPU, available to both Ethos-U55 & Ethos-U65)
- Dedicated_Sram (default for Ethos-U65 NPU, available to Ethos-U65 only)
- Sram_Only (available for Ethos-U55 only)

If it is necessary to use different build options, instead of executing the build steps manually, it is easier to use the default build script as a starting point and modify the “cmake_command” in the script to use customized build options.

5.2.3 Software components

This section explores the software components of the Machine Learning Evaluation Kit. Key components include TensorFlow Lite Micro, which provides the runtime for the executable program, the Ethos-U Driver, and platform codes that offer flexibility to support multiple hardware targets.

5.2.3.1 TensorFlow Lite Micro

The executable program image contains the TensorFlow Lite Micro runtime, which is downloaded as part of the "build_default.py" script execution. After being downloaded, the files are located in this location: ml-embedded-evaluation-kit/dependencies/tensorflow.

The TensorFlow Lite Micro kernel provides support for Ethos-U. Some useful information about software integration and initialization is documented in ml-embedded-evaluation-kit/dependencies/tensorflow/tensorflow/lite/micro/kernels/ethos_u/README.md

In order to invoke the TensorFlow Lite Micro interpreter, the application code must include several header files and must also setup the model before invoking the micro-interpreter. To understand the bare minimal code required to setup and execute TensorFlow Lite Micro, please visit this page. It contains a good overview of TensorFlow Lite Micro low level operations.

5.2.3.2 Ethos-U Driver

The Ethos-U driver can be found in the following location: ml-embedded-evaluation-kit/dependencies/core-driver

5.2.3.3 Platform codes

The platform support codes can be a bit confusing because the evaluation kit contains files from multiple repositories, and some of them have their own platform driver codes. The ML Embedded Evaluation Kit did not use the driver code from the 3rd party repository because it needed extra flexibility to support multiple hardware targets (e.g. Corstone-300, Corstone-310). As a result, platform support codes can be found in the following locations:

Location	Note
ml-embedded-evaluation-kit/source/hal/source/platform/mps3/	This is the one that is actually used in the example projects
ml-embedded-evaluation-kit/dependencies/tensorflow/tensorflow/lite/micro/cortex_m_corstone_300	From Google TensorFlow - Not used
ml-embedded-evaluation-kit/dependencies/core-platform/targets/corstone-300	From MLplatform.org - Not used

Details of Ethos-U integration (e.g. Base address, IRQ assignment) can be found in this file: ml-embedded-evaluation-kit/source/hal/source/platform/{platform}/CMakeLists.txt. The linker scripts can be found in this location: ml-embedded-evaluation-kit/scripts/cmake/platforms/{platform}.

5.2.4 Creating custom applications in the ML Embedded Evaluation Kit

The page Implementing custom ML application contains information about creating custom applications running in the ML Embedded Evaluation Kit. It is also possible to add custom platform support.

6. CMSIS-Pack Based Machine Learning Examples

In addition to the ML Embedded Evaluation Kit, which provides a quick path for users when trying out the Ethos-U55/U65 NPUs, Arm is also working on a new set of examples based on the CMSIS-Pack, a software component packaging solution. The CMSIS-Pack based ML examples are available at this github location. The examples in this GitHub repository:

- Enables the use of the CMSIS-Pack as a software integration mechanism - which is more suitable for IDE development environments.
- Provides a greater choices of hardware support.
- Can be used in Windows environments.

Please note that the CMSIS-Pack based ML examples repository is work-in-progress and currently there are some limitation(s). These are:

- The ML "use-cases" are limited to Key Word Spotting (KWS) and Object Detection.
- Currently only Arm Compiler 6 is supported.
- Currently CMSIS-Toolbox 2.0 is not supported. Please use CMSIS-Toolbox 1.7. (You can find the installation files for version 1.7 here)

In addition to Arm Compiler 6 and the CMSIS-Toolbox, the following tools are required:

- git, cmake, make, ninja

In a Linux system, the tools can be installed using the following command:

```
sudo apt install -y cmake make git ninja-build
```

In a Windows system, you will need to install the utilities using installers from the following websites:

Utility	website
git	https://git-scm.com/download/win
cmake	https://cmake.org/download/
GNU make	http://gnuwin32.sourceforge.net/packages/make.htm
Ninja-build	https://github.com/ninja-build/ninja/releases

It is also recommended that Visual Studio Code IDE with Keil Studio Pack Extension should be used. Alternatively, Keil Studio Cloud can also be used.

6.1 CMSIS-Toolbox setup

If you have already setup the CMSIS-Toolbox for your development environment, you can skip this section.

6.1.1 CMSIS-Toolbox version 1.7 only

To install the CMSIS-ToolBox, please download release 1.7 from this page. Installation details can be found on this page.

After installation, you need to ensure that:

- The CMSIS-Toolbox binaries are in the search path.
- The cmake template file for Arm Compiler 6 (This can be found in{cmsis_toolbox_install_path}/etc/AC6.6.18.0.cmake) contains the correct path in the "TOOLCHAIN_ROOT" variable using the "set(TOOLCHAIN_ROOT {path})" command. E.g. set(TOOLCHAIN_ROOT "C:/Keil_v5/ARM/ARMCLANG/bin")
    - Note: Starting from CMSIS-Toolbox 2.0, the preferred way to define the toolchain installation location is to use environmental variables. For example, the command "set AC6_TOOLCHAIN_6_19_0=C:/Keil_v5/ARM/ARMCLANG/bin" sets up an environment variable that indicates that Arm Compiler 6.19 is installed in "C:/Keil_v5/ARM/ARMCLANG/bin".
- The environment variable CMSIS_PACK_ROOT is pointing to the CMSIS-Pack Root directory that stores the software packs.
- The CMSIS-Pack Root directory has been initialized. See the cpackget page for details. (e.g. cpackget init –pack-root path/to/new/pack-root https://www.keil.com/pack/index.pidx)
- The environment variable CMSIS_COMPILER_ROOT is pointing to the etc directory in the CMSIS-Toolbox (e.g. {install_path}/etc)

6.1.2 CMSIS-Toolbox version 2.0 (Not supported in the current release of the CMSIS-Pack based ML Examples)

To install the CMSIS-ToolBox, please download release 2.0 from this page. Installation details can be found on this page.

After installation, you need to ensure that:

- The CMSIS-Toolbox binaries are in the search path.
- The environment variables for the toolchain installation path is set. For example: set AC6_TOOLCHAIN_6_19_0=C:/Keil_v5/ARM/ARMCLANG/bin
- The environment variable CMSIS_PACK_ROOT is pointing to the CMSIS-Pack Root directory that stores the software packs.
- The CMSIS-Pack Root directory has been initialized. See the cpackget page for details. (e.g. cpackget init –pack-root path/to/new/pack-root https://www.keil.com/pack/index.pidx)

- The environment variable CMSIS_COMPILER_ROOT is pointing to the etc directory in the CMSIS-Toolbox (e.g. {install_path}/etc)

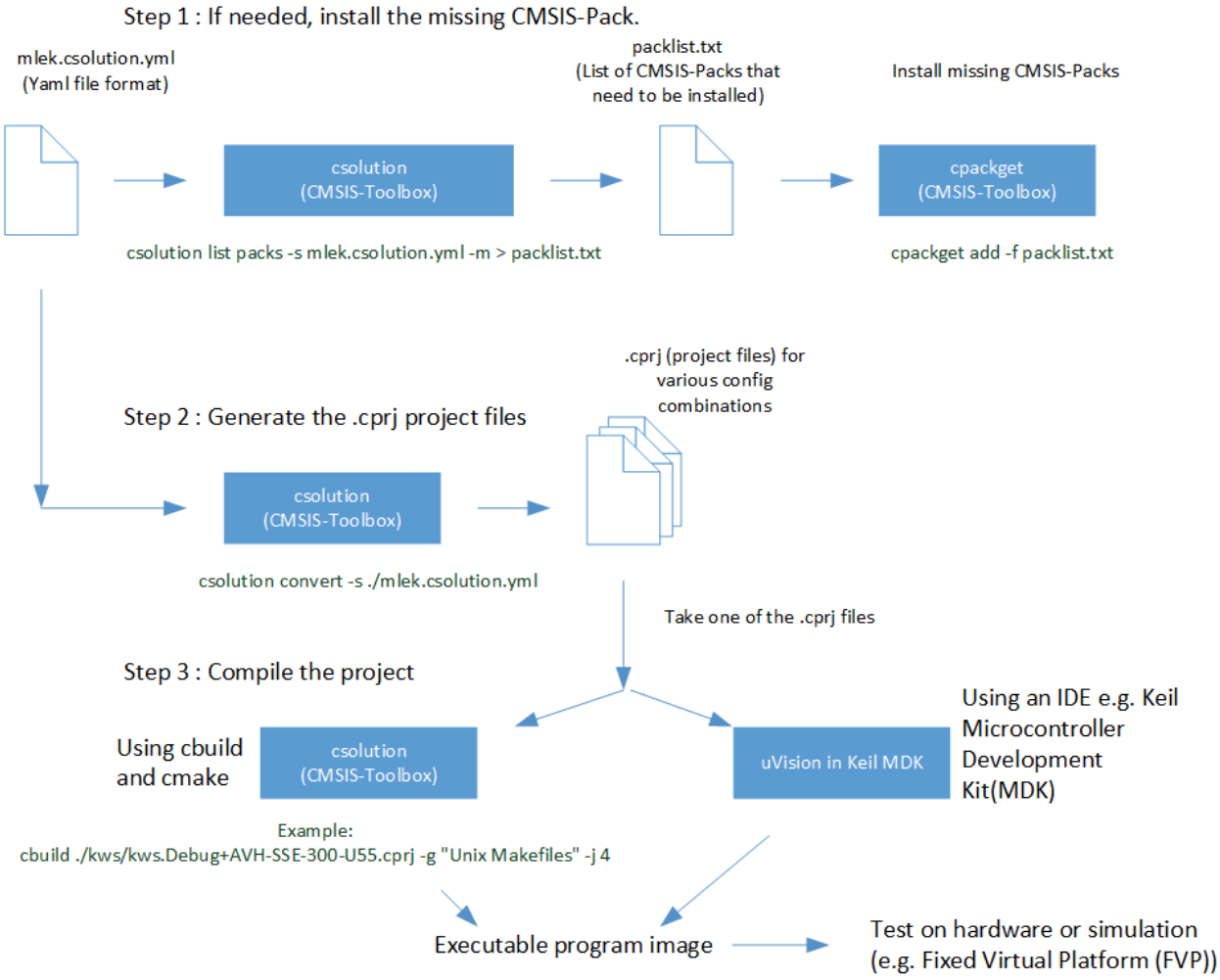
6.2 Getting Started

The workflow of the "CMSIS-Pack based ML Examples" utilizes the CMSIS-Toolbox to generate compilation setups from the Yaml files. The flow involves 3 steps:

- If some of the required software packs are not installed, install the missing CMSIS-Pack(s) .
- Generate the .cprj files from the main Yaml (.yml) file for each of the "use-cases".
- Compile the project.

The 3 steps are illustrated as below:

Figure 6-1: Concept of software workflow in the CMSIS-Pack based ML examples



By default, step 2 generates a number of .cprj files for various configuration combinations. There are many combinations because:

- The examples support a number of ML use-cases
- The examples support a number of target platforms
- The build type can be Debug or Release

You can specify to generate just one specific combination in step 2 if needed. e.g.

```
csolution convert -s ./mlek.csolution.yml -c object_detection.Release+AVH-SSE-300-U55
```

After the executable is generated, the program image can be tested on target hardware or in a simulation environment. Further information can be found here.

Unlike the ML Embedded Evaluation Kit, the ML models and input samples have already been converted to C++ (e.g. KWS, object detection). There is therefore no need to execute the Vela Compiler.

7. Project Integration of TensorFlow Lite Micro

The following section will help to integrate TensorFlow Lite Micro, Ethos-U driver and the model(s) into your project. Skip the first paragraph if you already have an existing project.

7.1 Create a new project in Visual Studio Code

A very good introduction for new users to the Visual Studio Code extensions, that Arm provides for CMSIS based projects, can be found here:

Get started with an example project

You will be guided through the tool installation and setup and getting a basic Cortex-M project to build and debug. This is the necessary requirement to continue with the integration of TensorFlow Lite micro software stack.

7.2 Add TensorFlow Lite Micro CMSIS components

The most advanced way to integrate and manage TensorFlow along the full project life-cycle is by using the CMSIS-Pack based release. This pack is build from sources that are maintained and versioned by Arm on mlplatform.org.

Integration can be done by selection components in the component management facilites that are familiar to you from your IDE of choice. The steps required are:

1. Access the component selection dialog of your IDE. In Visual Studio Code this can be accessed as explained here: Manage CMSIS Components
2. The most important component so select will be:
   - Machine Learning:TensorFlow:Kernel&Ethos-U
3. By selecting the variant “Ethos-U” a couple of dependencies can be resolved by the environment:
   - Arm::Machine Learning:NPU Support:Ethos-U Driver&Generic U55 *
   - ARM::CMSIS:DSP&Source
   - ARM::CMSIS:NN Lib
   - tensorflow::Data Exchange:Serialization:flatbuffers
   - tensorflow::Data Processing:Math:gemmlowp fixed-point
   - tensorflow::Data Processing:Math:kissfft
   - tensorflow::Data Processing:Math:ruy

- tensorflow::Machine Learning:TensorFlow:Kernel Utils

* Please notice that the Ethos-U Driver defaults to a variant called “Generic U55”. Depending on your actual device, a different driver variant can be available from the device vendor, and needs to be selected here.

Most relevant information to build optimized code is detected by the components itself, after you selected the available variants of the components.

7.3 Adding the model to your project

Generally there is two options to store your model on the embedded target. Either on an existing filesystem or compile it into the firmware image and flash it together with the application.

7.3.1 Loading from a filesystem

The TensorFlow library comes with the capability to interpret tflite files as they are stored on filesystem. This can be useful if you need to need to handle multiple models, and want to update them indepentently from the application.

The Keil Middleware File System provides a set of APIs to interact with a file system on a storage device like an SD card or USB flash drive. Here’s a simple code snippet showing how you might load a tflite file from an SD card:

```
#include "rl_fs.h"
...

FILE *f;
char *buffer;
size_t buffer_size;

// Open the file for reading
f = fopen("/sdcard/my_model_vela.tflite", "r");
if (f == NULL) {
    // Error handling
}

// Get the size of the file and allocate a buffer
fseek(f, 0, SEEK_END);
buffer_size = ftell(f);
rewind(f)
buffer = malloc(buffer_size + 1);  // +1 for the null terminator
if (buffer == NULL) {
    // Error handling
    fclose(f);
    return;
}

// Read the file into the buffer
size_t num = fread(buffer, 1, buffer_size, f);
if (num != buffer_size) {
    printf("Failed to read file\n");
} else {
    buffer[buffer_size] = '\0';  // Null-terminate the string
    printf("File contents: %s\n", buffer);
```

```
fclose(f);

const tflite::Model* model = ::tflite::GetModel(buffer);
free(buffer);

...
```

Loading model files will require much RAM and is not suitable for all system designs. A very typical scenario for this will be a test system based on Arm Virtual Hardware, where you load many different model variants for profiling runs. Even if a file system is available on your hardware target, you may want to store the model in ROM alongside the application, as explained in the next paragraph.

7.3.2 Storing in firmware image

To compile the content of a tflite file into your firmware image, it needs to be represented as an array in C language syntax. A hexdump utility like xxd can help to convert the binary tflite file into a headerfile to include in your project.

```
xxd -i my_model_vela.tflite my_network_model.h
```

To make sure that the data is stored in ROM memory and starts at a 16 Byte alignment boundary, the definition should be changed to:

```
const unsigned char network_model __ALIGNED(16) {

 ...

}
```

In the application code, include my_network_model.h and load the model with TensorFlow:

```
#include "my_network_model.h"

...

const tflite::Model* model = ::tflite::GetModel(network_model);

...
```

7.4 Using the TensorFlow Lite Micro API in the application

Following the above steps will have added all the necessary assets to your project to make a functional instance of TensorFlow Lite Micro available. The following section will provide and discuss a C++ code template for a typical implementation. Input tensors, preprocessing and output interpretation will be dependend on your model and might differ.

```
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/micro/tflite_bridge/micro_error_reporter.h"
```

```
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"

// Include your model data. Replace this with the header file for your model.
#include "model.h"

// Define the number of elements in the input tensor
#define INPUT_SIZE 300*300

using MyOpResolver = tflite::MicroMutableOpResolver<10>;

// Replace this with your model's input and output tensor sizes
const int tensor_arena_size = 2 * 1024;
uint8_t tensor_arena[tensor_arena_size] __align(16);

void RunModel(int8_t* input_data) {
    tflite::MicroErrorReporter micro_error_reporter;
    tflite::ErrorReporter* error_reporter = &micro_error_reporter;

    const tflite::Model* model = ::tflite::GetModel(g_model);

    MyOpResolver op_resolver;
    // Register your model's operations here

    tflite::MicroInterpreter interpreter(model, op_resolver, tensor_arena,
                                         tensor_arena_size, error_reporter);

    interpreter.AllocateTensors();

    TfLiteTensor* input = interpreter.input(0);
    // Add checks for input tensor properties here if needed

    // Copy image data to model's input tensor
    for (int i = 0; i < INPUT_SIZE; ++i) {
        input->data.int8[i] = input_data[i];
    }

    TfLiteStatus invoke_status = interpreter.Invoke();
    // Add checks for successful inference here if needed

    TfLiteTensor* output = interpreter.output(0);
    // Add checks for output tensor properties here if needed

    // Process your output value here
    // For example, SSD models typically produce an array of bounding boxes
}
```

The template will require C++ 17 as the minimum language standard, which complies to the standard TensorFlow is written in.

If you need to create your own model, a good starting point into the workflow is found in the TensorFlow Guide

Next step might be the Profiling and Optimization of ML Models. Arm provides many powerful tools to test and improve a models performance running on Ethos-U NPUs.

8. Profiling and Optimization of ML Models

The memory requirement and code size of a TensorFlow Lite (TFLite) Micro model can be determined by analyzing the model's parameters, layers, and the hardware platform on which it will run. Here are the key steps to estimate the memory requirement and code size:

- Model Analysis: Begin by examining the structure of the TFLite Micro model itself. This includes the number and type of layers, the input and output shapes, and the size of the model parameters. The model parameters consist of the weights and biases associated with each layer. A helpful tool can be visualization like prvided by Netron App
- TFLite Micro Profiler: TFLite Micro provides a profiler tool that allows you to measure memory usage and code size. You can find example code and instructions in the TFLite Micro documentation.
- Ethos Performance Counters to analyze code and memory usage.
- Memory Estimation Functions: TensorFlow Lite for Microcontrollers provides memory estimation functions that allow you to estimate the memory requirement programmatically. These functions help you calculate memory usage based on model parameters and tensor shapes.
- Compilers and Linkers: Arm Compiler, LLVM, GCC Linker Code/object size reports.
- Static Analysis Tools: Static analysis tools can provide information about the code size and memory usage of your TFLite Micro model without running it. (CI)

8.1 Ethos-U Vela Optimizations

Many optimizations can be made during the compilation step with Vela. Refer to this manual page for latest information.

8.2 Operator Mapping/Usage

Running vela with the `--verbose-performance` parameter will output a table with details of the TensorFlow operator usage in your model.

```
############################################################################
Performance for NPU Subgraph _split_1
TFLite_operator      NNG Operator         SRAM Usage  Peak%  Op Cycles Network%
   NPU    SRAM AC    DRAM AC OnFlash AC OffFlash AC  MAC Count Network%  Util% Name
-------------------- ------------------- ---------- ------ ---------- --------
 ---------- ---------- ---------- ---------- ---------- ---------- -------- ------
 --------------------
CONV_2D              Conv2DBias               629616  86.18    1889913
 46.80    1889913      21504          0          0           0   99090432
  49.99  20.48 ResNet18/activation_32/Relu;ResNet18/batch_normalization_32/
FusedBatchNormV3;ResNet18/conv2d_38/BiasAdd/ReadVariableOp/resource;ResNet18/
conv2d_38/BiasAdd;ResNet18/conv2d_39/Conv2D;ResNet18/conv2d_38/Conv2D1
CONV_2D              Conv2DBias               730624 100.00    2127584    52.69
  2127584      21504          0          0           0   99090432    49.99  18.19
 ResNet18/batch_normalization_33/FusedBatchNormV3;ResNet18/conv2d_39/BiasAdd/
ReadVariableOp/resource;ResNet18/conv2d_39/BiasAdd;ResNet18/conv2d_39/Conv2D
```

```
ADD                  Add                          43008   5.89       16128      0.40
    16128       8064          0          0           0           0     0.00    0.00
 ResNet18/activation_33/Relu;ResNet18/add_15/add
AVERAGE_POOL_2D      AvgPool                      27648   3.78        4224      0.10
     2200       4224          0          0           0       24576     0.01    2.27
 ResNet18/average_pooling2d_1/AvgPool
```

Cycles and Network% give a good indication which layer is compute and memory intensive.

MAC Count of 0 gives an indication of Operators that are not off-loaded to Ethos-U NPU. These will still be executed optimized by CMSIS-NN typically, but consume CPU time. If this is the case for a majority of your network, the model used is not suitable for Ethos-U.

8.3 MLIA guided optimizations (Experimental)

The ML Inference Advisor (MLIA) is a tool designed to assist AI developers in designing and optimizing neural network models for efficient inference on Arm® targets. It enables performance analysis and provides actionable advice early in the model development cycle. The advice can cover supported operators, performance analysis, and suggestions for model optimization, such as pruning and clustering.

MLIA works with sub-commands

- check to perform compatibility or performance checks on the model, and “optimize” to apply specified optimizations.
- optimize to apply specified optimizations. Optimization is only available for Keras model format (h5), not tflite.

8.4 Ethos-U Performance Profiling

The content for this section is currently under development and will be added in the near future.

9. MLOps Systems

Arm provides a set of foundation tools and software components to enable MLOps systems and the overall development flow for machine learning applications. Arm is also working with several MLOps partners to integrate these components into established MLOps systems. The Arm Partner Ecosystem Catalog provides a searchable list of AI partners.

9.1 Tool and Software Components

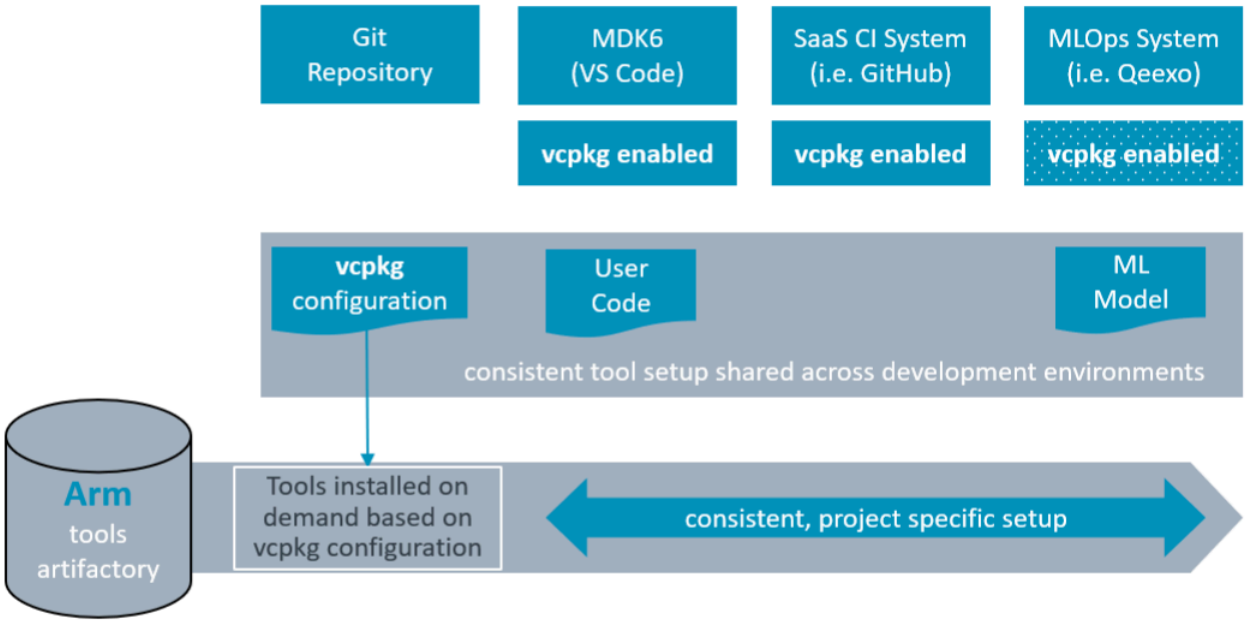
The section Overall Development Process describes the process steps for developing machine learning applications. Multiple development steps are supported by Arm with the following tool and software components.

Development Step	Tool and Software Components
Data Collection	Keil MDK, SDS Framework, MDK Middleware
Model Evaluation	Arm Compiler, Vela, Arm Virtual Hardware
System Validation	Arm Virtual Hardware, SDS Framework
Model Deployment	Keil MDK, Trusted Firmware

9.2 Tool Integration

To facilitate tool integration Arm provides critical foundation tools in a tools artifactory. These tools may be directly downloaded or the download process can be managed using the Microsoft tool vcpkg. Refer to "Install tools on the command line using vcpkg" for further details.

Figure 9-1: Arm Tools Artifactory



To activate the Arm development tools a license is required. Contact Arm for details on how to become an Arm MLOps partner.

10. Resources for Ethos-U

The following resources are available for Ethos-U:

- Product pages
- Product document
- Software and examples
- Other materials
- Partner's solution

10.1 Product pages

The following resources are available:

Product	Resource
Ethos-U55	https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55
	https://developer.arm.com/Processors/Ethos-U55
Ethos-U65	https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u65
	https://developer.arm.com/Processors/Ethos-U65

10.2 Product document

The following resources are available:

Product	Resource
Ethos-U55	Technical Reference Manual
Ethos-U55	Product Brief
Ethos-U65	Technical Reference Manual
Ethos-U65	Product Brief
Ethos-U55/U65	Arm Ethos-U NPU Application development overview

10.3 Software and examples

The following resources are available:

- Vela compiler
- TensorFlow Lite for Microcontrollers
- Arm Model Zoo

- ML Inference Advisor (MLIA)
- Ethos-U ML Embedded Evaluation kit
- Using the Arm Corstone-300 with Arm Cortex-M55 and Arm Ethos-U55 NPU - Jupyter notebook
- Ethos-U Core Platform
- CMSIS-Pack based Machine Learning Examples
- CMSIS-NN
- Additional ML examples for Arm processors
- learn.arm.com: Navigate Machine Learning development with Ethos-U processors
- learn.arm.com: Build and run the Arm Machine Learning Evaluation Kit examples
- Running TVM on bare metal Arm(R) Cortex(R)-M55 CPU, Ethos(TM)-U55 NPU and CMSIS-NN
- Benefit of pruning and clustering a neural network for before deploying on Arm Ethos-U NPU

10.4 Other materials

The following resources are available:

Product	Resource
Ethos-U55	Technical Overview of Ethos-U55 - video
Ethos-U55	Running Machine Learning on Arm's Ethos-U55 NPU - slide
Cortex-M55 + Ethos-U55	Arm M55 and U55 Performance Optimization for Edge-based Audio and Machine Learning Applications
TensorFlow Lite Micro	TensorFlow Lite Micro low level operations
ML model optimization	Optimize a ML model for fast inference on Ethos-U microNPU
Vela compiler	Vela Complier: The first step to deploy your NN model on the Arm Ethos-U microNPU
Arm ML Embedded Evaluation Kit	Blog: Arm ML Embedded Evaluation Kit
Using uTVM with Ethos-U	tinyML Summit 2023: Arm Ethos-U support in TVM ML framework - video

10.5 Partner's solution

The following resources are available:

Partner	Product/solution
Sensory	Sensory Speech Technologies on Arm IP
Arcturus	Energy-Efficient ML Vision App Development with Ethos-U65/i.MX 93