SOPHON-SAIL User Manual

Release 3.10.6

SOPHGO

Sep 18, 2025

Catalog

CHAPTER 1

Disclaimer

Legal Disclaimer

Notice

The purchased products, services and features are stipulated by the contract made between SOPHGO and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied. The information in this document is subject to change

without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Technical Support

Address Floor 6, Building 1, Yard 9, FengHao East Road, Haidian District, Beijing, 100094, China

Website https://www.sophgo.com/

Email sales@sophgo.com

Phone +86-10-57590723 +86-10-57590724

SAIL Release Records

Version	Release Date	Description
V2.0.0	SEP. 20, 2019	First release.
V2.0.1	NOV. 16, 2019	Version V2.0.1 was released.
V2.0.3	MAY. 07, 2020	Version V2.0.3 was released.
V2.2.0	OCT. 12, 2020	Version V2.2.0 was released.
V2.3.0	JAN. 11, 2021	Version V2.3.0 was released.
V2.3.1	MAR. 09, 2021	Version V2.3.1 was released.
V2.3.2	APR. 01, 2021	Version V2.3.2 was released.
V2.4.0	MAY. 23, 2021	Version V2.4.0 was released.
V2.5.0	SEP. 02, 2021	Version V2.5.0 was released.
V2.6.0	JAN. 30, 2022	Version V2.6.0 was released.
V2.7.0	MAR. 06, 2022	Version V2.7.0 was released, 20220531 was released as a patch version.
V3.0.0	JUL. 16, 2022	Version V3.0.0 was released.
V3.1.0	NOV. 01, 2022	Version V3.1.0 was released.
V3.2.0	DEC. 01, 2022	Version V3.2.0 was released.
V3.3.0	JAN. 01, 2023	Version V3.3.0 was released.

SAIL Update Content

- Add 'imdecode' interface to Bmcv for decoding images in memory.
- Add 'bmcv' interface to Bmcv for fast Fourier transform.
- Add 'MultiDecoder' interface for decoding multiple video channels at the same time.

- Add 'ImagePreProcess' interface for asynchronous image preprocessing.
- Add 'EngineImagePreProcess' interface, which combines image preprocessing and inference, and does parallel processing internally.
- Add video parallel reasoning routines to greatly improve the efficiency of full processing in a Python environment, reduce the complexity of code calls; simplify the calling process of C++ code.

CHAPTER 2

SAIL

SAIL(SOPHON Artificial Intelligent Library) is the core module of SOPHON-SAIL. SAIL wraps BMLib, sophon-mw(For BM1688 or CV186AH, named sophon-media), BMCV, and BMRuntime in SOPHONSDK, The original functions in SOPHONSDK such as "load bmodel and drive Tensor Computing Processor inference" , "drive Tensor Computing Processor for image processing" and "drive VPU for image and video decoding" are abstracted into simpler C++ interfaces and provided; and wrapped again with pybind11 to provide a clean and easy to use python interface.

Currently, all classes, enumerations and functions in the SAIL module are under the "sail" namespace. The documentation in this unit will give you an in-depth introduction to the modules and classes in SAIL that you may use. The core classes are as follows:

- Handle:

Wrapper class for bm_handle_t of BMLib in SDK (device handle, contextual information), used to interact with kernel driver information.

- Tensor:

Wrapper class for BMLib in SDK, encapsulating device memory management and synchronization with system memory.

- Engine:

Wrapper class for the BMRuntime in the SDK that loads the bmodel and drives the Tensor Computing Processor for reasoning. An Engine instance can load an arbitrary bmodel that automatically manages the memory corresponding to the input tensor and the output tensor.

- Decoder:

Use VPU to decode video and JPU to decode images, both in hardware.

- Bmcv:

Wrapper class for BMCV in the SDK, encapsulating a series of image processing functions that can drive the Tensor Computing Processor for image processing.

CHAPTER 3

Compilation and Installation Guide

3.1 The Directory Structure of the Source Code

The directory structure of the source code is as follows:

```
└─── SOPHON-SAIL
    ├─── 3rdparty
    │    ├─── json
    │    ├─── prebuild
    │    ├─── pybind11
    │    ├─── pybind11_new
    │    ├─── spdlog
    │    └─── ...
    ├─── cmake                  # Cmake Files
    │    ├─── BM168x_ARM_PCIE
    │    ├─── BM168x_LoongArch64
    │    ├─── BM168x_RISCV
    │    ├─── BM168x_SOC
    │    └─── ...
    ├─── docs                   # Documentation codes
    │    ├─── common
    │    ├─── source_common
    │    ├─── source_en
    │    └─── source_zh
    ├─── include                # Includes
    ├─── pyis                   # Stub files
    ├─── pysail-dev             # Files for secondary dev
    │    └─── ...
    ├─── python                 # Wheel codes
    ├─── python_wheels          # Python Wheels
```

(continues on next page)

(continued from previous page)

```
    |    ├──── arm_pcie
    |    ├──── loongarch
    |    ├──── soc_BM1684_BM1684X
    |    └──── soc_BM1688
    |    └──── ...
    ├──── sample                # Sample files
    |    └──── ...
    ├──── src                   # Source codes
    |    └──── ...
    └──── ...
```

Among them, 3rdparty mainly contains some header files of the third party on which sail needs to be compiled; cmake contains some cmake files used for compilation; include contains some header files of sail; pyis contains some stub files for Python interfaces; pysail-dev contains headers and libs for secondary developing other Python module; python folder contains the packaging code and scripts of python whl for each platform; python_wheels contains come pre-compiled sail wheels; sample contains some code samples for developers; src folder contains the code of each interface.

3.2 SAIL Compilation and Installation

Note: BM1688 processor only supports soc related compilation options, BM1684 and BM1684X processors do not have this restriction.

3.2.1 Compilation Parameters

- BUILD_TYPE : It is used to specify the type of compilation, currently there are three modes: pcie, soc and arm_pcie. pcie means to compile the SAIL package available on the x86 host, soc means to use cross-compilation to compile the SAIL package available on the soc of the x86 host. arm_pcie means to use cross-compilation to compile the SAIL package available on the x86 host with a bm168x card plugged in. The default is pcie.
- ONLY_RUNTIME : It is used to specify whether the compilation result contains only runtime but not bmcv, sophon-ffmpeg, or sophon-opencv; if this compilation option is ON, this SAIL encoding and decoding and the Bmcv interface are not available, and only the inference interface is available. The default is OFF.
- INSTALL_PREFIX : It is used to specify the installation path when executing make install, default "/opt/sophon" in pcie mode, same as the installation path of libsophon, default "build_soc" in cross-compile mode.
- PYTHON_EXECUTABLE : It is used to specify the path name (path+name) of "python3" for compilation; by default, the default python3 of the current system is used.

- CUSTOM_PY_LIBDIR : It is used to specify the path of the python3 dynamic library to be used for compilation (path only); by default, the current system's default python3 dynamic library directory is used.
- LIBSOPHON_BASIC_PATH : It is used to specify the path of libsophon in cross-compile mode; if the configuration is not correct, the compilation will fail. In pcie mode, this compilation option does not take effect.
- FFMPEG_BASIC_PATH : It is used to specify the path of sophon-ffmpeg in cross-compile mode; if the configuration is not correct and ONLY_RUNTIME is "OFF", the compilation will fail. In pcie mode, this compilation option does not take effect.
- OPENCV_BASIC_PATH : It is used to specify the path of sophon-opencv in cross-compile mode; if the configuration is not correct and ONLY_RUNTIME is "OFF", the compilation will fail. In pcie mode, this compilation option does not take effect.
- TOOLCHAIN_BASIC_PATH : It is used to specify the path of cross-compiler in cross-compile mode. Currently it only takes effect when BUILD_TYPE is loongarch.
- BUILD_PYSAIL : It is used to specify whether the compilation result contains the python version of SAIL. The default is "ON", which includes the python version of SAIL.

3.2.2 Compile dynamic libraries and header files that can be called by the C++ interface

Note: BM1688 and CV186AH processors only support SOC MODE chapter, BM1684 and BM1684X processors do not have this limitation.Note that for BM1688 or CV186AH, sophon-mw needs to be changed to sophon-media.

PCIE MODE

Install libsophon, sophon-ffmpeg, sophon-opencv for SAIL

The installation of libsophon, sophon-ffmpeg, and sophon-opencv can be found in the sophon's official documentation

.Compiling SAIL with Multimedia Modules

Compile SAIL with bmcv, sophon-ffmpeg, sophon-opencv using the default installation path

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DBUILD_PYSAIL=OFF ..
make sail
```

4. Install SAIL dynamic library and header files; the compiled result will be installed under the "/opt/sophon" directory

```
sudo make install
```

.Compiling SAIL without Multimedia Modules

Compile SAIL without bmcv, sophon-ffmpeg, sophon-opencv using the default installation path

The SAIL compiled in this way cannot use its Decoder, Bmcv, and other multimedia-related interfaces.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
mkdir build && cd build
```

4. Install SAIL dynamic library and header files; the compiled result will be installed under the "/opt/sophon" directory

```
sudo make install
```

SOC MODE

.Get the libsophon, sophon-ffmpeg, and sophon-opencv needed for cross-compilation

All compilation operations in this section are performed on the x86 host using cross-compilation. The following examples choose to use libsophon version 0.4.1, sophon-ffmpeg version 0.4.1, and sophon-opencv version 0.4.1.

1. Get "libsophon_soc_0.4.1_aarch64.tar.gz" from sophon' s official website and unpack it

```
tar -xvf libsophon_soc_0.4.1_aarch64.tar.gz
```

The directory of libsophon after unpacking is "libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1"

2. Get "sophon-mw-soc_0.4.1_aarch64.tar.gz" from sophon' s official website and unpack it

```
tar -xvf sophon-mw-soc_0.4.1_aarch64.tar.gz
```

The directory of sophon-ffmpeg after unpacking is "sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1" .

The directory of sophon-opencv after unpacking is "sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1" .

.Install the gcc-aarch64-linux-gnu toolchain

If already installed, you can ignore this step

```
sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

.Compiling SAIL with Multimedia Modules

Compile SAIL containing bmcv, sophon-ffmpeg, sophon-opencv through cross-compilation.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

   ```
   mkdir build && cd build
   ```

3. Execute the compilation command

   ```
   cmake -DBUILD_TYPE=soc -DBUILD_PYSAIL=OFF \
     -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_SOC/ToolChain_
   ↪aarch64_linux.cmake \
     -DLIBSOPHON_BASIC_PATH=libsophon_soc_0.4.1_aarch64/opt/sophon/
   ↪libsophon-0.4.1 \
     -DFFMPEG_BASIC_PATH=sophon-mw-soc_0.4.1_aarch64/opt/sophon/
   ↪sophon-ffmpeg_0.4.1 \
     -DOPENCV_BASIC_PATH=sophon-mw-soc_0.4.1_aarch64/opt/sophon/
   ↪sophon-opencv_0.4.1 ..
   make sail
   ```

4. Install SAIL dynamic library and header files; The program will automatically create "build_soc" in the source code directory and the compiled result will be installed under "build_soc"

   ```
   make install
   ```

5. Copy "sophon-sail" from the "build_soc" folder to the "/opt/sophon" directory on the target SOC, then can use SAIL on the target SOC host

.Compiling SAIL without Multimedia Modules

Compile SAIL that dose not include bmcv, sophon-ffmpeg, sophon-opencv through cross-compilation

The SAIL compiled in this way cannot use its Decoder, Bmcv, and other multimedia-related interfaces.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DBUILD_TYPE=soc \
  -DBUILD_PYSAIL=OFF \
  -DONLY_RUNTIME=ON \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_SOC/ToolChain_
↪aarch64_linux.cmake \
  -DLIBSOPHON_BASIC_PATH=libsophon_soc_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 ..
make sail
```

4. Install SAIL dynamic library and header files; The program will automatically create "build_soc" in the source code directory and the compiled result will be installed under "build_soc"

```
make install
```

5. Copy "sophon-sail" from the "build_soc" folder to the "/opt/sophon" directory on the target SOC, then can use SAIL on the target SOC host

ARM PCIE MODE

.Get the libsophon, sophon-ffmpeg, and sophon-opencv needed for cross-compilation

All compilation operations in this section are performed on the x86 host using cross-compilation. The following examples choose to use libsophon version 0.4.1, sophon-ffmpeg version 0.4.1, and sophon-opencv version 0.4.1.

1. Get "sophon-mw_0.4.1_aarch64.tar.gz" from sophon's official website and unpack it

```
tar -xvf libsophon_0.4.1_aarch64.tar.gz
```

The directory of libsophon after unpacking is "libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1"

2. Get "sophon-mw_0.4.1_aarch64.tar.gz" from sophon's official website and unpack it

```
tar -xvf sophon-mw_0.4.1_aarch64.tar.gz
```

The directory of sophon-ffmpeg after unpacking is "sophon-mw_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1" .

The directory of sophon-opencv after unpacking is "sophon-mw_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1" .

.Install the gcc-aarch64-linux-gnu toolchain

If already installed, you can ignore this step

```
sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

.Compiling SAIL with Multimedia Modules

Compile SAIL containing bmcv, sophon-ffmpeg, sophon-opencv through cross-compilation.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

   ```
   mkdir build && cd build
   ```

3. Execute the compilation command

   ```
   cmake -DBUILD_TYPE=arm_pcie \
     -DBUILD_PYSAIL=OFF \
     -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_ARM_PCIE/
   ↪ToolChain_aarch64_linux.cmake \
     -DLIBSOPHON_BASIC_PATH=libsophon_0.4.1_aarch64/opt/sophon/
   ↪libsophon-0.4.1 \
     -DFFMPEG_BASIC_PATH=sophon-mw_0.4.1_aarch64/opt/sophon/sophon-
   ↪ffmpeg_0.4.1 \
     -DOPENCV_BASIC_PATH=sophon-mw_0.4.1_aarch64/opt/sophon/sophon-
   ↪opencv_0.4.1 ..
   make sail
   ```

4. Install SAIL dynamic library and header files; The program will automatically create "build_arm_pcie" in the source code directory and the compiled result will be installed under "build_arm_pcie"

   ```
   make install
   ```

5. Copy "sophon-sail" from the "build_arm_pcie" folder to the "/opt/sophon" directory on the target ARM host, then can use SAIL on the target ARM host

.Compiling SAIL without Multimedia Modules

Compile SAIL that dose not include bmcv, sophon-ffmpeg, sophon-opencv through cross-compilation

The SAIL compiled in this way cannot use its Decoder, Bmcv, and other multimedia-related interfaces.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

   ```
   mkdir build && cd build
   ```

3. Execute the compilation command

```
cmake -DBUILD_TYPE=soc \
   -DBUILD_PYSAIL=OFF \
   -DONLY_RUNTIME=ON \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_SOC/ToolChain_
↪aarch64_linux.cmake \
   -DLIBSOPHON_BASIC_PATH=libsophon_soc_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 ..
make sail
```

4. Install SAIL dynamic library and header files; The program will automatically create "build_arm_pcie" in the source code directory and the compiled result will be installed under "build_arm_pcie"

```
make install
```

5. Copy "sophon-sail" from the "build_arm_pcie" folder to the "/opt/sophon" directory on the target ARM host, then can use SAIL on the target ARM host

LOONGARCH64 MODE

.Install the loongarch64-linux-gnu toolchain

Get the [cross-compiled toolchain](http://ftp.loongnix.cn/toolchain/gcc/release/loongarch/gcc8/loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.1.tar.xz) from the LoongArch64 official website, and unzip it locally. The directory structure after decompression is as follows:

```
└─── loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.1
   ├─── bin
   ├─── lib
   ├─── lib64
   ├─── libexec
   ├─── loongarch64-linux-gnu
   ├─── share
   ├─── sysroot
   └─── versions
```

.Get the libsophon, sophon-ffmpeg, and sophon-opencv needed for cross-compilation

All compilation operations in this section are performed on the x86 host using cross-compilation. The following examples choose to use libsophon version 0.4.7, sophon-ffmpeg version 0.6.0, and sophon-opencv version 0.6.0.

.Compiling SAIL with Multimedia Modules

Compile SAIL containing bmcv, sophon-ffmpeg, sophon-opencv through cross-compilation.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DBUILD_TYPE=loongarch -DBUILD_PYSAIL=OFF F
↪-DTOOLCHAIN_BASIC_PATH=toolchains/loongson-gnu-toolchain-8.
↪3-x86_64-loongarch64-linux-gnu-rc1.1 -DCMAKE_TOOLCHAIN_FILE=.
↪./cmake/BM168x_LoongArch64/ToolChain_loongarch64_linux.cmakeF
↪ -DLIBSOPHON_BASIC_PATH=libsophon_0.4.7_loongarch64/opt/
↪sophon/libsophon-0.4.7 -DFFMPEG_BASIC_PATH=sophon-mw_0.6.
↪0_loongarch64/opt/sophon/sophon-ffmpeg_0.6.0 -DOPENCV_BASIC_
↪PATH=sophon-mw_0.6.0_loongarch64/opt/sophon/sophon-opencv_0.6.0 F
↪ ..
make sail
```

4. Install SAIL dynamic library and header files; The program will automatically create "build_loongarch" in the source code directory and the compiled result will be installed under "build_loongarch"

```
make install
```

5. Copy "sophon-sail" from the "build_loongarch" folder to the "/opt/sophon" directory on the target LOONGARCH host, then can use SAIL on the target LOONGARCH host

.Compiling SAIL without Multimedia Modules

Compile SAIL that dose not include bmcv, sophon-ffmpeg, sophon-opencv through cross-compilation

The SAIL compiled in this way cannot use its Decoder, Bmcv, and other multimedia-related interfaces.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DBUILD_TYPE=loongarch \
  -DBUILD_PYSAIL=OFF \
  -DONLY_RUNTIME=ON \
  -DTOOLCHAIN_BASIC_PATH=toolchains/loongson-gnu-toolchain-8.3-x86_
↪64-loongarch64-linux-gnu-rc1.1 \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_LoongArch64/
↪ToolChain_loongarch64_linux.cmake \
  -DLIBSOPHON_BASIC_PATH=libsophon_0.4.7_loongarch64/opt/sophon/
↪libsophon-0.4.7 \
  ..
make sail
```

4. Install SAIL dynamic library and header files; The program will automatically create "build_loongarch" in the source code directory and the compiled result will be installed under "build_loongarch"

```
make install
```

5. Copy "sophon-sail" from the "build_loongarch" folder to the "/opt/sophon" directory on the target LOONGARCH host, then can use SAIL on the target LOONGARCH host

3.2.3 Compile dynamic libraries and header files that can be called by the Python interface

Note: BM1688 and CV186AH processors only support SOC MODE chapter, BM1684 and BM1684X processors do not have this limitation.Note that for BM1688 or CV186AH, sophon-mw needs to be changed to sophon-media.

PCIE MODE

Install libsophon, sophon-ffmpeg, sophon-opencv for SAIL

The installation of libsophon, sophon-ffmpeg, and sophon-opencv can be found in the sophon's official documentation

.Compiling SAIL with Multimedia Modules

Compile SAIL with bmcv, sophon-ffmpeg, sophon-opencv using the default installation path

If you don't need to use the python interface, you can ignore sections 5 and 6

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake ..
make pysail
```

4. Pack python wheel, the path of the generated wheel package is "python/dist" and the file name is "sophon-3.10.6 -py3-none-any.whl"

```
cd ../python
chmod +x sophon_whl.sh
./sophon_whl.sh
```

5. Install python wheel

```
pip3 install ./dist/sophon-3.10.6
-py3-none-any.whl --force-reinstall
```

.Compiling SAIL without Multimedia Modules

Compile SAIL without bmcv, sophon-ffmpeg, sophon-opencv using the default installation path

The SAIL compiled in this way cannot use its Decoder, Bmcv, and other multimedia-related interfaces.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DONLY_RUNTIME=ON ..
make pysail
```

4. Pack python wheel, the path of the generated wheel package is "python/dist" and the file name is "sophon-3.10.6 -py3-none-any.whl"

```
cd ../python
chmod +x sophon_whl.sh
./sophon_whl.sh
```

5. Install python wheel

```
pip3 install ./dist/sophon-3.10.6
-py3-none-any.whl --force-reinstall
```

.Compiling SAIL with a Specific Python Version

If the python3 version in the production environment is not the same as the development environment, you can make it consistent by upgrading the python3 version. You can also get the corresponding python3 package through the official python3 website. Or you can download the already compiled python3 from [Get Python3 for cross-compilation on an X86 host]. That is, use the non-system default python3, compile SAIL containing bmcv, sophon-ffmpeg, and sophon-opencv, and package it in the "build_pcie" directory. The path of python3 used in this example is "python_3.8.2/bin/python3" , and the dynamic library directory of python3 is "python_3.8.2/lib" .

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 -DCUSTOM_
↪PY_LIBDIR=python_3.8.2/lib ..
make pysail
```

4. Pack python wheel, the path of the generated wheel package is "python/dist" and the file name is "sophon-3.10.6 -py3-none-any.whl"

```
cd ../python
chmod +x sophon_whl.sh
./sophon_whl.sh
```

7. Install python wheel

Copy "sophon-3.10.6 -py3-none-any.whl" to the target machine, then execute the following installation command

```
pip3 install ./dist/sophon-3.10.6
-py3-none-any.whl --force-reinstall
```

3.2.4 SOC MODE

.Get the libsophon, sophon-ffmpeg, and sophon-opencv needed for cross-compilation

All compilation operations in this section are performed on the x86 host using cross-compilation. The following examples choose to use libsophon version 0.4.1, sophon-ffmpeg version 0.4.1, and sophon-opencv version 0.4.1.

1. Get "libsophon_soc_0.4.1_aarch64.tar.gz" from sophon' s official website and unpack it

```
tar -xvf libsophon_soc_0.4.1_aarch64.tar.gz
```

The directory of libsophon after unpacking is "libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1"

2. Get "sophon-mw-soc_0.4.1_aarch64.tar.gz" from sophon' s official website and unpack it

```
tar -xvf sophon-mw-soc_0.4.1_aarch64.tar.gz
```

The directory of sophon-ffmpeg after unpacking is "sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1" .

The directory of sophon-opencv after unpacking is "sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1" .

.Install the gcc-aarch64-linux-gnu toolchain

If already installed, you can ignore this step

```
sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

.Compiling SAIL with Multimedia Modules

Compile SAIL with bmcv, sophon-ffmpeg, and sophon-opencv by cross-compiling using the specified version of python3 (consistent with the version of python3 on the target SOC). You

can also get the corresponding python3 package through the official python3 website. Or you can download the already compiled python3 from [Get Python3 for cross-compilation on an X86 host]. The path of python3 used in this example is "python_3.8.2/bin/python3", and the dynamic library directory of python3 is "python_3.8.2/lib".

If you don't need to use the python interface, you can ignore sections 6 and 7

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DBUILD_TYPE=soc \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_SOC/ToolChain_
↪aarch64_linux.cmake \
  -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 \
  -DCUSTOM_PY_LIBDIR=python_3.8.2/lib \
  -DLIBSOPHON_BASIC_PATH=libsophon_soc_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 \
  -DFFMPEG_BASIC_PATH=sophon-mw-soc_0.4.1_aarch64/opt/sophon/
↪sophon-ffmpeg_0.4.1 \
  -DOPENCV_BASIC_PATH=sophon-mw-soc_0.4.1_aarch64/opt/sophon/
↪sophon-opencv_0.4.1 ..
make pysail
```

4. Pack python wheel, the path of the generated wheel package is "python/dist" and the file name is "sophon_arm-3.10.6 -py3-none-any.whl"

```
cd ../python
chmod +x sophon_whl.sh
./sophon_whl.sh
```

5. Install python wheel

Copy "sophon_arm-3.10.6 -py3-none-any.whl" to the target SOC, then execute the following installation command

```
pip3 install sophon_arm-3.10.6
-py3-none-any.whl --force-reinstall
```

.Compiling SAIL without Multimedia Modules

Compile SAIL without bmcv, sophon-ffmpeg, and sophon-opencv by cross-compiling using the specified version of python3 (consistent with python3 on the target SOC). You can also get the corresponding python3 package through the official python3 website. Or you can download the already compiled python3 from [Get Python3 for cross-compilation on an X86 host]. The path of python3 used in this example is "python_3.8.2/bin/python3", and the dynamic library directory of python3 is "python_3.8.2/lib".

The SAIL compiled in this way cannot use its Decoder, Bmcv, and other multimedia-related interfaces.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DBUILD_TYPE=soc \
   -DONLY_RUNTIME=ON \
   -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_SOC/ToolChain_
↪aarch64_linux.cmake \
   -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 \
   -DCUSTOM_PY_LIBDIR=python_3.8.2/lib \
   -DLIBSOPHON_BASIC_PATH=libsophon_soc_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 ..
make pysail
```

4. Pack python wheel, the path of the generated wheel package is "python/dist" and the file name is "sophon_arm-3.10.6 -py3-none-any.whl"

```
cd ../python
chmod +x sophon_whl.sh
./sophon_whl.sh
```

5. Install python wheel

Copy "sophon_arm-3.10.6 -py3-none-any.whl" to the target SOC, then execute the following installation command

```
pip3 install sophon_arm-3.10.6
-py3-none-any.whl --force-reinstall
```

3.2.5 ARM PCIE MODE

.Get the libsophon, sophon-ffmpeg, and sophon-opencv needed for cross-compilation

All compilation operations in this section are performed on the x86 host using cross-compilation. The following examples choose to use libsophon version 0.4.1, sophon-ffmpeg version 0.4.1, and sophon-opencv version 0.4.1.

1. Get "libsophon_0.4.1_aarch64.tar.gz" from sophon' s official website and unpack it

```
tar -xvf libsophon_0.4.1_aarch64.tar.gz
```

The directory of libsophon after unpacking is "libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1"

2. Get "sophon-mw_0.4.1_aarch64.tar.gz" from sophon' s official website and unpack it

```
tar -xvf sophon-mw_0.4.1_aarch64.tar.gz
```

The directory of sophon-ffmpeg after unpacking is "sophon-mw_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1" .

The directory of sophon-opencv after unpacking is "sophon-mw_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1" .

.Install the gcc-aarch64-linux-gnu toolchain

If already installed, you can ignore this step

```
sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

.Compiling SAIL with Multimedia Modules

Compile SAIL with bmcv, sophon-ffmpeg, and sophon-opencv by cross-compiling using the specified version of python3 (consistent with the version of python3 on the target ARM host). You can also get the corresponding python3 package through the official python3 website. Or you can download the already compiled python3 from [Get Python3 for cross-compilation on an X86 host]. The path of python3 used in this example is "python_3.8.2/bin/python3" , and the dynamic library directory of python3 is "python_3.8.2/lib" .

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

   ```
   mkdir build && cd build
   ```

3. Execute the compilation command

   ```
   cmake -DBUILD_TYPE=arm_pcie \
     -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_ARM_PCIE/
   ↪ToolChain_aarch64_linux.cmake \
     -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 \
     -DCUSTOM_PY_LIBDIR=python_3.8.2/lib \
     -DLIBSOPHON_BASIC_PATH=libsophon_0.4.1_aarch64/opt/sophon/
   ↪libsophon-0.4.1 \
     -DFFMPEG_BASIC_PATH=sophon-mw_0.4.1_aarch64/opt/sophon/sophon-
   ↪ffmpeg_0.4.1 \
     -DOPENCV_BASIC_PATH=sophon-mw_0.4.1_aarch64/opt/sophon/sophon-
   ↪opencv_0.4.1 ..
   make pysail
   ```

4. Pack python wheel, the path of the generated wheel package is "python/dist" and the file name is "sophon_arm_pcie-3.10.6 -py3-none-any.whl"

   ```
   cd ../python
   chmod +x sophon_whl.sh
   ./sophon_whl.sh
   ```

5. Install python wheel

Copy "sophon_arm_pcie-3.10.6 -py3-none-any.whl" to the target ARM host, then execute the following installation command

```
pip3 install sophon_arm_pcie-3.10.6
-py3-none-any.whl --force-reinstall
```

.Compiling SAIL without Multimedia Modules

Compile SAIL without bmcv, sophon-ffmpeg, and sophon-opencv by cross-compiling using the specified version of python3 (consistent with python3 on the target ARM host). You can also get the corresponding python3 package through the official python3 website. Or you can download the already compiled python3 from [Get Python3 for cross-compilation on an X86 host].. The path of python3 used in this example is "python_3.8.2/bin/python3" , and the dynamic library directory of python3 is "python_3.8.2/lib" .

The SAIL compiled in this way cannot use its Decoder, Bmcv, and other multimedia-related interfaces.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DBUILD_TYPE=arm_pcie \
  -DONLY_RUNTIME=ON \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_ARM_PCIE/
↪ToolChain_aarch64_linux.cmake \
  -DPYTHON_EXECUTABLE=python_3.8.2/bin/python3 \
  -DCUSTOM_PY_LIBDIR=python_3.8.2/lib \
  -DLIBSOPHON_BASIC_PATH=libsophon_0.4.1_aarch64/opt/sophon/
↪libsophon-0.4.1 ..
make
```

4. Pack python wheel, the path of the generated wheel package is "python/dist" and the file name is "sophon_arm_pcie-3.10.6 -py3-none-any.whl"

```
cd ../python
chmod +x sophon_whl.sh
./sophon_whl.sh
```

5. Install python wheel

Copy "sophon_arm_pcie-3.10.6 -py3-none-any.whl" to the target ARM host, then execute the following installation command

```
pip3 install sophon_arm_pcie-3.10.6
-py3-none-any.whl --force-reinstall
```

LOONGARCH64 MODE

.Install the loongarch64-linux-gnu toolchain

Get the [cross-compiled toolchain](http://ftp.loongnix.cn/toolchain/gcc/release/loongarch/gcc8/loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.1.tar.xz) from the LoongArch64 official website, and unzip it locally. The directory structure after decompression is as follows:

```
└─── loongson-gnu-toolchain-8.3-x86_64-loongarch64-linux-gnu-rc1.1
    ├─── bin
    ├─── lib
    ├─── lib64
    ├─── libexec
    ├─── loongarch64-linux-gnu
    ├─── share
    ├─── sysroot
    └─── versions
```

.Get the libsophon, sophon-ffmpeg, and sophon-opencv needed for cross-compilation

All compilation operations in this section are performed on the x86 host using cross-compilation. The following examples choose to use libsophon version 0.4.7.

.Compiling SAIL with Multimedia Modules

Compile SAIL without bmcv, sophon-ffmpeg, and sophon-opencv by cross-compiling using the specified version of python3 (consistent with python3 on the target ARM host). You can also get the corresponding python3 package through the official python3 website. Or you can download the already compiled python3 from [Get Python3 for cross-compilation on an X86 host].. The path of python3 used in this example is "python_3.8.2/bin/python3" , and the dynamic library directory of python3 is "python_3.8.2/lib" .

The SAIL compiled in this way cannot use its Decoder, Bmcv, and other multimedia-related interfaces.

1. Download the SOPHON-SAIL source code, unpack it and go to its source code directory
2. Create the build folder "build" and go to the "build" folder

```
mkdir build && cd build
```

3. Execute the compilation command

```
cmake -DBUILD_TYPE=loongarch \
  -DONLY_RUNTIME=ON \
  -DTOOLCHAIN_BASIC_PATH=toolchains/loongson-gnu-toolchain-8.3-x86_
↪64-loongarch64-linux-gnu-rc1.1 \
  -DCMAKE_TOOLCHAIN_FILE=../cmake/BM168x_LoongArch64/
↪ToolChain_loongarch64_linux.cmake \
  -DPYTHON_EXECUTABLE=python_3.7.3/bin/python3 \
  -DCUSTOM_PY_LIBDIR=python_3.7.3/lib \
```

(continues on next page)

(continued from previous page)

```
  -DLIBSOPHON_BASIC_PATH=libsophon_0.4.7_loongarch64/opt/sophon/
↪libsophon-0.4.7 \
  ..
make pysail
```

The path in the cmake option needs to be adjusted according to the configuration of your environment

- DLIBSOPHON_BASIC_PATH: The directory corresponding to the decompression of libsophon_<x.y.z>_loongarch64.tar.gz under libsophon in SOPHONSDK。

4. Pack python wheel, the path of the generated wheel package is "python/dist" and the file name is "sophon_loongarch64-3.10.6 -py3-none-any.whl"

```
cd ../python
chmod +x sophon_whl.sh
./sophon_whl.sh
```

5. Install python wheel

Copy "sophon_loongarch64-3.10.6 -py3-none-any.whl" to the target ARM host, then execute the following installation command

```
pip3 install sophon_loongarch64-3.10.6
-py3-none-any.whl --force-reinstall
```

3.2.6 Compile User Manual

.Install software packages

```
# Update apt
sudo apt update
# Install latex
sudo apt install texlive-xetex texlive-latex-recommended
# Install Sphinx
pip3 install sphinx sphinx-autobuild sphinx_rtd_theme rst2pdf
# Install the jieba Chinese text segmentation library to support Chinese search
pip3 install jieba3k
```

.Install fonts

[Fandol](https://ctan.org/pkg/fandol) - Four basic fonts for Chinese typesetting

```
# Download the font
wget http://mirrors.ctan.org/fonts/fandol.zip
# Unpack the font package
unzip fandol.zip
# Copy and install the font package
sudo cp -r fandol /usr/share/fonts/
cp -r fandol ~/.fonts
```

.Execute compilation

```
cd docs
make pdf LANG=en
```

The compiled user manual path is "docs/build/SOPHON-SAIL_en.pdf"

If the compilation still reports errors, you can run "sudo apt-get install texlive-lang-chinese", and then re-run the above command.

3.3 Develop Programs Using SAIL's Python Interface

Note: BM1688 and CV186AH processors only support SOC MODE chapter, BM1684 and BM1684X processors do not have this limitation.Note that for BM1688 or CV186AH, sophon-mw needs to be changed to sophon-media.

3.3.1 PCIE MODE

After compiling SAIL with PCIE MODE and installing python wheel, you can call SAIL in python, the interface documentation can be found in the API chapter.

3.3.2 SOC MODE

.Use your own compiled Python wheel package

After compiling SAIL by cross-compiling with SOC MODE, copy the python wheel to SOC and install it, then you can call SAIL in python, the interface document can be found in the API chapter.

.Use the pre-compiled Python wheel package

1. Check libsophon version and sophon-mw(sophon-ffmpeg,sophon-opencv) version on SOC

   ```
   ls /opt/sophon/
   ```

2. Check Python3 version on SOC

   ```
   python3 --version
   ```

3. You can find the corresponding version of the wheel package from the pre-compiled Python wheel package, copy it to the SOC, and install it; then, you can use python to call SAIL. Its interface documentation can be found in the API chapter.

3.3.3 ARM PCIE MODE

After compiling SAIL by cross-compiling with ARM PCIE MODE, copy the python wheel to ARM host and install it, then you can call SAIL in python, the interface document can be found in the API chapter.

1. Check libsophon version and sophon-mw(sophon-ffmpeg,sophon-opencv) version on the ARM host

```
ls /opt/sophon/
```

2. Check Python3 version on the ARM host

```
python3 --version
```

3. You can find the corresponding version of the wheel package from the pre-compiled Python wheel package, copy it to the ARM host, and install it; then, you can use python to call SAIL. Its interface documentation can be found in the API chapter.

3.4 Develop Programs Using SAIL's C++ Interface

Note: BM1688 and CV186AH processors only support SOC MODE chapter, BM1684 and BM1684X processors do not have this limitation.Note that for BM1688 or CV186AH, sophon-mw needs to be changed to sophon-media.

3.4.1 PCIE MODE

After compiling SAIL with PCIE MODE and installing SAIL's c++ libraries by running "sudo make install" or by copying them. It is recommended to use cmake to link the SAIL libraries to your application. If you need to use SAIL multimedia-related functions, you also need to add libsophon, sophon-ffmpeg, sophon-opencv header file directory, and dynamic library directory to your program. You can add the following paragraph to your program's CMakeLists.txt:

```
find_package(libsophon REQUIRED)
include_directories(${LIBSOPHON_INCLUDE_DIRS})
# Add libsophon's header file directories

set(SAIL_DIR  /opt/sophon/sophon-sail/lib/cmake)
find_package(SAIL REQUIRED)
include_directories(${SAIL_INCLUDE_DIRS})
link_directories(${SAIL_LIB_DIRS})
# Add SAIL header files and dynamic library directories

set(OpenCV_DIR  /opt/sophon/sophon-opencv-latest/lib/cmake/opencv4)
find_package(OpenCV REQUIRED)
include_directories(${OpenCV_INCLUDE_DIRS})
```

(continues on next page)

(continued from previous page)

```
# Add the header file directories of sophon-opencv

set(FFMPEG_DIR  /opt/sophon/sophon-ffmpeg-latest/lib/cmake)
find_package(FFMPEG REQUIRED)
include_directories(${FFMPEG_INCLUDE_DIRS})
link_directories(${FFMPEG_LIB_DIRS})
# Add the header file directories and dynamic library directories of sophon-ffmpeg

add_executable(${YOUR_TARGET_NAME} ${YOUR_SOURCE_FILES})
target_link_libraries(${YOUR_TARGET_NAME} sail)
```

The functions in sail can be called from within your code:

```
#define USE_FFMPEG  1
#define USE_OPENCV  1
#define USE_BMCV    1

#include <stdio.h>
#include <sail/cvwrapper.h>
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int device_id = 0;
    std::string video_path = "test.avi";
    sail::Decoder decoder(video_path,true,device_id);
    if(!decoder.is_opened()){
        printf("Video[%s] read failed!\n",video_path.c_str());
        exit(1) ;
    }

    sail::Handle handle(device_id);
    sail::Bmcv bmcv(handle);

    while(true){
        sail::BMImage ost_image = decoder.read(handle);
        bmcv.imwrite("test.jpg", ost_image);
        break;
    }

    return 0;
}
```

3.4.2 SOC MODE

.Compile the program on the SOC device

After installing libsophon, sophon-ffmpeg, sophon-opencv, and SAIL on the SOC device, you can use cmake to link the libraries in SAIL to your application by referring to the PCIE MODE development method. If you need to use SAIL multimedia-related functions, you also need to add libsophon, sophon-ffmpeg, and sophon-opencv header file directories and dynamic library directories to your application.

.Cross-compile programs on x86 hosts

If you want to build a cross-compilation environment using SAIL, you will need libsophon, sophon-ffmpeg, sophon-opencv, and the gcc-aarch64-linux-gnu toolchain.

.Create the "soc-sdk" folder

Create the "soc-sdk" folder, the header files and dynamic libraries needed for subsequent cross-compilation will be stored in this directory.

```
mkdir soc-sdk
```

.Get the libsophon,sophon-ffmpeg,sophon-opencv libraries needed for cross-compilation

The following examples choose to use libsophon version 0.4.1, sophon-ffmpeg version 0.4.1, and sophon-opencv version 0.4.1.

1. Get "libsophon_soc_0.4.1_aarch64.tar.gz" from sophon's official website and unpack and copy it to the "soc-sdk" folder

```
tar -xvf libsophon_soc_0.4.1_aarch64.tar.gz
cp -r libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1/include soc-sdk
cp -r libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1/lib soc-sdk
```

The directory of libsophon after unpacking is "libsophon_soc_0.4.1_aarch64/opt/sophon/libsophon-0.4.1"

2. Get "sophon-mw-soc_0.4.1_aarch64.tar.gz" from sophon's official website and unpack and copy it to the "soc-sdk" folder

```
tar -xvf sophon-mw-soc_0.4.1_aarch64.tar.gz
cp -r sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1/include
↪soc-sdk
cp -r sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1/lib soc-sdk
cp -r sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1/include/
↪opencv4/opencv2 soc-sdk/include
cp -r sophon-mw-soc_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1/lib soc-sdk
```

.Copy the cross-compiled SAIL, i.e. "build_soc", to the "soc-sdk" folder

```
cp build_soc/sophon-sail/include soc-sdk
cp build_soc/sophon-sail/lib soc-sdk
```

.Install the gcc-aarch64-linux-gnu toolchain

If already installed, you can ignore this step

```
sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

After the above steps are configured, you can finish cross-compiling by configuring cmake. Add the following paragraph to your program' s CMakeLists.txt:

CMakeLists.txt needs to use "/opt/sophon/soc-sdk" as the absolute path to "soc-sdk" , which needs to be configured according to the actual location of the file when it is applied.

```
set(CMAKE_C_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_ASM_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_CXX_COMPILER aarch64-linux-gnu-g++)

include_directories("/opt/sophon/soc-sdk/include")
include_directories("/opt/sophon/soc-sdk/include/sail")
# Add the header file directories to be used for cross-compilation

link_directories("/opt/sophon/soc-sdk/lib")
# Add dynamic library directories to be used for cross-compilation

add_executable(${YOUR_TARGET_NAME} ${YOUR_SOURCE_FILES})
target_link_libraries(${YOUR_TARGET_NAME} sail)
# sail is the library that needs to be linked
```

3.4.3 ARM PCIE MODE

.Compile the program on the ARM host

After installing libsophon, sophon-ffmpeg, sophon-opencv, and SAIL on the ARM host, you can use cmake to link the libraries in SAIL to your application by referring to the PCIE MODE development method. If you need to use SAIL multimedia-related functions, you also need to add libsophon, sophon-ffmpeg, and sophon-opencv header file directories and dynamic library directories to your application.

.Cross-compile programs on x86 hosts

If you want to build a cross-compilation environment using SAIL, you will need libsophon, sophon-ffmpeg, sophon-opencv, and the gcc-aarch64-linux-gnu toolchain.

.Create the "arm_pcie-sdk" folder

Create the "arm_pcie-sdk" folder, the header files and dynamic libraries needed for subsequent cross-compilation will be stored in this directory.

```
mkdir arm_pcie-sdk
```

.Get the libsophon,sophon-ffmpeg,sophon-opencv libraries needed for cross-compilation

The following examples choose to use libsophon version 0.4.1, sophon-ffmpeg version 0.4.1, and sophon-opencv version 0.4.1.

1. Get "libsophon_0.4.1_aarch64.tar.gz" from sophon's official website and unpack and copy it to the "arm_pcie-sdk" folder

```
tar -xvf libsophon_0.4.1_aarch64.tar.gz
cp -r libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1/include arm_pcie-sdk
cp -r libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1/lib arm_pcie-sdk
```

The directory of libsophon after unpacking is "libsophon_0.4.1_aarch64/opt/sophon/libsophon-0.4.1"

2. Get "sophon-mw_0.4.1_aarch64.tar.gz" from sophon's official website and unpack and copy it to the "arm_pcie-sdk" folder

```
tar -xvf sophon-mw_0.4.1_aarch64.tar.gz
cp -r sophon-mw_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1/include arm_
↪pcie-sdk
cp -r sophon-mw_0.4.1_aarch64/opt/sophon/sophon-ffmpeg_0.4.1/lib arm_pcie-
↪sdk
cp -r sophon-mw_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1/include/
↪opencv4/opencv2 arm_pcie-sdk/include
cp -r sophon-mw_0.4.1_aarch64/opt/sophon/sophon-opencv_0.4.1/lib arm_pcie-
↪sdk
```

.Copy the cross-compiled SAIL, i.e. "build_arm_pcie", to the "arm_pcie-sdk" folder

```
cp build_arm_pcie/sophon-sail/include arm_pcie-sdk
cp build_arm_pcie/sophon-sail/lib arm_pcie-sdk
```

.Install the gcc-aarch64-linux-gnu toolchain

If already installed, you can ignore this step

```
sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

After the above steps are configured, you can finish cross-compiling by configuring cmake. Add the following paragraph to your program's CMakeLists.txt:

CMakeLists.txt needs to use "/opt/sophon/arm_pcie-sdk as the absolute path to "arm_pcie-sdk", which needs to be configured according to the actual location of the file when it is applied.

```
set(CMAKE_C_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_ASM_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_CXX_COMPILER aarch64-linux-gnu-g++)

include_directories("/opt/sophon/arm_pcie-sdk/include")
include_directories("/opt/sophon/arm_pcie-sdk/include/sail")
# Add the header file directories to be used for cross-compilation
```

(continues on next page)

(continued from previous page)

```
link_directories("/opt/sophon/arm_pcie-sdk/lib")
# Add dynamic library directories to be used for cross-compilation

add_executable(${YOUR_TARGET_NAME} ${YOUR_SOURCE_FILES})
target_link_libraries(${YOUR_TARGET_NAME} sail)
# sail is the library that needs to be linked
```


CHAPTER 4

SAIL C++ API

4.1 Basic function

Mainly used to obtain or configure device information and properties.

4.1.1 get_available_tpu_num

Get the number of available Tensor Computing Processors in the current device.

Interface:

```
int get_available_tpu_num();
```

Returns:

Returns the number of Tensor Computing Processors available in the current device.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    int tpu_len = sail::get_available_tpu_num();
    std::cout << "available tpu: " << tpu_len << std::endl;
    return 0;
}
```

4.1.2 set_print_flag

Set whether to print the program' s calculation time information.

Interface:

```
int set_print_flag(bool print_flag);
```

Returns:

- print_flag: bool

When print_flag is True, the main time-consuming information of the calculation of program is printed, otherwise it is not printed.

Sample:

```
#include <sail/engine_multi.h>

int main() {
    int ret = sail::set_print_flag(true);
    if (ret == 0){
        std::cout << "set print time success" << std::endl;
    }
    return 0;
}
```

4.1.3 set_dump_io_flag

Set whether to store input data and output data.

Interface:

```
int set_dump_io_flag(bool dump_io_flag);
```

Parameter:

- dump_io_flag: bool

When dump_io_flag is True, input data and output data are stored, otherwise they are not stored.

Sample:

```
#include <sail/engine_multi.h>

int main() {
    ret = sail::set_dump_io_flag(true);
    if (ret == 0){
        std::cout << "set save data success" << std::endl;
    }
    return 0;
}
```

4.1.4 set_loglevel

Set the logging level to the specified level. Lower log levels are typically used in production environments to reduce performance overhead and the volume of log data, while higher log levels are suitable for development and debugging in order to capture more detailed information.

Interface:

```
int set_loglevel(LogLevel loglevel);
```

Parameters:

- loglevel: LogLevel

The Target log level as a sail.LogLevel enum value. The optional values include TRACE, DEBUG, INFO, WARN, ERR, CRITICAL, OFF, and the default level is INFO.

Returns:

return: int

Returning 0 indicates the log level was set successfully, whereas returning -1 indicates a failure due to an unknown log level.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    int ret = sail::set_loglevel(sail::LogLevel::TRACE);
    if (ret == 0){
        std::cout << "Set log level successfully" << std::endl;
    }
    else{
        std::cout << "Unknown log level, set failed." << std::endl;
    }
    return 0;
}
```

4.1.5 set_decoder_env

Set the parameters for the Decoder (including MutlDecoder) through environment variables. These must be set before the Decoder is constructed; otherwise, the default values will be used. This is mainly applicable to video decoding.

Interface:

```
int set_decoder_env(std::string env_name, std::string env_value);
```

Parameter:

- env_name: string

The property name to set for the Decoder. The available property names are as follows:

- 'rtsp_transport' : The transport protocol used for RTSP. The default is TCP.
- 'extra_frame_buffer_num' : The maximum number of cached frames for the Decoder. The default is 5.
- 'stimeout' : Raise error timeout, in milliseconds. The default is 20000000, i.e., 20 seconds.
- 'skip_non_idr' : Decoding frame skip mode. 0, no skip; 1, skip Non-RAP frames; 2, skip non-reference frames. The default is 0.
- 'fflags' : format flags, like "nobuffer" . Read ffmpeg official docs for more details.
- 'rtsp_flags' : Set RTSP flags. The default is prefer_tcp.
- 'refcounted_frames' : When set to 1, the decoded images need to be manually released by the program; when set to 0, they are automatically released by the Decoder.
- 'probesize' : the max size of the data to analyze to get stream information. 5000000 by default.
- 'analyzeduration' : How many microseconds are analyzed to probe the input. 5000000 by default.
- 'buffer_size' : The maximum socket buffer size in bytes.
- 'max_delay' : Maximum demuxing delay in microseconds.
- env_value: string

The configuration value of this property

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    sail::set_decoder_env("extra_frame_buffer_num", "3"); // Decrease buffer num for
↪lower memory usage
    sail::set_decoder_env("probesize", "1024") // Decrease probesize for lower latency
    sail::set_decoder_env("skip_non_idr", "2") // skip non-reference frames
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string video_path = "input_video.mp4";
    sail::Decoder decoder(video_path, true, dev_id);
    sail::BMImage bmimg = decoder.read(handle);
    return 0;
}
```

4.1.6 base64_enc

Base64 encode the data to generate the corresponding base64 encoded string. BM1688 and CV186AH PCI

```
int base64_enc(Handle& handle, const void *data, uint32_t dlen, std::string& encoded);
```

Parameter:

- handle: Handle

The handle of the device.

- data: const void*

The pointer to the data to be encoded.

- dlen: uint32_t

The byte length of the data to be encoded.

- encoded: string

encoded The string generated by base64 encoding.

Returns

Return 0 on successful base64 encoding, otherwise return -1.

Sample:

```
#include <sail/base64.h>

int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);

    std::string data = "hello,world!";

    // base64 encode
    std::string base64_encoded;
    uint32_t dlen = data.length();
    ret = sail::base64_enc(handle, data.c_str(), dlen, base64_encoded);
    if (ret == 0){
        std::cout << dlen << std::endl;
        std::cout << "base64 encode success!" << "based 64:" << base64_encoded << " lens
↪" << dlen << std::endl;
    }
    return 0;
}
```

4.1.7 base64_dec

Base64 encode the data to generate the corresponding base64 encoded string. BM1688 and CV186AH PCIE mode are not supported.

```
int base64_dec(Handle& handle, const void *data, uint32_t dlen, uint8_t* p_outbuf,
↪ uint32_t *p_size);
```

Parameter:

- handle: Handle

The handle of the device.

- data: const void*

The pointer to the data to be decoded.

- dlen: uint32_t

The byte length of the data to be decoded.

- p_outbuf: uint8_t*

Pointer to the decoded data.

- p_size: uint32_t *

Length of the pointer to the decoded data.

Returns

Return 0 on successful base64 encoding, otherwise return -1.

Sample:

```
#include <sail/base64.h>

int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);

    std::string data = "hello,world!";

    // base64 encode
    std::string base64_encoded;
    uint32_t dlen = data.length();
    ret = sail::base64_enc(handle, data.c_str(), dlen, base64_encoded);
    if (ret == 0){
        std::cout << dlen << std::endl;
        std::cout << "base64 encode success!" << "based 64:" << base64_encoded << "lens
↪" << dlen << std::endl;
    }
```

(continues on next page)

(continued from previous page)

```
    // base64_dec
    uint32_t dlen_based = base64_encoded.length();
    uint8_t out_data_buf[100]; // set enough space for decoded data
    uint32_t out_data_size; // decoded data length
    ret =sail::base64_dec(handle, base64_encoded.c_str(), dlen_based, out_data_buf, &out_
↪data_size);
    if (ret == 0){
        std::cout << "base64 decode success,data size is:" << out_data_size << std::endl;
        for(uint32_t i = 0; i < out_data_size; i++) {
            std::cout << out_data_buf[i];
        }
        std::cout << std::endl;
    }
    return 0;
}
```

4.1.8 get_tpu_util

Get the processor utilization of the specified device

Interface:

```
int get_tpu_util(int dev_id);
```

Parameter:

- dev_id: int

Device ID.

Return:

Returns the processor utilization of the device corresponding to the ID.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    int tpu_util;
    tpu_util = sail::get_tpu_util(0);
    std::cout << "tpu_util " << tpu_util << "%"<< std::endl;
    return 0;
}
```

4.1.9 get_vpu_util

Get the VPU percent utilization of the specified device

Interface:

```
std::vector<int> get_tpu_util(int dev_id);
```

Parameter:

- dev_id: int

Device ID.

Return:

The vpu of bm1684 is 5-core, and the return value is a list of length 5. The vpu of bm1684x is 3-core, and the return value is a list of length 3. Each integer in the List is the percent utilization of the corresponding core.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    std::vector<int> vpu_util;
    vpu_util = sail::get_vpu_util(0);

    for(int i = 0; i < vpu_util.size(); i++) {
        std::cout << "VPU ID: " << i << ", Util Value: " << vpu_util[i] << "%" <<F
↪std::endl;
    }
    return 0;
}
```

4.1.10 get_vpp_util

Get the VPP utilization of the specified device

Interface:

```
std::vector<int> get_vpp_util(int dev_id);
```

Parameter:

- dev_id: int

Device ID.

Return:

The vpp of bm1684 and bm1684x are both 2-core, and the return value is a list of length 2. Each integer in the List is the percent utilization of the corresponding core.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    std::vector<int> vpp_util;
    vpp_util = sail::get_vpu_util(0);

    for(int i = 0; i < vpp_util.size(); i++) {
        std::cout << "VPU ID: " << i << ", Util Value: " << vpp_util[i] << "%" <<F
↪std::endl;
    }
    return 0;
}
```

4.1.11 get_board_temp

Get the temperature of the board.

Interface:

```
int get_board_temp(int dev_id);
```

Parameter:

- dev_id: int

Device ID.

Return:

The board temperature for the corresponding card, with the default unit in Celsius (°C)

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    int board_temp;
    board_temp = sail::get_board_temp(0);
    std::cout << "board_temp " << board_temp << "°C"<< std::endl;
    return 0;
}
```

4.1.12 get_chip_temp

Get the temperature of the chip.

Interface:

```
int get_chip_temp(int dev_id);
```

Parameter:

- dev_id: int

Device ID.

Return:

The processor temperature for the corresponding card, with the default unit in Celsius (°C)

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    int chip_temp;
    chip_temp = sail::get_chip_temp(0);
    std::cout << "chip_temp " << bchip_temp << "°C"<< std::endl;
    return 0;
}
```

4.1.13 get_dev_stat

Get device memory information.

Interface:

```
int get_dev_stat(int dev_id);
```

Parameter:

- dev_id: int

Device ID.

Return:

A list of memory information for the corresponding device: [mem_total, mem_used, tpu_util].

Sample:

```
#include <sail/cvwrapper.h>

int main() {
```

(continues on next page)

(continued from previous page)

```
    std::vector<int> dev_stat;
    dev_stat = sail::get_dev_stat(0);

    std::cout << "mem_total: " << dev_stat[0] << " MB" << std::endl;
    std::cout << "mem_used: " << dev_stat[1] << " MB" << std::endl;
    std::cout << "tpu_util: " << dev_stat[2] << " %" << std::endl;
    return 0;
}
```

4.2 SAIL enum type

4.2.1 Data type

Define commonly used data types in the SOPHON environment

Interface:

```
enum bm_data_type_t {
    BM_FLOAT32,
    BM_INT8,
    BM_UINT8,
    BM_INT32,
    BM_UINT32
};
```

Parameters:

- BM_FLOAT32 Data type is float32
- BM_INT8 Data type is int8
- BM_UINT8 Data type is uint8
- BM_INT32 Data type is int32
- BM_UINT32 Data type is uint32

4.2.2 Format

Define commonly used image formats.

Interface:

```
FORMAT_YUV420P
FORMAT_YUV422P
FORMAT_YUV444P
FORMAT_NV12
FORMAT_NV21
```

(continues on next page)

(continued from previous page)

```
FORMAT_NV16
FORMAT_NV61
FORMAT_NV24
FORMAT_RGB_PLANAR
FORMAT_BGR_PLANAR
FORMAT_RGB_PACKED
FORMAT_BGR_PACKED
FORMAT_RGBP_SEPARATE
FORMAT_BGRP_SEPARATE
FORMAT_GRAY
FORMAT_COMPRESSED
```

Paramters:

- FORMAT_YUV420P

Pre-create an image in YUV420 format with three planes

- FORMAT_YUV422P

Pre-create an image in YUV422 format with three planes

- FORMAT_YUV444P

Pre-create an image in YUV444 format with three planes

- FORMAT_NV12

Pre-create an image in NV12 format with two planes

- FORMAT_NV21

Pre-create an image in NV21 format with two planes

- FORMAT_NV16

Pre-create an image in NV16 format with two planes

- FORMAT_NV61

Pre-create an image in NV61 format with two planes

- FORMAT_RGB_PLANAR

Pre-create an image in RGB format, with RGB arranged separately and a plane

- FORMAT_BGR_PLANAR

Pre-create an image in BGR format, with BGR arranged separately and a plane

- FORMAT_RGB_PACKED

Pre-create an image in RGB format, with RGB staggered and a plane

- FORMAT_BGR_PACKED

Pre-create an image in BGR format, with BGR staggered and a plane

- FORMAT_RGBP_SEPARATE

Pre-create an image in RGB planar format, in which RGB is arranged separately and each occupies a plane. There are 3 planes in total.

- FORMAT_BGRP_SEPARATE

Pre-create an image in BGR planar format, in which BGR is arranged separately and each occupies a plane. There are 3 planes in total.

- FORMAT_GRAY

Pre-create an image in grayscale format with a plane

- FORMAT_COMPRESSED

Pre-create an iamge in the VPU internal compression format. There are four planes, and the contents are stored as follows:

plane0: Y compression table

plane1: Y compressed data

plane2: CbCr compression table

plane3: CbCr compressed data

4.2.3 ImgDtype

Define several image storage formats. The type of values is BMCV enum `bm_image_data_format_ext` .

Interface:

```
DATA_TYPE_EXT_FLOAT32
DATA_TYPE_EXT_1N_BYTE
DATA_TYPE_EXT_4N_BYTE
DATA_TYPE_EXT_1N_BYTE_SIGNED
DATA_TYPE_EXT_4N_BYTE_SIGNED
```

Parameters:

- DATA_TYPE_EXT_FLOAT32

The data type of the image is float32.

- DATA_TYPE_EXT_1N_BYTE

The data type of the image is uint8。

- DATA_TYPE_EXT_4N_BYTE

The data type of the image is uint8, and the data of each 4 images is staggered, and the data reading and writing efficiency is higher.

- DATA_TYPE_EXT_1N_BYTE_SIGNED

The data type of the image is int8。

- DATA_TYPE_EXT_4N_BYTE_SIGNED

The data type of the image is int8, and the data of each 4 images is staggered, and the data reading and writing efficiency is higher.

4.2.4 IOMode

IOMode is used to define the memory location information (device memory or system memory) of the input Tensor and output Tensor.

Interface:

```
enum IOMode {
    SYSI,
    SYSO,
    SYSIO,
    DEVIO
};
```

Parameters:

- SYSI

The input Tensor is in system memory, and the output Tensor is in device memory.

- SYSO

The input Tensor is in device memory, and the output Tensor is in system memory.

- SYSIO

The input Tensor is in system memory, and the output Tensor is in system memory.

- DEVIO

The input Tensor is in device memory, and the output Tensor is in device memory.

4.2.5 bmcv_resize_algorithm

The interpolation strategies in image resize

Interface:

```
enum bmcv_resize_algorithm_ {
    BMCV_INTER_NEAREST = 0,
    BMCV_INTER_LINEAR  = 1,
    BMCV_INTER_BICUBIC = 2
} bmcv_resize_algorithm;
```

Parameters

- BMCV_INTER_NEAREST

Using nearest neighbor interpolation method while resizing.

- BMCV_INTER_LINEAR

Using linear interpolation method while resizing.

- BMCV_INTER_BICUBIC

Using double cubic interpolation method while resizing.

4.2.6 sail_resize_type

Preprocessing method corresponding to image preprocessing.

Interface:

```
enum sail_resize_type {
   BM_RESIZE_VPP_NEAREST = 0,
   BM_RESIZE_TPU_NEAREST = 1,
   BM_RESIZE_TPU_LINEAR = 2,
   BM_RESIZE_TPU_BICUBIC = 3,
   BM_PADDING_VPP_NEAREST = 4,
   BM_PADDING_TPU_NEAREST = 5,
   BM_PADDING_TPU_LINEAR = 6,
   BM_PADDING_TPU_BICUBIC = 7
};
```

Paramters:

- BM_RESIZE_VPP_NEAREST

Use VPP to perform image scale transformation using the nearest neighbor method.

- BM_RESIZE_TPU_NEAREST

Use Tensor Computing Processor to perform image scale transformation using the nearest neighbor method.

- BM_RESIZE_TPU_LINEAR

Use Tensor Computing Processor to perform image scale transformation using linear interpolation.

- BM_RESIZE_TPU_BICUBIC

Use Tensor Computing Processor to perform image scale transformation using bicubic interpolation.

- BM_PADDING_VPP_NEAREST

Use VPP to perform image scale transformation with padding using the nearest neighbor method.

- BM_PADDING_TPU_NEAREST

Use Tensor Computing Processor to perform image scale transformation with padding using the nearest neighbor method.

- BM_PADDING_TPU_LINEAR

Use Tensor Computing Processor to perform image scale transformation with padding using linear interpolation.

- BM_PADDING_TPU_BICUBIC

Use Tensor Computing Processor to perform image scale transformation with padding using bicubic interpolation method.

4.3 PaddingAtrr

PaddingAtrr stores various attributes of data padding, and data filling can be performed by configuring PaddingAtrr.

```
class PaddingAtrr {
public:
        PaddingAtrr(){};
        PaddingAtrr(
            unsigned int crop_start_x,
            unsigned int crop_start_y,
            unsigned int crop_width,
            unsigned int crop_height,
            unsigned char padding_value_r,
            unsigned char padding_value_g,
            unsigned char padding_value_b);
    PaddingAtrr(const PaddingAtrr& other);
    ~PaddingAtrr(){};
    void set_stx(unsigned int stx);
    void set_sty(unsigned int sty);
    void set_w(unsigned int w);
    void set_h(unsigned int h);
    void set_r(unsigned int r);
    void set_g(unsigned int g);
    void set_b(unsigned int b);

    unsigned int    dst_crop_stx; // Offset x information relative to the origin of dst image
    unsigned int    dst_crop_sty; // Offset y information relative to the origin of dst image
    unsigned int    dst_crop_w;   // The width after resize
    unsigned int    dst_crop_h;   // The height after resize
    unsigned char   padding_r;    // Pixel value information of R channel
    unsigned char   padding_g;    // Pixel value information of G channel
    unsigned char   padding_b;    // Pixel value information of B channel
};
```

4.3.1 Constructor PaddingAtrr()

Initialize PaddingAtrr

Interface:

```
PaddingAtrr()

PaddingAtrr(
        unsigned int crop_start_x,
        unsigned int crop_start_y,
        unsigned int crop_width,
        unsigned int crop_height,
        unsigned char padding_value_r,
        unsigned char padding_value_g,
        unsigned char padding_value_b);
```

Parameters:

- crop_start_x: int

The offset of the original image relative to the target image in the x direction.

- crop_start_y: int

The offset of the original image relative to the target image in the y direction

- crop_width: int

The original image can be resized while padding. Width is the width of the original image after resize. If no resize is performed, width is the width of the original image.

- crop_height: int

The original image can be resized while padding. Height is the height of the original image after resize. If no resize is performed, height is the height of the original image.

- padding_value_r: int

The pixel value to fill on the R channel when padding.

- padding_value_g: int

The pixel value to fill on the G channel when padding.

- padding_value_b: int

The pixel value to fill on the B channel when padding

4.3.2 set_stx

Set the offset of the original image relative to the target image in the x direction

Interface:

```
void set_stx(unsigned int stx);
```

Parameters:

- stx: int

The offset of the original image relative to the target image in the x direction

4.3.3 set_sty

Set the offset of the original image relative to the target image in the y direction

Interface:

```
void set_sty(unsigned int sty);
```

Parameters:

- sty: int

The offset of the original image relative to the target image in the y direction

4.3.4 set_w

Set the width of the original image after resize

Interface:

```
void set_w(unsigned int w);
```

Parameters:

- width: int

The original image can be resized while padding. Width is the width of the original image after resize. If no resize is performed, width is the width of the original image.

4.3.5 set_h

Set the height of the original image after resize

Interface:

```
void set_h(unsigned int h);
```

Parameters:

- height: int

The original image can be resized while padding. Height is the height of the original image after resize. If no resize is performed, height is the height of the original image.

4.3.6 set_r

Set the padding value on the R channel

Interface:

```
void set_r(unsigned int r);
```

Parameters

- r: int

The padding value on R channel

4.3.7 set_g

Set the padding value on the G channel

Interface:

```
void set_g(unsigned int g);
```

Parameters:

- g: int

The padding value on G channel

4.3.8 set_b

Set the padding value on the B channel

Interface:

```
void set_b(unsigned int b);
```

Parameters

- b: int

The padding value on channel B

4.4 Handle

Handle is a wrapper class for device handles and is used to identify devices in programs.

4.4.1 Constructor

Initialize Handle

Interface:

```
Handle(int tpu_id);
```

Parameter:

- tpu_id: int

Create the ID number of the Tensor Computing Processor used by Handle

4.4.2 get_device_id

Get the id of Tensor Computing Processor in Handle

Interface:

```
int get_device_id();
```

Parameter:

- tpu_id: int

The ID of the Tensor Computing Processor in the Handle.

4.4.3 get_sn

Get the SN(serial number) identifying the device in Handle

Interface:

```
std::string get_sn();
```

Returns:

- serial_number: string

Returns the serial number of the device in Handle

4.4.4 get_target

Get the Tensor Computing Processor model of the device

Interface:

```
std::string get_target();
```

Returns:

- Tensor Computing Processor type: str

Returns the model of the device Tensor Computing Processor

4.5 Tensor

Tensor is the input and output type of model inference, which contains data information and implements memory management.

4.5.1 Constructor

Initialize Tensor and allocate memory for Tensor. If synchronization between system memory and device memory is required, sync_d2s or sync_s2d needs to be executed.

Interface:

```
Tensor(
    const std::vector<int>& shape={},
    bm_data_type_t          dtype=BM_FLOAT32);

Tensor(
    Handle                  handle,
    const std::vector<int>& shape,
    bm_data_type_t          dtype=BM_FLOAT32,
```

(continues on next page)

(continued from previous page)

```
    bool                own_sys_data,
    bool                own_dev_data);
```

Parameters:

- handle: Handle

The device identification Handle.

- shape: std::vector<int>

Set the shape of Tensor.

- dtype: Dtype

The data type of Tensor.

- own_sys_data: bool

Indicates whether the Tensor has system memory.

- own_dev_data: bool

Indicates whether the Tensor has device memory

Sample:

```
#include "tensor.h"

int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1,input_tensor2;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32; // dtype can choose  BM_FLOAT32,
↪ BM_INT8, BM_UINT8, BM_INT32, BM_UINT32

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);
    input_tensor2 = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,F
↪true, true);

    return 0;
}
```

Interface:

This initialization method creates a new Tensor based on an existing source Tensor and reuses a portion of the source Tensor' s device memory without copying device memory. It is suitable for scenarios such as LLM inference where memory reuse is required.

During the use of this Tensor, it is necessary to ensure that the source Tensor is not released.

```
Tensor(const Tensor& src, const std::vector<int>& shape, unsigned int offset)
```

Parameters:

- src: sail::Tensor

The source Tensor used to create the Tensor

- shape: std::vector<int>

The shape of the created Tensor, which is a sequence of integers. The number of elements corresponding to the new shape must not exceed the number of elements in the source Tensor.

- offset: unsigned int

The offset of the Tensor' s device memory relative to the source Tensor' s device memory, in bytes of the dtype.

Example Code:

```
#include <sail/tensor.h>
#include <vector>

int main() {
    sail::Handle handle(0);
    int height = 1080;
    int width = 1920;
    std::vector<int> src_shape = {1, 3, height, width};
    sail::Tensor src_tensor(handle, src_shape, BM_INT32, false, true);

    std::vector<int> dst_shape = {1, 1, height, width};
    unsigned int offset = height * width;
    sail::Tensor dst_tensor(src_tensor, dst_shape, offset);

    return 0;
}
```

4.5.2 shape

Get the shape of Tensor

Interface:

```
const std::vector<int>& shape() const;
```

Parameters:

- tensor_shape : std::vector<int>

Returns a vector containing the shape of the Tensor.

Sample:

```cpp
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // get shape
    std::vector<int> tensor_shape;
    tensor_shape = input_tensor1->shape();
    std::cout << "tensor shape: ";
    for(int i = 0; i < tensor_shape.size(); i++) {
        std::cout << tensor_shape[i] << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

4.5.3 dtype

Get the dtype of Tensor

Interface:

```cpp
bm_data_type_t dtype() const;
```

Return:

- data_type : bm_data_type_t

dtype of Tensor

Sample:

```cpp
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // get dtype
    bm_data_type_t input_dtype_;
```

(continues on next page)

(continued from previous page)

```
    input_dtype_ = input_tensor1->dtype();
    return 0;
}
```

4.5.4 scale_from

First scale the data proportionally, and then update the data to the system memory of Tensor.

Interface:

```
void scale_from(float* src, float scale, int size);
```

Parameters:

- src: float*

The starting address of the data.

- scale: float32

The scale when scaling proportionally.

- size: int

The length of the data.

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1,input_tensor2;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // prepare data
    std::shared_ptr<float> src_ptr(
            new float[10 * 10],
            std::default_delete<float[]>());
    float * src_data = src_ptr.get();
    for(int i = 0; i < 10 * 10; i++) {
        src_data[i] = rand() % 255;
    }

    // scale data len is 99
    input_tensor1->scale_from(src_data, 0.1, 99);
```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

4.5.5 scale_to

First scale the Tensor proportionally and then return the data to the system memory.

Interface:

```
void scale_to(float* dst, float scale);

void scale_to(float* dst, float scale, int size);
```

Parameters:

- dst: float*

The starting address of the data.

- scale: float32

The scale when scaling proportionally.

- size: int

The length of the data.

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // prepare dst
    float* dst = new float[100];

    // scale data len is 99
    input_tensor1->scale_to(dst, 0.1, 99);

    // print scaled data
    for (int i = 0; i < size; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
```

(continues on next page)

(continued from previous page)

```
    delete[] dst;

    return 0;
}
```

4.5.6 reshape

Reshape Tensor

Interface:

```
void reshape(const std::vector<int>& shape);
```

Parameters:

- shape: std::vector<int>

Set the desired new shape.

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor1;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor1 = std::make_shared<sail::Tensor>(input_shape, input_dtype);

    // reshape from 10x10 to 2x50
    input_tensor1->reshape({2,50});

    // get shape
    std::vector<int> tensor_shape;
    tensor_shape = input_tensor1->shape();
    std::cout << "tensor new shape: ";
    for(int i = 0; i < tensor_shape.size(); i++) {
        std::cout << tensor_shape[i] << " ";
    }
    std::cout << std::endl;
    return 0;
}
```


4.5.7 own_sys_data

Query whether the Tensor has a data pointer in system memory.

Interface:

```
bool& own_sys_data();
```

Returns:

- judge_ret: bool

Returns True if it owns the data pointer of system memory, otherwise False.

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true); // own sys mem:true, own dev mem:true
    // input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,F
↪false, true); // own sys mem:true, own dev mem:false

    // input_tensor: own sys or dev data
    bool _own_sys_data = input_tensor->own_sys_data();
    std::cout << "input_tensor own_sys_data:" << _own_sys_data << std::endl;
    return 0;
}
```

4.5.8 own_dev_data

Query whether the Tensor has data in the device memory.

Interface:

```
bool& own_dev_data();
```

Returns:

- judge_ret : bool

Returns True if the Tensor owns the data in device memory, False otherwise.

Sample:

```cpp
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true); // own sys mem:true, own dev mem:true
    // input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,F
↪true, false); // own sys mem:true, own dev mem:false

    // input_tensor: own sys or dev data
    bool _own_dev_data = input_tensor->own_dev_data();
    std::cout << "input_tensor own_dev_data:" << _own_dev_data << std::endl;

    return 0;
}
```

4.5.9 sync_s2d

Copy the data in Tensor from system memory to device memory.

Interface:

```cpp
void sync_s2d();

void sync_s2d(int size);
```

Parameters:

- size: int

Copy data of a specific size bytes from system memory to device memory.

Interface:

```cpp
void sync_s2d(Tensor* src, int offset_src, int offset_dst, int len);
```

Parameters:

- Tensor*: src

Specifies the Tensor to be copied from.

- offset_src: int

Specifies the number of elements to offset in the source Tensor from where to start copying.

- offset_dst: int

Specifies the number of elements to offset in the destination Tensor from where to start copying.

- len: int

Specifies the length of the copy, i.e., the number of elements to copy.

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true); // own sys mem:true, own dev mem:true
    // prepare data
    input_tensor->ones();

    // input_tensor -> sync_s2d(); // copy all data
    input_tensor -> sync_s2d(99); // copy part data

    // prepare another data: output_tensor, which is on sys mem, and don't have data
    // copy input_tensor to output_tensor
    std::shared_ptr<sail::Tensor> output_tensor;
    output_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,F
↪true, true);

    sail::Tensor& input_ref = *input_tensor;
    output_tensor -> sync_s2d(input_ref,2,3,10);

    // test if copy success
    // must copy to system memory and save to dst
    output_tensor -> sync_d2s();
    int size = 100;
    float* dst = new float[size];
    output_tensor->scale_to(dst, 1, size);
    for (int i = 0; i < size; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
    delete[] dst;
    return 0;
}
```

4.5.10 sync_d2s

Copy the data in Tensor from device memory to system memory.

Interface:

```
void sync_d2s();

void sync_d2s(int size);
```

Parameters:

- size: int

Copies data of a specific size bytes from device memory to system memory.

Interface:

```
void sync_d2s(Tensor* src, int offset_src, int offset_dst, int len);
```

Parameters:

- Tensor*: src

Specifies the Tensor to be copied from.

- offset_src: int

Specifies the number of elements to offset in the source Tensor from where to start copying.

- offset_dst: int

Specifies the number of elements to offset in the destination Tensor from where to start copying.

- len: int

Specifies the length of the copy, i.e., the number of elements to copy.

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, false,
↪ true); // own sys mem:false, own dev mem:true

    // prepare data
    input_tensor->ones();
```

(continues on next page)

(continued from previous page)

```
    input_tensor -> sync_d2s(); // copy all data
    // input_tensor -> sync_d2s(99); // copy part data

    // prepare another data: output_tensor, which is on sys mem, and don't have data
    // copy input_tensor to output_tensor
    std::shared_ptr<sail::Tensor> output_tensor;
    output_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,F
↪true, true);

    sail::Tensor& input_ref = *input_tensor;
    output_tensor -> sync_d2s(input_ref,2,3,10);

    // test if copy success
    int size = 100;
    float* dst = new float[size];
    output_tensor->scale_to(dst, 1, size);
    for (int i = 0; i < size; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
    delete[] dst;
    return 0;
}
```

4.5.11 sync_d2d

Copies the data from another Tensor' s device memory to this Tensor' s device memory.

Interface:

```
void sync_d2d(Tensor* src, int offset_src, int offset_dst, int len);
```

Parameters:

- Tensor*: src

Specifies the Tensor to be copied from.

- offset_src: int

Specifies the number of elements to offset in the source Tensor from where to start copying.

- offset_dst: int

Specifies the number of elements to offset in the destination Tensor from where to start copying.

- len: int

Specifies the length of the copy, i.e., the number of elements to copy.

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    sail::Handle handle_(dev_id+1);
    std::shared_ptr<sail::Tensor> input_tensor,output_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, false,
↪ true); // on dev0
    output_tensor = std::make_shared<sail::Tensor>(handle_, input_shape, input_dtype,F
↪false, true); // on dev1
    // prepare data
    input_tensor -> ones();

    // d2d
    sail::Tensor& input_ref = *input_tensor;
    output_tensor -> sync_d2d(input_ref,1,1,10);

    return 0;
}
```

4.5.12 sync_d2d_stride

Copies the data from another Tensor' s device memory to this Tensor' s device memory in stride.

Interface:

```
void sync_d2d_stride(Tensor* src, int stride_src, int stride_dst, int count);
```

Parameters:

- Tensor*: src

Specifies the Tensor to be copied from.

- stride_src: int

Specifies the stride of the the source Tensor.

- stride_dst: int

Specifies the stride of the destination Tensor.stride_dst must be 1, EXCEPT: stride_dst == 4 && stride_src == 1 && Tensor_type_size == 1

- count: int

Specifies the count of elements to copy.Ensure count * stride_src <= tensor_src_size, count * stride_dst <= tensor_dst_size.

4.5.13 dump_data

Write the data in Tensor to the specified file. If synchronization between system memory and device memory is required, sync_d2s needs to be executed.

Interface:

```
void dump_data(std::string file_name, bool bin = false);
```

Parameters:

- file_name: string

The path to the file to write to.

- bin: bool

Whether to store Tensor in binary form, default false.

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;
    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true); // own sys mem:true, own dev mem:true
    // prepare data
    input_tensor->ones();

    input_tensor->dump_data("dumped_tensor.txt",false);
    input_tensor->dump_data("dumped_tensor_bin.bin",true);

    return 0;
}
```

4.5.14 memory_set

Fill the memory of the Tensor with the first N bytes of value, N can be 1, 2, 4, depending on the dtype of the Tensor.

Interface:

```
void memory_set(void* value);
```

Parameters:

- value: void*

the value to fill.

Sample:

```
void test_if_success(int size, std::shared_ptr<sail::Tensor> output_tensor){
    float* dst = new float[size];
    output_tensor->scale_to(dst, 1);
    for (int i = 0; i < 100; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
    delete[] dst;
}

int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {3, 1920, 1080};
    bm_data_type_t input_dtype = BM_FLOAT32;
    input_tensor = std::make_shared<sail::Tensor>(handle,input_shape, input_dtype,true,
↪true);

    // set data
    std::shared_ptr<float> src_ptr(
            new float[3 * 1920 * 1080],
            std::default_delete<float[]>());
    float * src_data = src_ptr.get();
    for(int i = 0; i < 3 * 1920 * 1080; i++) {
        src_data[i] = rand() % 255;
    }
    // print src_data
    for (int i = 0; i < 100; ++i) {
        std::cout << src_data[i] << " ";
    }
    std::cout << std::endl;

    // memory set to tensor
    input_tensor->memory_set(src_data);
    test_if_success(3 * 1920 * 1080,input_tensor);

    return 0;
}
```

4.5.15 memory_set

Fill memory with a scalar, it will be automatically converted to tensor' s dtype. This interface may has precision loss due to data type conversion, It is recommended to use the interface above.

Interface:

```
void memory_set(float c);
```

Parameters:

- c: float

the value to fill.

Sample:

```
void test_if_success(int size, std::shared_ptr<sail::Tensor> output_tensor){
    float* dst = new float[size];
    output_tensor->scale_to(dst, 1);
    for (int i = 0; i < size; ++i) {
        std::cout << dst[i] << " ";
    }
    std::cout << std::endl;
    delete[] dst;
}

int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {1};
    bm_data_type_t input_dtype = BM_FLOAT32;
    input_tensor = std::make_shared<sail::Tensor>(handle,input_shape, input_dtype,true,
↪true);

    float value_ = 1.1;
    input_tensor->memory_set(value_);
    test_if_success(1,input_tensor);

    return 0;
}
```

4.5.16 zeros

fill memory with zeros.

Interface:

```
void zeros();
```

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true);
    // prepare data
    input_tensor->zeros();

    return 0;
}
```

4.5.17 ones

fill memory with ones.

Interface:

```
void ones();
```

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true);
    // prepare data
    input_tensor->ones();
```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

4.5.18 size

Return the number of elements contained in the Tensor.

Interface:

```
int size() const;
```

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true);
    // Output result
    std::cout << input_tensor->size() << " ";

    return 0;
}
```

4.5.19 element_size

Returns the size in bytes of an individual element.

Interface:

```
int element_size() const;
```

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;
```

(continues on next page)

(continued from previous page)

```
    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true);
    // Output result
    std::cout << input_tensor->element_size() << " ";

    return 0;
}
```

4.5.20 nbytes

Return the total number of bytes occupied by all elements of Tensor.

Interface:

```
int nbytes() const;
```

Sample:

```
int main() {
    int dev_id = 0;
    int ret;
    sail::Handle handle(dev_id);
    std::shared_ptr<sail::Tensor> input_tensor;
    std::vector<int> input_shape = {10,10};
    bm_data_type_t input_dtype = BM_FLOAT32;

    // init tensor
    input_tensor = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype, true,
↪ true);
    // Output result
    std::cout << input_tensor->nbytes() << " ";

    return 0;
}
```

4.6 Engine

Engine can load and manage bmodel and is the main module for model reasoning.

4.6.1 Constructor

Initialize Engine

Interface 1:

Create an Engine instance and do not load bmodel

```
Engine(int tpu_id);

Engine(const Handle&   handle);
```

Parameters 1:

- tpu_id: int

Specify the Tensor Computing Processor id used by the Engine instance

- handle: Handle

Specify the device identification Handle used by the Engine instance

Interface 2:

To create an Engine instance and load bmodel, you need to specify the bmodel path or location in memory.

```
Engine(
  const std::string& bmodel_path,
  int                tpu_id,
  IOMode             mode);

Engine(
  const std::string& bmodel_path,
  const Handle&      handle,
  IOMode             mode);

Engine(
   const void* bmodel_ptr,
   size_t      bmodel_size,
   int         tpu_id,
   IOMode      mode);

Engine(
  const void*        bmodel_ptr,
  size_t             bmodel_size,
  const Handle&      handle,
  IOMode             mode);
```

Parameters 2:

- bmodel_path: string

Specify the path to the bmodel file

- tpu_id: int

Specify the Tensor Computing Processor id used by the Engine instance

- mode: IOMode

Specify the memory location where the input/output Tensor is located: system memory or device memory.

- bmodel_ptr: void*

The starting address of bmodel in system memory.

- bmodel_size: size_t

The number of bytes in memory of bmodel

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    sail::Engine engine1(dev_id);
    sail::Engine engine2(handle);
    std::string bmodel_path = "your_bmodel.bmodel";
    sail::Engine engine3(bmodel_path, dev_id, SYSI);
    sail::Engine engine4(bmodel_path, handle, SYSI);

    // Open file input stream
    std::ifstream file(bmodel_path, std::ios::binary);
    // Get file size
    file.seekg(0, std::ios::end);
    size_t bmodel_size = file.tellg();
    file.seekg(0, std::ios::beg);
    // Allocate memory to store model data
    char* bmodel_ptr = new char[bmodel_size];
    // Read file content into memory
    file.read(bmodel_ptr, bmodel_size);
    // Close file input stream
    file.close();
    sail::Engine engine5(bmodel_ptr, bmodel_size, dev_id, sail::SYSI);
    delete [] bmodel_ptr;
    return 0;
}
```


4.6.2 get_handle

Get the device handle sail.Handle used in Engine

Interface:

```
Handle get_handle();
```

Returns:

- handle: Handle

Returns the device handle in Engine.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine1(dev_id);
    sail::Handle handle = engine1.get_handle();
    return 0;
}
```

4.6.3 load

Load bmodel into Engine.

Interface 1:

Specify the bmodel path and load bmodel from the file.

```
bool load(const std::string& bmodel_path);
```

Parameters 1:

- bmodel_path: string

The file path of bmodel.

Interface 2:

Load bmodel from system memory.

```
bool load(const void* bmodel_ptr, size_t bmodel_size);
```

Parameters 2:

- bmodel_ptr: void*

The starting address of bmodel in system memory.

- bmodel_size: size_t

The number of bytes in memory of bmodel.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    if (!engine.load(bmodel_path)) {
        // load failed
        std::cout << "Engine load bmodel "<< bmodel_file << "failed" << "\n";
        exit(0);
    }
    return 0;
}
```

4.6.4 get_graph_names

Get the names of all calculation graphs loaded in Engine.

Interface:

```
std::vector<std::string> get_graph_names();
```

Returns:

- graph_names: std::vector<std::string>

An array of names of all calculation graphs in Engine.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    return 0;
}
```

4.6.5 set_io_mode

Set the memory location where the input/output Tensor of Engine is located: system memory or device memory.

Interface:

```
void set_io_mode(
    const std::string& graph_name,
    IOMode             mode);
```

Parameters:

- graph_name: string

The name of the calculation graph that needs to be configured.

- mode: IOMode

Set the memory location where the input/output Tensor of Engine is located: system memory or device memory.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    engine.set_io_mode(bmodel_names[0], SYSI);
    return 0;
}
```

4.6.6 graph_is_dynamic

Determine whether a selected computational map is dynamic.

Interface:

```
bool graph_is_dynamic(const std::string& graph_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

Returns:

- is_dynamic: bool

A boolean value indicating whether the selected computation graph is dynamic or not.

Sample:

```cpp
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    bool is_dynamic = engine.graph_is_dynamic(bmodel_names[0]);
    return 0;
}
```

4.6.7 get_input_names

Get the names of all input Tensors in the selected calculation graph

Interface:

```cpp
std::vector<std::string> get_input_names(const std::string& graph_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

Returns:

- input_names: std::vector<std::string>

Returns a list of the names of all input Tensors in the selected computation graph.

Sample:

```cpp
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
    return 0;
}
```

4.6.8 get_output_names

Get the names of all output Tensors in the selected calculation graph.

Interface:

```
std::vector<std::string> get_output_names(const std::string& graph_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

Returns:

- output_names: std::vector<std::string>

Returns a list of the names of all output Tensors in the selected calculation graph.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> output_names = engine.get_output_names(bmodel_names[0]);
    return 0;
}
```

4.6.9 get_max_input_shapes

Query the maximum shape corresponding to all input Tensors in the selected calculation graph.

In the static model, the shape of the input Tensor is fixed and should be equal to the maximum shape.

In the dynamic model, the shape of the input Tensor should be less than or equal to the maximum shape.

Interface:

```
std::map<std::string, std::vector<int>> get_max_input_shapes(
    const std::string& graph_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

Returns:

- max_shapes: std::map<std::string, std::vector<int> >

Returns the largest shape in the input Tensor.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::map<std::string, std::vector<int>> input_max_shapes = engine.get_max_input_
↪shapes(bmodel_names[0]);
    return 0;
}
```

4.6.10 get_input_shape

Query the shape of a specific input Tensor in the selected computational graph.

Interface:

```
std::vector<int> get_input_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

- tensor_name: string

The name of the Tensor to be queried.

Returns:

- tensor_shape: std::vector<int>

The shape of the largest dimension in the input Tensor under this name.

Sample:

```
#include "engine.h"

int main() {
```

(continues on next page)

(continued from previous page)

```
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
    std::vector<int> input_shape_0 = engine.get_input_shape(bmodel_names[0],input_
↪names[0]);
    return 0;
}
```

4.6.11 get_max_output_shapes

Query the maximum shape corresponding to all output Tensors in the selected calculation graph.

In the static model, the shape of the output Tensor is fixed and should be equal to the maximum shape.

In the dynamic model, the shape of the output Tensor should be less than or equal to the maximum shape.

Interface:

```
std::map<std::string, std::vector<int>> get_max_output_shapes(
    const std::string& graph_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

Returns:

- std::map<std::string, std::vector<int> >

Returns the largest shape in the output Tensor.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::map<std::string, std::vector<int>> output_max_shapes = engine.get_max_
↪output_shapes(bmodel_names[0]);
```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

4.6.12 get_output_shape

Query the shape of a specific output Tensor in the selected calculation graph.

Interface:

```
std::vector<int> get_output_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

- tensor_name: string

The name of the Tensor to be queried.

Returns:

- tensor_shape: std::vector<int>

The shape of the output Tensor under this name.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> output_names = engine.get_output_names(bmodel_names[0]);
    std::vector<int> output_shape_0 = engine.get_output_shape(bmodel_names[0],output_
↪names[0]);
    return 0;
}
```

4.6.13 get_input_dtype

Get the data type of a specific input Tensor for a specific computational graph.

Interface:

```
bm_data_type_t get_input_dtype(
    const std::string& graph_name,
    const std::string& tensor_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

- tensor_name: string

The name of the Tensor to be queried.

Returns:

- datatype: bm_data_type_t

Returns the data type of the data in the Tensor.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
    std::vector<int> input_dype_0 = engine.get_input_dtype(bmodel_names[0],input_
↪names[0]);
    return 0;
}
```

4.6.14 get_output_dtype

Get the data type of a specific output Tensor for a specific computational graph.

Interface:

```
bm_data_type_t get_output_dtype(
    const std::string& graph_name,
    const std::string& tensor_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

- tensor_name: string

The name of the Tensor to be queried.

Returns:

- datatype: bm_data_type_t

Returns the data type of the data in the Tensor.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
    std::vector<int> output_dype_0 = engine.get_output_dtype(bmodel_names[0],input_
↪names[0]);
    return 0;
}
```

4.6.15 get_input_scale

Get the scale of a specific input Tensor of a specific calculation graph, only valid in the int8 model.

Interface:

```
float get_input_scale(
    const std::string& graph_name,
    const std::string& tensor_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

- tensor_name: string

The name of the Tensor to be queried.

Returns:

- scale: float32

Returns the scale of the Tensor data.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(bmodel_names[0]);
    float input_scale_0 = engine.get_input_scale(bmodel_names[0],input_names[0]);
    return 0;
}
```

4.6.16 get_output_scale

Get the scale of a specific output Tensor of a specific calculation graph, only valid in the int8 model.

Interface:

```
float get_output_scale(
    const std::string& graph_name,
    const std::string& tensor_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

- tensor_name: string

The name of the Tensor to be queried.

Returns:

- scale: float32

Returns the scale of the Tensor data.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> bmodel_name = engine.get_graph_names()[0];
    std::vector<std::string> input_names = engine.get_input_names(bmodel_name);
```

(continues on next page)

(continued from previous page)

```
    float output_scale_0 = engine.get_output_scale(bmodel_names[0],input_names[0]);
    return 0;
}
```

4.6.17 process

Perform forward inference on a specific computational graph.

Interface:

```
void process(
    const std::string&              graph_name,
    std::map<std::string, Tensor*>& input,
    std::map<std::string, Tensor*>& output,
    std::vector<int>                core_list = {});

void process(
    const std::string&                      graph_name,
    std::map<std::string, Tensor*>&         input,
    std::map<std::string, std::vector<int>>& input_shapes,
    std::map<std::string, Tensor*>&         output,
    std::vector<int>                        core_list = {});
```

Parameters:

- graph_name: string

Input parameter. A specific computational graph name.

- input: std::map<std::string, Tensor*>

Input parameter. All input Tensor data.

- input_shapes : std::map<std::string, std::vector<int> >

Input parameter. All shapes passed into Tensor.

- output: std::map<std::string, Tensor*>

Output parameter. All output Tensor data.

- core_list: std::vector<int>

Input parameter. This parameter is only valid for processors that support multi-core inference, and the core used for inference can be selected. Set bmodel as the corresponding kernel number N, and if corelist is empty, use N cores starting from core0 for inference; If the length of corelist is greater than N, use the corresponding top N cores in corelist for inference. This parameter can be ignored for processors that only support single core inference.

Sample:

```cpp
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(graph_names[0]);
    std::vector<std::string> output_names = engine.get_input_names(graph_names[0]);

    std::vector<int> input_shape, output_shape;
    bm_data_type_t input_dtype, output_dtype;
    // allocate input and output tensors with both system and device memory
    // or you can use engine.create_input\output_tensors_map to create
    for (int i = 0; i < input_names.size(); i++) {
        input_shape = engine.get_input_shape(graph_names[0], input_names[i]);
        input_dtype = engine.get_input_dtype(graph_names[0], input_names[i]);
        input_tensor[i] = std::make_shared<sail::Tensor>(handle, input_shape, input_dtype,
↪ true, true);
        input_tensors[input_names[i]] = input_tensor[i].get();
    }
    for (int i = 0; i < output_names.size(); i++) {
        output_shape = engine.get_output_shape(graph_names[0], output_names[i]);
        output_dtype = engine.get_output_dtype(graph_names[0], output_names[i]);
        output_tensor[i] = std::make_shared<sail::Tensor>(handle, output_shape[i], output_
↪dtype, true, true);
        output_tensors[output_names[i]] = output_tensor[i].get();
    }

    // process1
    engine.process(graph_names[0], input_tensors, output_tensors);

    // process2
    std::map<std::string, std::vector<int>> input_shapes;
    for (const auto& input_name : input_names) {
        input_shape = engine.get_input_shape(bmodel_names, input_name);
        input_shapes[input_name] = input_shape;
    }

    engine.process(graph_names[0], input_tensors, input_shapes, output_tensors);
    return 0;
}
```


4.6.18 get_device_id

Get the device ID number in Engine

Interface:

```
int get_device_id() const;
```

Parameters:

- tpu_id : int

Returns the device ID number in Engine.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Engine engine(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    engine.load(bmodel_path);
    int dev = engine.get_device_id();
    return 0;
}
```

4.6.19 create_input_tensors_map

Create a map of the input Tensor

Interface:

```
std::map<std::string, Tensor*> create_input_tensors_map(
    const std::string& graph_name,
    int create_mode = -1);
```

Parameters:

- graph_name: string

The name of specific computational graph.

- create_mode: int

Create a pattern for Tensor to allocate memory. When it is 0, only system memory is allocated. When it is 1, only device memory is allocated. Otherwise, it is allocated according to the IOMode configuration in Engine.

Returns:

input: std::map<std::string, Tensor*>

Returns the mapping of strings to tensors.

Sample:

```
#include "engine.h"


int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
    sail::Engine engine(bmodel_path, dev_id, sail::IOMode::SYSIO);
    std::string graph_name = engine.get_graph_names()[0];
    std::map<std::string, sail::Tensor> input_tensors_map = engine.create_input_tensors_
↪map(graph_name);
    return 0;
}
```

4.6.20 create_output_tensors_map

Create a mapping of the input Tensor, which is a dictionary dict{string : Tensor} in the python interface

Interface:

```
std::map<std::string, Tensor*> create_output_tensors_map(
    const std::string& graph_name,
    int create_mode = -1);
```

Parameters:

- graph_name: string

The name of specific computational graph.

- create_mode: int

Create a pattern for Tensor to allocate memory. When it is 0, only system memory is allocated. When it is 1, only device memory is allocated. Otherwise, it is allocated according to the IOMode configuration in Engine.

Returns:

output: std::map<std::string, Tensor*>

Returns a mapping of strings to tensors.

Sample:

```
#include "engine.h"

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string bmodel_path = "your_bmodel.bmodel";
```

(continues on next page)

(continued from previous page)

```
    sail::Engine engine(bmodel_path, dev_id, sail::IOMode::SYSIO);
    std::string graph_name = engine.get_graph_names()[0];
    std::map<std::string, sail::Tensor> output_tensors_map = engine.create_output_
↪tensors_map(graph_name);
    return 0;
}
```

4.7 MultiEngine

A multi-threaded inference engine that implements multi-threaded inference for specific calculation graphs.

4.7.1 MultiEngine

Initialize MutiEngine。

Interface:

```
MultiEngine(const std::string& bmodel_path,
        std::vector<int>   tpu_ids,
        bool               sys_out=true,
        int                graph_idx=0);
```

Parameters:

- bmodel_path: string

The file path where bmodel is located.

- tpu_ids: std::vector<int>

The ID of the Tensor Computing Processor visible in this MultiEngine.

- sys_out: bool

Whether to copy the results to system memory, the default is True

- graph_idx : int

The index of a specific computational graph.

Sample:

```
#include "engine_multi.h"

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

4.7.2 set_print_flag

Set whether to print debugging information.

Interface:

```
void set_print_flag(bool print_flag);
```

Parameters:

- print_flag: bool

When True, debugging information is printed, otherwise it is not printed.

Sample:

```
#include "engine_multi.h"

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    engine.set_print_flag(True);
    return 0;
}
```

4.7.3 set_print_time

Setting whether to print is mainly time-consuming.

Interface:

```
void set_print_time(bool print_flag);
```

Parameters:

- print_flag: bool

When it is True, printing is mainly time-consuming, otherwise it will not print.

Sample:

```
#include "engine_multi.h"

int main() {
    std::vector<int> dev_id = {0, 1};
```

(continues on next page)

(continued from previous page)

```
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    engine.set_print_time(True);
    return 0;
}
```

4.7.4 get_device_ids

Get the IDs of all available Tensor Computing Processors in MultiEngine.

Interface:

```
std::vector<int> get_device_ids();
```

Returns:

- device_ids: std::vector<int>

Returns the IDs of visible Tensor Computing Processors

Sample:

```
#include "engine_multi.h"

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<int> device_ids = engine.get_device_ids();
    return 0;
}
```

4.7.5 get_graph_names

Get the names of all loaded calculation graphs in MultiEngine.

Interface:

```
std::vector<std::string> get_graph_names();
```

Returns:

- graph_names: std::vector<std::string>

The list of names of all calculation graphs in MultiEngine.

Sample:

```cpp
#include "engine_multi.h"

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    return 0;
}
```

4.7.6 get_input_names

Get the names of all input Tensors in the selected calculation graph

Interface:

```cpp
std::vector<std::string> get_input_names(const std::string& graph_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

Returns:

- input_names: std::vector<std::string>

Returns a list of the names of all input Tensors in the selected computation graph.

Sample:

```cpp
#include "engine_multi.h"

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(graph_names[0]);
    return 0;
}
```

4.7.7 get_output_names

Get the names of all output Tensors in the selected calculation graph.

Interface:

```
std::vector<std::string> get_output_names(const std::string& graph_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

Returns:

- output_names: std::vector<std::string>

Returns a list of the names of all output Tensors in the selected calculation graph.

Sample:

```
#include "engine_multi.h"

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> output_name = engine.get_output_names(graph_names[0]);
    return 0;
}
```

4.7.8 get_input_shape

Query the shape of a specific input Tensor in the selected computational graph.

Interface:

```
std::vector<int> get_input_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

- tensor_name: string

The name of the Tensor to be queried.

Returns:

- tensor_shape: std::vector<int>

The shape of the largest dimension in the input Tensor under this name.

Sample:

```
#include "engine_multi.h"

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(graph_names[0]);
    std::vector<int>  = engine.get_input_shape(graph_names[0],input_names[0]);
    return 0;
}
```

4.7.9 get_output_shape

Query the shape of a specific output Tensor in the selected calculation graph.

Interface:

```
std::vector<int> get_output_shape(
    const std::string& graph_name,
    const std::string& tensor_name);
```

Parameters:

- graph_name: string

Set the name of the calculation graph to be queried.

- tensor_name: string

The name of the Tensor to be queried.

Returns:

- tensor_shape: std::vector<int>

The shape of the output Tensor under this name.

Sample:

```
#include "engine_multi.h"

int main() {
    std::vector<int> dev_id = {0, 1};
    std::string bmodel_path = "your_bmodel.bmodel"
    sail::MultiEngine engine(bmodel_path, dev_id);
    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> output_names = engine.get_output_names(graph_names[0]);
```

(continues on next page)

(continued from previous page)

```
    std::vector<int> output_shape = engine.get_output_shape(graph_names[0],output_
↪names[0]);
    return 0;
}
```

4.7.10 process

Performing inference on a specific computational graph requires input data from system memory.

Interface:

```
std::vector<std::map<std::string, Tensor*>> process(std::vector<std::map<std::string,
↪Tensor*>>& input_tensors);
```

Parameters:

- input_tensors: std::vector<std::map<std::string, Tensor*> >

The input Tensors.

Returns:

- output_tensors: std::vector<std::map<std::string, Tensor*> >

Returns the result after inference.

Sample:

```
#include <sail/engine_multi.h>

int main() {
    std::vector<int> dev_id = {0, 1};

    std::string bmodel_path = "/home/jingyu/SAM-ViT-B_embedding_fp16_1b.bmodel";
    sail::MultiEngine engine(bmodel_path, dev_id);


    std::vector<std::string> graph_names = engine.get_graph_names();
    std::vector<std::string> input_names = engine.get_input_names(graph_names[0]);

    std::vector<int> input_shape = engine.get_input_shape(graph_names[0], input_
↪names[0]);

    // prepare one input tensor
    std::map<std::string, sail::Tensor*> input_tensors_map1;
    for (const auto& input_name : input_names) {
        sail::Tensor* input_tensor = new sail::Tensor(input_shape);
        input_tensors_map1[input_name] = input_tensor;
    }
    // prepare multi input...
```

(continues on next page)

(continued from previous page)

```
    std::vector<std::map<std::string, sail::Tensor*>> input_tensors_vector;
    input_tensors_vector.push_back(input_tensors_map1);

    // get multi output
    auto output_tensors_vector = engine.process(input_tensors_vector);

    for(auto& pair : input_tensors_map1) {
        delete pair.second;
    }
    return 0;
}
```

4.8 bm_image

bm_image is the basic structure in BMCV, which encapsulates the main information of an image and is the internal element of subsequent BMImage and BMImageArray.

struct:

```
struct bm_image {
    int width;
    int height;
    bm_image_format_ext image_format;
    bm_data_format_ext data_type;
    bm_image_private* image_private;
};
```

bm_image struct contains width,height,image_format,data_type and its private data.

Sample:

```
#include "cvwrapper.h"

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_image.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg = decoder.read(handle);
    // Convert BMImage to bm_image data structure; here is a bm_image
    struct bm_image img = BMimg.data();

    // print image width, height, format and dtype
    std::cout << img.width << " " << img.height << " " << img.image_format << " " <
↪< img.data_type << std::endl;

    return 0;
}
```

4.9 BMImage

BMImage encapsulates all the information of an image, and the Bmcv interface can be used to convert the BMImage into a Tensor for model inference.

BMImage is also the basic data type for other image processing operations through the Bmcv interface.

4.9.1 Constructor

Initialize BMImage.

Interface:

```
BMImage();

BMImage(
    Handle&                 handle,
    int                     h,
    int                     w,
    bm_image_format_ext     format,
    bm_image_data_format_ext dtype);

BMImage(
    Handle                  &handle,
    void*                   buffer,
    int                     h,
    int                     w,
    bm_image_format_ext     format,
    bm_image_data_format_ext dtype = DATA_TYPE_EXT_1N_BYTE,
    std::vector<int>        strides = {},
    size_t                  offset = 0);
```

Parameters:

- handle: Handle

Set the device handle where the BMImage is located.

- h: int

The height of the image.

- w: int

The width of the image.

- format : bm_image_format_ext

The format of the image. Supported formats are listed in sail.Format .

- dtype: bm_image_data_format_ext

The data type of the image. Supported data types are listed in ImgDtype .

- buffer: bytes | np.array

The address of the buffer when creating an image with a buffer.

- strides

The stride of the image when creating an image with a buffer. The unit is in bytes. The default is empty, indicating that it is the same as the data width of one row. If specified, ensure the number of elements in the list matches the number of image planes.

- offset

The offset of valid data relative to the start address of the buffer when creating an image with a buffer. The unit is in bytes, and the default is 0.

4.9.2 width

Get the width of the image.

Interface:

```
int width();
```

Returns:

- width : int

Returns the width of the image.

4.9.3 height

Get the height of the image.

Interface:

```
int height();
```

Returns:

- height : int

Returns the height of the image.

4.9.4 format

Get the format of the image.

Interface:

```
bm_image_format_ext format();
```

Returns:

- format : bm_image_format_ext

Returns the format of the image.

4.9.5 dtype

Get the data type of the image.

Interface:

```
bm_image_data_format_ext dtype() const;
```

Returns:

- dtype: bm_image_data_format_ext

Returns the data type of the image.

4.9.6 data

Get bm_image inside BMImage.

Interface:

```
bm_image& data();
```

Returns:

- img : bm_image

Returns bm_image inside the image.

4.9.7 get_device_id

Get the device id number in BMImage.

Interface:

```
int get_device_id() const;
```

Returns

- device_id : int

Returns the device id in BMImage

4.9.8 get_handle

Get Handle of the BMImage.

Interface:

```
Handle get_handle();
```

Return:

- Handle : Handle

Return the Handle of BMImage.

4.9.9 get_plane_num

Get the number of image planes in BMImage.

Interface:

```
int get_plane_num() const;
```

Returns:

- planes_num : int

Returns the number of image planes in BMImage.

4.9.10 align

Align BMImage as 64 bits

Interface:

```
int align();
```

Returns:

- ret : int

Returns whether the BMImage is successfully aligned, -1 represents failure, 0 represents success

4.9.11 check_align

Get whether the image in BMImage is aligned

Interface:

```
bool check_align()const;
```

Returns:

- ret : bool

1 means aligned, 0 means not aligned

4.9.12 unalign

unalign the BMImage

Interface:

```
int unalign();
```

Returns:

- ret : int

Returns whether the BMImage is successfully unaligned, -1 means failure, 0 means success

4.9.13 check_contiguous_memory

Get whether the image memory in BMImage is continuous

Interface:

```
bool check_contiguous_memory()const;
```

Returns:

- ret : bool

1 means continuous, 0 means discontinuous

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_image.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg = decoder.read(handle);
```

(continues on next page)

(continued from previous page)

```
    // Get the image information
    int width = BMimg.width();
    int height = BMimg.height();
    bm_image_format_ext format = BMimg.format();
    bm_image_data_format_ext dtype = BMimg.dtype();

    // Convert BMImage to bm_image data structure
    bm_image bmimg = BMimg.data();

    // Get the device id and handle
    int device_id = BMimg.get_device_id();
    sail::Handle handle_ = BMimg.get_handle();
    int plane_num = BMimg.get_plane_num();
    std::cout << "Width: " << width << ", Height: " << height << ", Format: " <<F
↪format << ", Data Type: " << dtype << ", Device ID: " << device_id << ", PlaneF
↪Num: " << plane_num << std::endl;

    int ret;
    // Align the image
    ret = BMimg.align();
    if (ret != 0) {
        std::cout << "Failed to align the image!" << std::endl;
    }
    std::cout << "is align: " << BMimg.check_align() << std::endl;

    // unalign the image
    ret = BMimg.unalign();
    if (ret != 0) {
        std::cout << "Failed to unalign the image!" << std::endl;
    }
    std::cout << "is align: " << BMimg.check_align() << std::endl;

    // check contiguous memory
    std::cout << "is continues: " <<BMimg.check_contiguous_memory()<< std::endl;

    // create BMImage with data from buffer
    std::vector<uint8_t> buf(200 * 100 * 3);
    for (int i = 0; i < 200 * 100 * 3; ++i) {
        buf[i] = i % 256;
    }
    sail::BMImage img_fromRawdata(handle, buf.data(), 200, 100, sail::Format::FORMAT_
↪BGR_PACKED);

    return 0;
}
```

4.9.14 get_pts_dts

Get pts or dts.

Interface:

```
vector<double> get_pts_dts()
```

Returns

- result : int

the value of pts and dts.

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Handle handle(tpu_id);
    Decoder decoder(file_path, true, tpu_id);
    BMImage image;

    int ret = decoder.read(handle, image);
    if (ret != 0) {
        cout << "Failed to read a frame!" << endl;
        return ret;
    }

    std::vector<int> pts_dts;
    pts_dts = image.get_pts_dts();
    cout << "pts: " << pts_dts[0] << endl;
    cout << "dts: " << pts_dts[1] << endl;
    return 0;
}
```

4.10 BMImageArray

BMImageArray is an array of BMImage, which can apply for continuous memory space for multiple images.

When declaring BMImageArray, developer need to specify different instances according to the number of images.

Example: The construction method of BMImageArray when there are 4 images is as follows: images = BMImageArray<4>()

4.10.1 Constructor

Initialize BMImageArray.

Interface:

```
BMImageArray();

BMImageArray(
    Handle                  &handle,
    int                     h,
    int                     w,
    bm_image_format_ext     format,
    bm_image_data_format_ext dtype);
```

Parameters:

- handle: Handle

Set the device handle where the BMImage is located.

- h: int

The height of the image.

- w: int

The width of the image.

- format : bm_image_format_ext

The format of the image.

- dtype: bm_image_data_format_ext

The data type of the image.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    sail::Handle handle = sail::Handle(0);
    std::string image_path = "your_image.jpg"
    sail::Decoder decoder(image_path,false,0);
    sail::BMImage image;
    decoder.read(handle,image);

    // Create an instance of BMImageArray
    sail::BMImageArray<4> images = sail::BMImageArray<4>(handle,image.width(),image.
↪height(),image.format(),image.dtype());

    return 0;
}
```

4.10.2 copy_from

Copies the image to a specific index.

Interface:

```
int copy_from(int i, BMImage &data);
```

Parameters:

- i: int

Enter the index to be copied.

- data: BMImage

The image data that needs to be copied.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    sail::Handle handle = sail::Handle(0);
    std::string image_path = "your_image.jpg"
    sail::Decoder decoder(image_path,false,0);
    sail::BMImage image;
    decoder.read(handle,image);

    // Create an instance of BMImageArray
    sail::BMImageArray<4> images = sail::BMImageArray<4>(handle,image.width(),image.
↪height(),image.format(),image.dtype());
    // copy from BMImage
    int ret = images.copy_from(0,image);
    if (ret != 0) {
        std::cout << "copy_from failed" << std::endl;
        return -1;
    }
    return 0;
}
```

4.10.3 attach_from

Attach the image to a specific index. There is no memory copy, therefore, the original data needs to be cached.

Interface:

```
int attach_from(int i, BMImage &data);
```

Sample:

```cpp
#include <sail/cvwrapper.h>

int main() {
    sail::Handle handle = sail::Handle(0);
    std::string image_path = "your_image.jpg"
    sail::Decoder decoder(image_path,false,0);
    sail::BMImage image;
    decoder.read(handle,image);

    // Create an instance of BMImageArray
    sail::BMImageArray<4> images = sail::BMImageArray<4>(handle,image.width(),image.
↪height(),image.format(),image.dtype());
    // attach from BMImage
    ret = images.attach_from(1,image);
    if (ret != 0) {
        std::cout << "attach_from failed" << std::endl;
        return -1;
    }
    return 0;
}
```

4.10.4 get_device_id

Get the device number in BMImageArray.

Parameters:

```cpp
int get_device_id();
```

Returns:

- device_id: int

Device id number in BMImageArray.

Sample:

```cpp
#include <sail/cvwrapper.h>

int main() {
    sail::Handle handle = sail::Handle(0);
    std::string image_path = "/data/jinyu.lu/jpu_test/1920x1080_yuvj420.jpg";
    sail::Decoder decoder(image_path,false,0);
    sail::BMImage image;
    decoder.read(handle,image);

    // Create an instance of BMImageArray
    sail::BMImageArray<4> images = sail::BMImageArray<4>(handle,image.height(),image.
↪width(),image.format(),image.dtype());

    // get devid
```

(continues on next page)

(continued from previous page)

```
    int devid = images.get_device_id();
    std::cout << "devid: " << devid << std::endl;

    return 0;
}
```

4.11 Decoder

Decoder, which can decode images or videos.

4.11.1 Constructor

Initialize Decoder。

Interface:

```
Decoder(
    const std::string& file_path,
    bool               compressed = true,
    int                tpu_id = 0);
```

Parameters:

- file_path: str

The path or RTSP URL of the image or video file.

- compressed: bool

Whether to compress the decoded output to NV12, default is True

- tpu_id: int

Set the ID of Tensor Computing Processor.

4.11.2 is_opened

Judge if the source is opened successfully.

Interface:

```
bool is_opened();
```

Returns:

- judge_ret: bool

Returns True if the opening is successful and False if it fails.

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Decoder decoder(file_path, true, tpu_id);
    if (!decoder.is_opened()) {
        cout << "Failed to open the file!" << endl;
        return -1;
    }
    return 0;
}
```

4.11.3 read

Read a frame of image from the Decoder.

Interface:

```
int read(Handle& handle, BMImage& image);
```

Parameters:

- handle: Handle

Input parameter. Handle of Tensor Computing Processor used by Decoder.

- image: BMImage

Output parameter. Read data into image.

Returns:

- judge_ret: int

Returns 0 if the read is successful and other values if failed.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Handle handle(tpu_id);
```

(continues on next page)

(continued from previous page)

```
    Decoder decoder(file_path, true, tpu_id);
    BMImage image;

    int ret = decoder.read(handle, image);
    if (ret != 0) {
        cout << "Failed to read a frame!" << endl;
        return ret;
    }
    return 0;
}
```

4.11.4 read_

Read a frame of image from the Decoder.

Interface:

```
int read_(Handle& handle, bm_image& image);
```

Parameters:

- handle: Handle

Input parameter. Handle of Tensor Computing Processor used by Decoder.

- image: bm_image

Output parameter. Read data into image.

Returns:

- judge_ret: int

Returns 0 if the read is successful and other values if failed.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Handle handle(tpu_id);
    Decoder decoder(file_path, true, tpu_id);
    BMImage image;
    bm_image  bm_img = image.data();
    int ret = decoder.read_(handle, bm_img);
    if (ret != 0) {
```

(continues on next page)

(continued from previous page)

```
        cout << "Failed to read a frame!" << endl;
        return ret;
    }
    return 0;
}
```

4.11.5 get_frame_shape

Get the shape in the frame in the Decoder.

Interface:

```
std::vector<int> get_frame_shape();
```

Returns:

- frame_shape: std::vector<int>

Returns the shape of the current frame.

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Decoder decoder(file_path, true, tpu_id);
    vector<int> frame_shape = decoder.get_frame_shape();

    for (auto dim : frame_shape) {
        cout << dim << " ";
    }
    cout << endl;

    return 0;
}
```

4.11.6 release

Release Decoder resources.

Interface:

```
void release();
```

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Decoder decoder(file_path, true, tpu_id);
    decoder.release();
    return 0;
}
```

4.11.7 reconnect

Decoder connects again.

Interface:

```
int reconnect();
```

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;
    Decoder decoder(file_path, true, tpu_id);
    if (decoder.reconnect() != 0) {
        cout << "Reconnect failed!" << endl;
        return -1;
    }
    return 0;
}
```

4.11.8 enable_dump

Enable the dump input video function (without encoding) of decoder and cache up to 1000 frames of undecoded video.

Interface:

```
void enable_dump(int dump_max_seconds):
```

Parameters:

- dump_max_seconds: int

Input parameter. The maximum duration of the dump video is also the maximum length of the internal AVpacket cache queue.

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;
    int dump_max_seconds = 100;

    Decoder decoder(file_path, true, tpu_id);
    decoder.enable_dump(dump_max_seconds);

    return 0;
}
```

4.11.9 disable_dump

Turn off the dump input video function of decoder and clear the cached video frames when this function is turned on.

Interface:

```
void disable_dump():
    """ Disable  input video dump without encode.
    """
```

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;
```

(continues on next page)

(continued from previous page)

```
int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Decoder decoder(file_path, true, tpu_id);
    decoder.enable_dump(100);
    decoder.disable_dump();

    return 0;
}
```

4.11.10 dump

At the time of calling this function, dump the input video for several seconds before and after. Due to the lack of encoding, it is necessary to dump the keyframes that all frames depend on within a few seconds before and after. Therefore, the dump implementation of the interface is based on gop, and the actual video duration under dump will be higher than the input parameter duration. The error depends on the gop of the input video. The larger the size and gop, the larger the error.

Interface:

```
int dump(int dump_pre_seconds, int dump_post_seconds, std::string& file_path)
```

- dump_pre_seconds: int

Input parameter. Save the video several seconds before calling this interface.

- dump_post_seconds: int

Input parameter. Save the video a few seconds after the time this interface is called.

- file_path: std::string&

Input parameter. Video path.

Returns:

- judge_ret: int

Returns 0 if successful and other values if failed.

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
```

(continues on next page)

(continued from previous page)

```
    string file_path = "your_video_file_path.mp4";
    string output_file_path = "output_video_path.mp4";
    int tpu_id = 0;
    int dump_pre_seconds = 3;
    int dump_post_seconds = 3;

    Decoder decoder(file_path, true, tpu_id);
    int ret = decoder.dump(dump_pre_seconds, dump_post_seconds, output_file_path);
    if (ret != 0) {
        cout << "Dump failed with error code: " << ret << endl;
        return ret;
    }

    return 0;
}
```

4.11.11 get_pts_dts

Get pts or dts.

Interface:

```
vector<double> get_pts_dts()
```

Returns

- result : int

the value of pts and dts.

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
using namespace sail;

int main() {
    string file_path = "your_video_file_path.mp4";
    int tpu_id = 0;

    Handle handle(tpu_id);
    Decoder decoder(file_path, true, tpu_id);
    BMImage image;

    int ret = decoder.read(handle, image);
    if (ret != 0) {
        cout << "Failed to read a frame!" << endl;
        return ret;
    }
```

(continues on next page)

(continued from previous page)

```
    std::vector<int> pts_dts;
    pts_dts = decoder.get_pts_dts();
    cout << "pts: " << pts_dts[0] << endl;
    cout << "dts: " << pts_dts[1] << endl;
    return 0;
}
```

4.12 Encoder

Encoder can encode images or videos, save video files, and push rtsp/rtmp streams.

4.12.1 Constructor

Initialize Encoder

Image encoder initialization

Image Encoder Interface:

```
Encoder();
```

Vedio encoder initialization

Vedio Encoder Interface 1:

```
Encoder(const std::string &output_path,
        Handle &handle,
        const std::string &enc_fmt,
        const std::string &pix_fmt,
        const std::string &enc_params,
        int cache_buffer_length=5,
        int abort_policy=0);
```

Vedio Encoder Interface 2:

```
Encoder(const std::string &output_path,
        int device_id,
        const std::string &enc_fmt,
        const std::string &pix_fmt,
        const std::string &enc_params,
        int cache_buffer_length=5
        int abort_policy=0);
```

Parameters:

- output_path: string

Input parameter. Encoded video output path, supports local files (MP4, ts, etc.) and rtsp/rtmp streams. * handle: sail.Handle

Input parameter. Encoder handle instance. (Choose one between handle and device_id)

- device_id: int

Input parameter. Encoder device_id. (Choose one of device_id and handle. When device_id is specified, Handle will be created inside the encoder)

- enc_fmt: string

Input parameter. Encoding format, supports h264_bm and h265_bm/hevc_bm.

- pix_fmt: string

Input parameters. The pixel format of the encoding output supports NV12 and I420. I420 is recommed.

- enc_params: string

Input parameter. Encoding parameters, "width=1902:height=1080:gop=32:gop_preset=3:framerate=25:bitr
where width and height are required. By default, bitrate is used to control quality. Bitrate is invalid when qp is specified in the parameter. .

- cache_buffer_length: int

Input parameter. Internal cache queue length, default is 5. sail.Encoder internally maintains a cache queue to improve flow control fault tolerance when pushing streams.

- abort_policy: int

Input parameter. The rejection policy of the video_write interface when the cache queue is full. When set to 0, the video_write interface returns -1 immediately. When set to 1, pop the queue head. When set to 2, the queue is cleared. When set to 3, it blocks until the encoding thread consumes one frame and the queue becomes empty.

4.12.2 is_opened

Determine whether the encoder is turned on.

Interface:

```
bool is_opened();
```

Returns:

- judge_ret: bool

Returns True if the encoder is turned on and False if it fails.

Sample:

```
#include <sail/encoder.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string out_path = "path/to/your/output/file";
    string enc_fmt = "h264_bm";
    string pix_fmt = "I420";
    string enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25";
    int cache_buffer_length = 5;
    int abort_policy = 0;
    sail::Encoder encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy);
    if(encoder.is_opened())
    {
        cout<<"succeed!"<<endl;
    }
    return 0;
}
```

4.12.3 pic_encode

Encode an image and return the encoded data.

Interface 1:

```
int pic_encode(std::string& ext, bm_image &image, std::vector<u_char>& data);
```

Interface 2:

```
int pic_encode(std::string& ext, BMImage &image, std::vector<u_char>& data);
```

Parameters:

- ext: string

Input parameter. Image encoding format. ".jpg", ".png" etc.

- image: bm_image/BMImage

Input parameter. Input pictures, only FORMAT_BGR_PACKED, DATA_TYPE_EXT_1N_BYTE pictures are supported.

- data: vector<u_char>

Input parameter. Byte vector,a container for holding the data encoded into system memory.

Returns:

- size: int

The size of the encoded data held in system memory.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/encoder.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string image_path = "your_img_path";
    sail::Decoder decoder(image_path,false,dev_id);
    sail::BMImage img = decoder.read(handle);

    string out_path = "path/to/your/output/file";
    string enc_fmt = "h264_bm";
    string pix_fmt = "I420";
    string enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25";
    int cache_buffer_length = 5;
    int abort_policy = 0;
    sail::Encoder encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy);

    vector<u_char> data;
    string extension = ".jpg";
    int size = encoder.pic_encode(extension,img,data);
    //int size = encoder.pic_encode(extension,img.data(),data); //bm_image

    return 0;
}
```

4.12.4 video_write

Send a frame of image to the video encoder. Asynchronous interface, after format conversion, is put into the internal cache queue.

Interface 1:

```
int video_write(bm_image &image);
```

Interface 2:

```
int video_write(BMImage &image);
```

Parameters:

- image: bm_image/BMImage

On the BM1684, when the pixel format (pix_fmt) of the encoder is set to I420, the shape of the image to be encoded can differ from the encoder' s width and height. However, when the

pixel format is NV12, the image shape must match the encoder' s dimensions. In this case, a format conversion is performed internally using bmcv_image_storage_convert, which may utilize NPU resources.

On the BM1684X, the shape of the image to be encoded can differ from the encoder' s width and height. The internal resizing and format conversion are handled by bmcv_image_vpp_convert.

Returns:

- judge_ret: int

Returns 0 on success, -1 when the internal cache queue is full. -2 is returned when there is a frame in the internal buffer queue that fails to encode. One frame was successfully encoded, but failed to push and returned -3. Unknown deny policy returns -4.

Sample:

```
#include<sail/cvwrapper.h>
#include <sail/encoder.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string image_path = "your_img_path";
    sail::Decoder decoder(image_path,false,dev_id);
    sail::BMImage img = decoder.read(handle);
    string out_path = "out_put_path";
    string enc_fmt = "h264_bm";
    string pix_fmt = "I420";
    string enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25";
    int cache_buffer_length = 5;
    int abort_policy = 0;
    sail::Encoder encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy);
    int ret = encoder.video_write(img);
    // int ret = encoder.video_write(img.data());  //bm_image
    return 0;
}
```

4.12.5 release

Release the encoder.

Interface:

```
void release();
```

Sample:

```cpp
#include <sail/encoder.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string out_path = "path/to/your/output/file";
    string enc_fmt = "h264_bm";
    string pix_fmt = "I420";
    string enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25";
    int cache_buffer_length = 5;
    int abort_policy = 0;
    sail::Encoder encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy);
    encoder.release();
    return 0;
}
```

4.13 Decoder_RawStream

Original stream decoder for H264/H265 decoding。

4.13.1 Constructor

Initialize Decoder。

Interface:

```cpp
Decoder_RawStream(int tpu_id, string  decformt);
```

Parameters:

- tpu_id: int

The Tensor Computing Processor id that used, which defaults to 0.

- decformat: string

Input image format, supports h264 and h265.

4.13.2 read

Read a frame of image from Decoder.

Interface:

```
int read(uint8_t* data, int data_size, sail::BMImage &image,bool continueFrame = false);
```

Parameters:

- data: uint8_t*

Input parameter. The binary data of the original stream.

- image: BMImage

Output parameter. Read data into the BMImage.

- continueFrame: bool

Input parameter. Whether to read frames continuously, the default is False.

Returns:

- judge_ret: int

Returns 0 if the read is successful and other values if failed.

4.13.3 read_

Read a frame of image from Decoder.

Interface:

```
int read_(uint8_t* data, int data_size, bm_image &image,bool continueFrame = false);
```

Parameters:

- data: uint8_t*

Input parameter. The binary data of the original stream.

- image: bm_image

Output parameter. Read data into the bm_image.

- continueFrame: bool

Input parameter. Whether to read frames continuously, the default is False.

Returns:

- judge_ret: int

Returns 0 if the read is successful and other values if failed.

4.13.4 release

Release Decoder resources.

Interface:

```
void release();
```

4.14 Bmcv

Bmcv encapsulates commonly used image processing interfaces and supports hardware acceleration.

4.14.1 The constructor of Bmcv()

Init Bmcv

Interface:

```
Bmcv(Handle handle);
```

Parameters:

- handle: Handle

Specify the device handle used by Bmcv.

4.14.2 bm_image_to_tensor

Convert BMImage/BMImageArray to Tensor.

Interface 1:

```
void bm_image_to_tensor(BMImage &img, Tensor &tensor);

Tensor bm_image_to_tensor(BMImage &img);
```

Parameters 1:

- image: BMImage

Input parameter. Image image that needs to be converted

- tensor: Tensor

The converted Tensor.

Return 1:

- tensor: Tensor

Returns the converted Tensor.

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string image_path = "your_image_path";
    sail::Decoder decoder(image_path,false,dev_id);
    sail::BMImage img = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Tensor tensor = bmcv.bm_image_to_tensor(img);
    return 0;
}
```

Interface 2:

```
def bm_image_to_tensor(
        image: BMImageArray,
        tensor) -> Tensor
```

Parameters 2:

- image: BMImageArray

Input parameters. Image data that needs to be converted.

- tensor: Tensor

Output parameters. The converted Tensor.

Sample:

```
#include <sail/cvwrapper.h>

using namespace std;
int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    string image_path = "your_image_path";
    sail::Decoder decoder(image_path,false,dev_id);
    sail::BMImage img = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Tensor tensor(handle,{1920,1080},BM_FLOAT32,true,true);
    bmcv.bm_image_to_tensor(img,tensor);
    return 0;
}
```

4.14.3 tensor_to_bm_image

Convert Tensor to BMImage/BMImageArray.

Interface 1:

```
void tensor_to_bm_image(Tensor &tensor, BMImage &img, bool bgr2rgb=false,
↪std::string layout = std::string("nchw"));

void tensor_to_bm_image(Tensor &tensor, BMImage &img, bm_image_format_ext
↪format_);

BMImage tensor_to_bm_image(Tensor &tensor, bool bgr2rgb=false, std::string layout =
↪std::string("nchw"));

BMImage tensor_to_bm_image (Tensor &tensor, bm_image_format_ext format_);
```

Parameters 1:

- tensor: Tensor

Input parameters. The Tensor to be converted.

- img : BMImage

The converted image.

Returns 1:

- image : BMImage

Returns the converted image.

Interface 2:

```
template<std::size_t N> void  bm_image_to_tensor (BMImageArray<N> &imgs,
↪Tensor &tensor);
template<std::size_t N> Tensor bm_image_to_tensor (BMImageArray<N> &imgs);
```

Parameters 2:

- tensor: Tensor

Input parameters. The Tensor to be converted.

- img : BMImage | BMImageArray

Output parameters. Returns the converted image.

Returns 2:

- image : Tensor

Return the converted tensor.

Sample1:

```
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Tensor tensor = bmcv.bm_image_to_tensor(BMimg);
    sail::BMImage BMimg2 = bmcv.tensor_to_bm_image(tensor);
    return 0;
    }
```

Sample2:

```
#include <sail/cvwrapper.h>
#include <sail/tensor.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Tensor tensor = bmcv.bm_image_to_tensor(BMimg);
    sail::BMImage new_img();
    bmcv.tensor_to_bm_image(tensor,new_img);
    return 0;
    }
```

4.14.4 crop_and_resize

Crop then resize an image or an image array.

Interface:

```
int crop_and_resize(
    BMImage                 &input,
    BMImage                 &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);
```

(continues on next page)

(continued from previous page)

```
BMImage crop_and_resize(
  BMImage                &input,
  int                    crop_x0,
  int                    crop_y0,
  int                    crop_w,
  int                    crop_h,
  int                    resize_w,
  int                    resize_h,
  bmcv_resize_algorithm  resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int crop_and_resize(
   BMImageArray<N>          &input,
   BMImageArray<N>          &output,
   int                      crop_x0,
   int                      crop_y0,
   int                      crop_w,
   int                      crop_h,
   int                      resize_w,
   int                      resize_h,
   bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> crop_and_resize(
   BMImageArray<N>          &input,
   int                      crop_x0,
   int                      crop_y0,
   int                      crop_w,
   int                      crop_h,
   int                      resize_w,
   int                      resize_h,
   bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);
```

Parameters:

- input : BMImage | BMImageArray

The image or array of images to be processed.

- output : BMImage | BMImageArray

Processed image or image array.

- crop_x0 : int

The starting point of the cropping window on the x-axis.

- crop_y0 : int

The starting point of the cropping window on the y-axis.

- crop_w : int

The width of the crop window.

- crop_h : int

The height of the crop window.

- resize_w : int

The target width for image resize.

- resize_h : int

The target height for image resize.

- resize_alg : bmcv_resize_algorithm

Interpolation algorithm for image resize, default is bmcv_resize_algorithm.BMCV_INTER_NEAREST

Returns :

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.crop_and_resize(BMimg, 0, 0, BMimg.width(), BMimg.
↪height(), 640, 640);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    ssail::BMImage BMimg3;
    int ret = bmcv.crop_and_resize(BMimg, BMimg3,0, 0, BMimg.width(), BMimg.height(),
↪ 640, 640);
```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

4.14.5 crop

Crop the image.

Interface:

```
int crop(
    BMImage                 &input,
    BMImage                 &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h);


BMImage crop(
    BMImage                 &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h);

template<std::size_t N>
int crop(
    BMImageArray<N>         &input,
    BMImageArray<N>         &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h);

template<std::size_t N>
BMImageArray<N> crop(
    BMImageArray<N>         &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h);
```

Parameters:

- input : BMImage | BMImageArray

Input parameter. The image or array of images to be processed.

- output : BMImage | BMImageArray

Output parameter. Processed image or image array.

- crop_x0 : int

Input parameter. Start point x of the crop window.

- crop_y0 : int

Input parameter. Start point y of the crop window.

- crop_w : int

Input parameter. Width of the crop window.

- crop_h : int

Input parameter. Height of the crop window.

Returns:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.crop(BMimg,100,100,200,200);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.crop(BMimg, BMimg3,100,100,200,200);
    return 0;
}
```

4.14.6 resize

Resize the image.

Interface:

```
int resize(
  BMImage                  &input,
  BMImage                  &output,
  int                      resize_w,
  int                      resize_h,
  bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);

BMImage resize(
  BMImage                  &input,
  int                      resize_w,
  int                      resize_h,
  bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int resize(
   BMImageArray<N>          &input,
   BMImageArray<N>          &output,
   int                      resize_w,
   int                      resize_h,
   bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> resize(
   BMImageArray<N>          &input,
   int                      resize_w,
   int                      resize_h,
   bmcv_resize_algorithm    resize_alg = BMCV_INTER_NEAREST);
```

Parameters:

- input : BMImage | BMImageArray

The image or array of images to be processed.

- output : BMImage | BMImageArray

Processed image or image array.

- resize_w : int

The target width for image resize.

- resize_h : int

The target height for image resize.

- resize_alg : bmcv_resize_algorithm

Interpolation algorithm for image resize, default is bmcv_resize_algorithm.BMCV_INTER_NEAREST

Returns:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.resize(BMimg,640,640);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.resize(BMimg, BMimg3,640,640);
    return 0;
}
```

4.14.7 vpp_crop_and_resize

Use VPP hardware to accelerate image cropping and resizing.

Interface:

```
int vpp_crop_and_resize(
    BMImage                  &input,
    BMImage                  &output,
    int                      crop_x0,
    int                      crop_y0,
```

(continues on next page)

(continued from previous page)

```
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

BMImage vpp_crop_and_resize(
    BMImage                 &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int vpp_crop_and_resize(
    BMImageArray<N>         &input,
    BMImageArray<N>         &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> vpp_crop_and_resize(
    BMImageArray<N>         &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm   resize_alg = BMCV_INTER_NEAREST);
```

Parameters:

- input : BMImage | BMImageArray

The image or array of images to be processed.

- output : BMImage | BMImageArray

Processed image or image array.

- crop_x0 : int

The starting point of the cropping window on the x-axis.

- crop_y0 : int

The starting point of the cropping window on the y-axis.

- crop_w : int

The width of the crop window.

- crop_h : int

The height of the crop window.

- resize_w : int

The target width for image resize.

- resize_h : int

The target height for image resize.

- resize_alg : bmcv_resize_algorithm

Interpolation algorithm for image resize, default is bmcv_resize_algorithm.BMCV_INTER_NEAREST

Returns:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.vpp_crop_and_resize(BMimg,100,100,300,300,300,300);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
```

(continues on next page)

(continued from previous page)

```
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.vpp_crop_and_resize(BMimg, BMimg3,100,100,300,300,300,300);
    return 0;
}
```

4.14.8 vpp_crop_and_resize_padding

Use VPP hardware to accelerate image cropping and resizing, and padding to the specified size.

Interface:

```
int vpp_crop_and_resize_padding(
    BMImage                 &input,
    BMImage                 &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

BMImage vpp_crop_and_resize_padding(
    BMImage                 &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int vpp_crop_and_resize_padding(
    BMImageArray<N>           &input,
    BMImageArray<N>           &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
```

(continues on next page)

(continued from previous page)

```
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> vpp_crop_and_resize_padding(
    BMImageArray<N>             &input,
    int                         crop_x0,
    int                         crop_y0,
    int                         crop_w,
    int                         crop_h,
    int                         resize_w,
    int                         resize_h,
    PaddingAtrr                 &padding_in,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);
```

Parameters:

- input : BMImage | BMImageArray

The image or array of images to be processed.

- output : BMImage | BMImageArray

Processed image or image array.

- crop_x0 : int

The starting point of the cropping window on the x-axis.

- crop_y0 : int

The starting point of the cropping window on the y-axis.

- crop_w : int

The width of the crop window.

- crop_h : int

The height of the crop window.

- resize_w : int

The target width for image resize.

- resize_h : int

The target height for image resize.

- padding : PaddingAtrr

padding configuration information.

- resize_alg : bmcv_resize_algorithm

Interpolation algorithm for image resize, default is bmcv_resize_algorithm.BMCV_INTER_NEAREST

Returns:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    sail::BMImage BMimg4 = bmcv.vpp_crop_and_resize_padding(BMimg, 0, 0, BMimg.
↪width(), BMimg.height(), 640, 640, paddingatt);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    int ret = bmcv.vpp_crop_and_resize_padding(BMimg,BMimg3, 0, 0, BMimg.width(),F
↪BMimg.height(), 640, 640, paddingatt);
```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

4.14.9 vpp_crop

Use VPP hardware to accelerate image cropping.

Interface:

```
int vpp_crop(
    BMImage                 &input,
    BMImage                 &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h);

BMImage vpp_crop(
    BMImage                 &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h);

template<std::size_t N>
int vpp_crop(
    BMImageArray<N>         &input,
    BMImageArray<N>         &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h);

template<std::size_t N>
BMImageArray<N> vpp_crop(
    BMImageArray<N>         &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h);
```

Parameters:

- input : BMImage | BMImageArray

The image or array of images to be processed.

- output : BMImage | BMImageArray

Processed image or image array.

- crop_x0 : int

The starting point of the cropping window on the x-axis.

- crop_y0 : int

The starting point of the cropping window on the y-axis.

- crop_w : int

The width of the crop window.

- crop_h : int

The height of the crop window.

返回值说明:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.vpp_crop(BMimg,100,100,200,200);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.vpp_crop(BMimg, BMimg3,100,100,200,200);
    return 0;
}
```

4.14.10 vpp_resize

Use VPP hardware to accelerate image resize and use nearest neighbor interpolation algorithm.

接口形式1:

```
int vpp_resize(
    BMImage                 &input,
    BMImage                 &output,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

BMImage vpp_resize(
    BMImage                 &input,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
int vpp_resize(
    BMImageArray<N>          &input,
    BMImageArray<N>          &output,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> vpp_resize(
    BMImageArray<N>          &input,
    int                     resize_w,
    int                     resize_h,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);
```

Parameters:

- input : BMImage | BMImageArray

The image or array of images to be processed.

- output : BMImage | BMImageArray

Processed image or image array.

- resize_w : int

The target width for image resize.

- resize_h : int

The target height for image resize.

- resize_alg : bmcv_resize_algorithm

Interpolation algorithm for image resize, default is bmcv_resize_algorithm.BMCV_INTER_NEAREST

Returns:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3 = bmcv.vpp_resize(BMimg,100,100,200,200);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    int ret = bmcv.vpp_resize(BMimg, BMimg3,100,100,200,200);
    return 0;
}
```

4.14.11 vpp_resize_padding

Use VPP hardware to accelerate image resizing and padding.

Interface:

```
int vpp_resize_padding(
    BMImage                  &input,
    BMImage                  &output,
    int                   resize_w,
    int                   resize_h,
```

(continues on next page)

(continued from previous page)

```
    PaddingAtrr                &padding_in,
    bmcv_resize_algorithm      resize_alg = BMCV_INTER_NEAREST);

BMImage vpp_resize_padding(
  BMImage                  &input,
  int                      resize_w,
  int                      resize_h,
  PaddingAtrr              &padding_in,
   bmcv_resize_algorithm      resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
   int vpp_resize_padding(
   BMImageArray<N>           &input,
   BMImageArray<N>           &output,
   int                       resize_w,
   int                       resize_h,
   PaddingAtrr               &padding_in,
   bmcv_resize_algorithm     resize_alg = BMCV_INTER_NEAREST);

template<std::size_t N>
BMImageArray<N> vpp_resize_padding(
   BMImageArray<N>           &input,
   int                       resize_w,
   int                       resize_h,
   PaddingAtrr               &padding_in,
   bmcv_resize_algorithm     resize_alg = BMCV_INTER_NEAREST);
```

Parameters:

- input : BMImage | BMImageArray

The image or array of images to be processed.

- resize_w : int

The target width for image resize.

- resize_h : int

The target height for image resize.

- padding : PaddingAtrr

The configuration information of padding.

Returns:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

- resize_alg : bmcv_resize_algorithm

Interpolation algorithm for image resize, default is bmcv_resize_algorithm.BMCV_INTER_NEAREST

Sample1:

```cpp
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    sail::BMImage BMimg4 = bmcv.vpp_resize_padding(BMimg, 0, 0, 640, 640, paddingatt);
    return 0;
}
```

Sample2:

```cpp
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    int ret = bmcv.vpp_resize_padding(BMimg,BMimg3, 640, 640, paddingatt);
    return 0;
}
```

4.14.12 warp

Perform an affine transformation on the image.

Interface:

```
int warp(
  BMImage                       &input,
  BMImage                       &output,
  const std::pair<
    std::tuple<float, float, float>,
    std::tuple<float, float, float>> &matrix,
  int                           use_bilinear = 0,
  bool                          similar_to_opencv = false);

BMImage warp(
  BMImage                       &input,
  const std::pair<
    std::tuple<float, float, float>,
    std::tuple<float, float, float>> &matrix,
  int                           use_bilinear = 0,
  bool                          similar_to_opencv = false);

template<std::size_t N>
int warp(
    BMImageArray<N>                     &input,
    BMImageArray<N>                     &output,
    const std::array<
        std::pair<
        std::tuple<float, float, float>,
        std::tuple<float, float, float>>, N> &matrix,
    int                                 use_bilinear = 0,
    bool                                similar_to_opencv = false);

template<std::size_t N>
BMImageArray<N> warp(
    BMImageArray<N>                     &input,
    const std::array<
        std::pair<
        std::tuple<float, float, float>,
        std::tuple<float, float, float>>, N> &matrix,
    int                                 use_bilinear = 0,
    bool                                similar_to_opencv = false);
```

Parameters:

- input : BMImage | BMImageArray

The image or array of images to be processed.

- output : BMImage | BMImageArray

Processed image or image array.

- **matrix: std::pair**< std::tuple<float, float, float>, std::tuple<float, float, float> >

2x3 affine transformation matrix.

- use_bilinear: int

Whether to use bilinear interpolation, default to 0 using nearest neighbor interpolation, 1 being bilinear interpolation

- similar_to_opencv: bool

Whether to use the interface aligning the affine transformation interface of OpenCV

Returns:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
using AffineMatrix = std::pair<
    std::tuple<float, float, float>,
    std::tuple<float, float, float>>;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    AffineMatrix rotated_matrix = std::make_pair(
        std::make_tuple(0.9996914396, -0.02484, 0.0f),
        std::make_tuple(0.02484, 0.9996914396, 0.0f)
    );
    sail::BMImage BMimg6 = bmcv.warp(BMimg, rotated_matrix);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
using AffineMatrix = std::pair<
    std::tuple<float, float, float>,
    std::tuple<float, float, float>>;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
```

(continues on next page)

(continued from previous page)

```
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    AffineMatrix rotated_matrix = std::make_pair(
        std::make_tuple(0.9996914396, -0.02484, 0.0f),
        std::make_tuple(0.02484, 0.9996914396, 0.0f)
    );
    sail::BMImage BMimg6;
    int ret= bmcv.warp(BMimg,BMimg6, rotated_matrix);
    return 0;
}
```

4.14.13 convert_to

Perform a linear transformation on the image.

Interface:

```
int convert_to(
  BMImage                  &input,
  BMImage                  &output,
  const std::tuple<
    std::pair<float, float>,
    std::pair<float, float>,
    std::pair<float, float>>   &alpha_beta);

BMImage convert_to(
  BMImage                  &input,
  const std::tuple<
    std::pair<float, float>,
    std::pair<float, float>,
    std::pair<float, float>>   &alpha_beta);

template<std::size_t N>
int convert_to(
    BMImageArray<N>            &input,
    BMImageArray<N>            &output,
    const std::tuple<
        std::pair<float, float>,
        std::pair<float, float>,
        std::pair<float, float>>   &alpha_beta);

template<std::size_t N>
BMImageArray<N> convert_to(
    BMImageArray<N>            &input,
    const std::tuple<
        std::pair<float, float>,
        std::pair<float, float>,
        std::pair<float, float>>   &alpha_beta);
```

Parameters:

- input : BMImage | BMImageArray

The image or array of images to be processed.

- **alpha_beta: std::tuple**< std::pair<float, float>, std::pair<float, float>, std::pair<float, float> >

The coefficients of the linear transformation of the three channels ((a0, b0), (a1, b1), (a2, b2)).

- output : BMImage | BMImageArray

Output parameters. Processed image or image array.

Returns:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the processed image or image array.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::tuple<std::pair<float, float>, std::pair<float, float>, std::pair<float, float>> alpha_
↪beta =
        std::make_tuple(std::make_pair(1.0 / 255, 0), std::make_pair(1.0 / 255, 0), std::make_
↪pair(1.0 / 255, 0));
    sail::BMImage BMimg5 = bmcv.convert_to(BMimg, alpha_beta);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
```

(continues on next page)

(continued from previous page)

```
    sail::Bmcv bmcv(handle);
    std::tuple<std::pair<float, float>, std::pair<float, float>, std::pair<float, float>> alpha_
↪beta =
        std::make_tuple(std::make_pair(1.0 / 255, 0), std::make_pair(1.0 / 255, 0), std::make_
↪pair(1.0 / 255, 0));
    sail::BMImage BMimg5;
    int ret = bmcv.convert_to(BMimg,BMimg5,alpha_beta);
    return 0;
}
```

4.14.14 yuv2bgr

Convert the format of the image from YUV to BGR.

Interface:

```
int yuv2bgr(
    BMImage                 &input,
    BMImage                 &output);

BMImage yuv2bgr(BMImage  &input);
```

Parameters:

- input : BMImage | BMImageArray

The image to be converted.

Returns:

- ret: int

Returns 0 for success, others for failure.

- output : BMImage | BMImageArray

Returns the converted image.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg5 = bmcv.yuv2bgr(BMimg);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg5;
    int ret = bmcv.yuv2bgr(BMimg,BMimg5);
    return 0;
}
```

4.14.15 rectangle

Draw a rectangular box on the image.

Interface:

```
int rectangle(
    BMImage                     &image,
    int                         x0,
    int                         y0,
    int                         w,
    int                         h,
    const std::tuple<int, int, int> &color,
    int                         thickness=1);

int rectangle(
    const bm_image              &image,
    int                         x0,
    int                         y0,
    int                         w,
    int                         h,
    const std::tuple<int, int, int> &color, // BGR
    int                         thickness=1);
```

Parameters:

- image : BMImage | bm_image

The image of the rectangle to be drawn.

- x0 : int

The starting point of the rectangle on the x-axis.

- y0 : int

The starting point of the rectangular box on the y-axis.

- w : int

The width of the rectangular box.

- h : int

The height of the rectangular box.

- color : tuple

The color of the rectangle.

- thickness : int

The thickness of the rectangular box lines.

Returns:

Returns 0 if the frame is successful, otherwise returns a non-zero value.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.rectangle(BMimg, 20, 20, 600, 600, std::make_tuple(0, 0, 255), 2);
    return 0;
}
```

4.14.16 fillRectangle

Fill a rectangular box on the image.

Interface:

```
int fillRectangle(
    BMImage                     &image,
    int                         x0,
    int                         y0,
    int                         w,
    int                         h,
    const std::tuple<int, int, int> &color);

int fillRectangle(
    const bm_image              &image,
    int                         x0,
    int                         y0,
    int                         w,
```

(continues on next page)

(continued from previous page)

```
int                          h,
const std::tuple<int, int, int> &color);
```

Parameters:

- image : BMImage | bm_image

The image of the rectangle to be drawn.

- x0 : int

The starting point of the rectangle on the x-axis.

- y0 : int

The starting point of the rectangular box on the y-axis.

- w : int

The width of the rectangular box.

- h : int

The height of the rectangular box.

- color : tuple

The color of the rectangle.

Returns:

Returns 0 if the frame is successful, otherwise returns a non-zero value.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.fillRectangle(BMimg, 20, 20, 600, 600, std::make_tuple(0, 0, 255));
    return 0;
}
```


4.14.17 imwrite

Save the image in a specific file.

Interface:

```
int imwrite(
  const std::string &filename,
  BMImage           &image);

int imwrite(
  const std::string &filename,
  BMImage           &image,
  const std::vector<int> &params = {});

int imwrite(
  const std::string &filename,
  const bm_image     &image);
```

Parameters:

- file_name : string

The name of the file.

- output : BMImage | bm_image

The image needs to be saved.

- params : vector<int>

Format parameters for saving the image. Must ensure the validity of the parameters.

Returns:

- process_status : int

Returns 0 if the save is successful, otherwise returns a non-zero value.

Sample:

```
#include <sail/cvwrapper.h>
#include "opencv2/opencv.hpp"
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.imwrite("new_0.jpg", BMimg);
    std::vector<int> params = { cv::IMWRITE_JPEG_QUALITY, 95 };
    ret = bmcv.imwrite("new_1.jpg", BMimg, params);
```

(continues on next page)

(continued from previous page)

```
    return 0;
}
```

4.14.18 imread

Read and decode one image files and supports hard decoding only for JPEG baseline format. For other formats, such as PNG and BMP, soft decoding is used. The returned BMImage for JPEG baseline images keeps YUV color space, and the pixel format depends on the sampling information in the file like YUV420. The returned BMImage for other formats will maintain the corresponding color space of their input.

Interface:

```
BMImage imread(const std::string &filename);
```

Parameters:

- filename : string

Name of file to be read.

Returns:

- output : sail.BMImage

The decoded image.

Sample:

```
#include <sail/cvwrapper.h>
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    std::string filename = "your_image_path";
    sail::BMImage BMimg = bmcv.imread(filename);
    return 0;
}
```

4.14.19 get_handle

Get the device handle Handle in Bmcv.

Interface:

```
Handle get_handle();
```

Returns:

- handle: Handle

The device handle Handle in Bmcv.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::Handle handle1 = bmcv.get_handle();
    return 0;
}
```

4.14.20 crop_and_resize_padding

Crop and resize the image, then padding it.

Interface:

```
int vpp_crop_and_resize_padding(
    BMImage                 &input,
    BMImage                 &output,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);

BMImage vpp_crop_and_resize_padding(
    BMImage                 &input,
    int                     crop_x0,
    int                     crop_y0,
    int                     crop_w,
    int                     crop_h,
    int                     resize_w,
    int                     resize_h,
    PaddingAtrr             &padding_in,
    bmcv_resize_algorithm       resize_alg = BMCV_INTER_NEAREST);
```

Parameters:

- input : BMImage

The image to be processed.

- output : BMImage

The processed image.

- crop_x0 : int

The starting point of the cropping window on the x-axis.

- crop_y0 : int

The starting point of the cropping window on the y-axis.

- crop_w : int

The width of the crop window.

- crop_h : int

The height of the crop window.

- resize_w : int

The target width for image resize.

- resize_h : int

The target height for image resize.

- padding : PaddingAtrr

The configuration information of padding.

- resize_alg : bmcv_resize_algorithm

The interpolation algorithm used by resize.

Returns:

- process_status : int

Returns 0 if the save is successful, otherwise returns a non-zero value.

- output : BMImage

Return the processed image.

Sample1:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
```

(continues on next page)

(continued from previous page)

```
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    sail::BMImage BMimg4 = bmcv.crop_and_resize_padding(BMimg, 0, 0, BMimg.width(),
↪ BMimg.height(), 640, 640, paddingatt);
    return 0;
}
```

Sample2:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg3;
    sail::PaddingAtrr paddingatt;
    paddingatt.set_stx(0);
    paddingatt.set_sty(0);
    paddingatt.set_w(640);
    paddingatt.set_h(640);
    paddingatt.set_r(114);
    paddingatt.set_g(114);
    paddingatt.set_b(114);
    bm_image bm_img = bmcv.crop_and_resize_padding(BMimg.data(), 0, 0, BMimg.
↪width(), BMimg.height(), 640, 640, paddingatt);
    return 0;
}
```

4.14.21 convert_format

Convert the image format to the format in the output and copy it to the output.

Interface 1:

```
int convert_format(
    BMImage         &input,
    BMImage         &output
);
```

Parameters 1:

- input : BMImage

Input parameters. The image to be converted.

- output : BMImage

Output parameters. Convert the image in input to the image format of output and copy it to output.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg4;
    int ret = bmcv.convert_format(BMimg,BMimg4);
    return 0;
}
```

Interface 2:

Convert an image to the target format.

```
BMImage convert_format(
    BMImage          &input,
    bm_image_format_ext image_format = FORMAT_BGR_PLANAR
);
```

Parameters 2:

- input : BMImage

The image to be converted.

- image_format : bm_image_format_ext

The target format for conversion.

Returns 2:

- output : BMImage

Returns the converted image.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
```

(continues on next page)

(continued from previous page)

```
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg4 = bmcv.convert_format(BMimg);
    return 0;
}
```

4.14.22 vpp_convert_format

Use VPP hardware to accelerate image format conversion.

Interface 1:

```
int vpp_convert_format(
    BMImage         &input,
    BMImage         &output
);
```

Parameters 1:

- input : BMImage

Input parameters. The image to be converted.

- output : BMImage

Output parameters. Convert the image in input to the image format of output and copy it to output.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg4;
    int ret = bmcv.vpp_convert_format(BMimg,BMimg4);
    return 0;
}
```

Interface 2:

Convert an image to the target format.

```
BMImage vpp_convert_format(
    BMImage         &input,
```

(continues on next page)

(continued from previous page)

```
    bm_image_format_ext image_format = FORMAT_BGR_PLANAR
);
```

Parameters 2:

- input : BMImage

The image to be converted.

- image_format : bm_image_format_ext

The target format for conversion.

Returns 2:

- output : BMImage

Returns the converted image.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage BMimg4 = bmcv.vpp_convert_format(BMimg);
    return 0;
}
```

4.14.23 putText

Add text to the image.

Supported pixel format for input BMImage: FORMAT_GRAY, FORMAT_YUV420P, FORMAT_YUV422P, FORMAT_YUV444P, FORMAT_NV12, FORMAT_NV21, FORMAT_NV16, FORMAT_NV61

Interface:

```
int putText(
    const BMImage                 &image,
    const std::string             &text,
    int                           x,
    int                           y,
    const std::tuple<int, int, int> &color, // BGR
    float                         fontScale,
```

(continues on next page)

(continued from previous page)

```
    int                         thickness=1
);

int putText(
    const bm_image                  &image,
    const std::string               &text,
    int                         x,
    int                         y,
    const std::tuple<int, int, int> &color, // BGR
    float                       fontScale,
    int                         thickness=1
);
```

Parameters:

- input : BMImage | bm_image

The image to be processed.

- text: string

Text that needs to be added.

- x: int

The starting point for adding the text on the x-axis.

- y: int

The starting point for adding the text on the y-axis.

- color : tuple

The color of the font.

- fontScale: int

The size of the font.

- thickness : int

The thickness of the font.

Returns:

- process_status : int

Returns 0 if processing is successful, otherwise returns a non-zero value.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
```

(continues on next page)

(continued from previous page)

```
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage bgr_img = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage yuv_img = bmcv.convert_format(bgr_img, FORMAT_YUV420P)
    int ret = bmcv.putText(yuv_img, "some text" , 20, 20, std::make_tuple(0, 0, 255), 1.4,F
↪2);

    return 0;
}
```

4.14.24 image_add_weighted

Add two images with different weights.

Interface 1:

```
int image_add_weighted(
    BMImage         &input1,
    float           alpha,
    BMImage         &input2,
    float           beta,
    float           gamma,
    BMImage         &output
);
```

Parameters 1:

- input0 : BMImage

Input parameters. The image 0 to be processed.

- alpha : float

Input parameters. The weight alpha of the two images added together.

- input1 : BMImage

Input parameters. The image 1 to be processed.

- beta : float

Input parameters. The weight beta of the two images added together.

- gamma : float

Input parameters. The weight gamma of the two images added together.

- output: BMImage

Output parameters. The added image output = input1 * alpha + input2 * beta + gamma

Sample:

```cpp
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name1 = "your_image_path1";
    std::string image_name2 = "your_image_path2";
    sail::Decoder decoder1(image_name1, true, tpu_id);
    sail::Decoder decoder2(image_name2, true, tpu_id);
    sail::BMImage BMimg1 = decoder1.read(handle);
    sail::BMImage BMimg2 = decoder2.read(handle);
    sail::Bmcv bmcv(handle);
    float alpha=0.2,beta=0.5,gamma=0.8;
    int ret = bmcv.image_add_weighted(BMimg1,alpha,BMimg2,beta,gamma,BMimg2);
    return 0;
}
```

Interface 2:

```cpp
BMImage image_add_weighted(
    BMImage         &input1,
    float           alpha,
    BMImage         &input2,
    float           beta,
    float           gamma
);
```

Parameters 2:

- input0 : BMImage

Input parameters. The image 0 to be processed.

- alpha : float

Input parameters. The weight alpha of the two images added together.

- input1 : BMImage

Input parameters. The image 1 to be processed.

- beta : float

Input parameters. The weight beta of the two images added together.

- gamma : float

Input parameters. The weight gamma of the two images added together.

Returns 2:

- output: BMImage

Return the added image output = input1 * alpha + input2 * beta + gamma

Sample:

```
#include <sail/cvwrapper.h>
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name1 = "your_image_path1";
    std::string image_name2 = "your_image_path2";
    sail::Decoder decoder1(image_name1, true, tpu_id);
    sail::Decoder decoder2(image_name2, true, tpu_id);
    sail::BMImage BMimg1 = decoder1.read(handle);
    sail::BMImage BMimg2 = decoder2.read(handle);
    sail::Bmcv bmcv(handle);
    float alpha=0.2,beta=0.5,gamma=0.8;
    sail::BMImage img= bmcv.image_add_weighted(BMimg1,alpha,BMimg2,beta,gamma);
    return 0;
}
```

4.14.25 image_copy_to

Copy data between images

Interface:

```
int image_copy_to(BMImage &input, BMImage &output, int start_x = 0, int start_y = 0);

template<std::size_t N>
int image_copy_to(BMImageArray<N> &input, BMImageArray<N> &output, int start_
↪x = 0, int start_y = 0);
```

Parameters:

- input: BMImage|BMImageArray

Input parameter. The BMImage or BMImageArray to be copied.

- output: BMImage|BMImageArray

Output parameter. Copied BMImage or BMImageArray

- start_x: int

Input parameter. Copy to the starting point of the target image.

- start_y: int

Input parameter. Copy to the starting point of the target image.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
```

(continues on next page)

(continued from previous page)

```
    std::string image_name1 = "your_image_path1";
    std::string image_name2 = "your_image_path2";
    sail::Decoder decoder1(image_name1, true, tpu_id);
    sail::Decoder decoder2(image_name2, true, tpu_id);
    sail::BMImage BMimg1 = decoder1.read(handle);
    sail::BMImage BMimg2 = decoder2.read(handle);
    sail::Bmcv bmcv(handle);
    bmcv.image_copy_to(BMimg1,BMimg2,0,0);
    return 0;
}
```

4.14.26 image_copy_to_padding

Copy and padding the image data between input and output.

Interface:

```
int image_copy_to_padding(BMImage &input,
                          BMImage &output,
                          unsigned int padding_r,
                          unsigned int padding_g,
                          unsigned int padding_b,
                          int start_x = 0,
                          int start_y = 0);

template<std::size_t N>
int image_copy_to_padding(BMImageArray<N> &input,
                          BMImageArray<N> &output,
                          unsigned int padding_r,
                          unsigned int padding_g,
                          unsigned int padding_b,
                          int start_x = 0,
                          int start_y = 0);
```

Parameters:

- input: BMImage|BMImageArray

Input parameter. The BMImage or BMImageArray to be copied.

- output: BMImage|BMImageArray

Output parameter. Copied BMImage or BMImageArray.

- padding_r: int

Input parameter. The padding value of the R channel.

- padding_g: int

Input parameter. The padding value of the G channel.

- padding_b: int

Input parameter. The padding value of the B channel.

- start_x: int

Input parameter. Copy to the starting point on x-axis of the target image.

- start_y: int

Input parameter. Copy to the starting point on y-axis of the target image.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name1 = "your_image_path1";
    std::string image_name2 = "your_image_path2";
    sail::Decoder decoder1(image_name1, true, tpu_id);
    sail::Decoder decoder2(image_name2, true, tpu_id);
    sail::BMImage BMimg1 = decoder1.read(handle);
    sail::BMImage BMimg2 = decoder2.read(handle);
    sail::Bmcv bmcv(handle);
    bmcv.image_copy_to_padding(BMimg1,BMimg2,128,128,128,0,0);
    return 0;
}
```

4.14.27 nms

Using Tensor Computing Processor for NMS

Note: For details about whether this operator in current SDK supports BM1688, check the page "BMCV API" in 《Multimedia User Guide》.

Interface:

```
nms_proposal_t* nms(
    face_rect_t *input_proposal,
    int proposal_size,
    float threshold);
```

Parameters:

- input_proposal: face_rect_t

Data starting address.

- proposal_size: int

The size of the detection frame data to be processed.

- threshold: float

Threshold of nms.

Returns:

- result: nms_proposal_t

Returns the detection frame array after NMS.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    face_rect_t *input_proposal;
    int proposal_size = 100;
    float threshold = 0.5;
    nms_proposal_t* result = bmcv.nms(input_proposal, proposal_size, threshold);
    return 0;
}
```

4.14.28 drawPoint

Draw points on the image.

Interface:

```
int drawPoint(
    const BMImage &image,
    std::pair<int,int> center,
    std::tuple<unsigned char, unsigned char, unsigned char> color,   // BGR
    int radius);

int drawPoint(
    const bm_image  &image,
    std::pair<int,int> center,
    std::tuple<unsigned char, unsigned char, unsigned char> color,  // BGR
    int radius);
```

Parameters:

- image: BMImage

Input image. Draw points directly on the BMImage as output.

- center: std::pair<int,int>

The center coordinates of the point.

- color: std::tuple<unsigned char, unsigned char, unsigned char>

The color of the point.

- radius: int

The radius of the point.

Returns

If the point is drawn successfully, 0 is returned, otherwise a non-zero value is returned.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path1";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    int ret = bmcv.drawPoint(BMimg, std::pair(320, 320), std::make_tuple(0, 255, 255), 2);
    return 0;
}
```

4.14.29 warp_perspective

Performs perspective transformation on the image.

Interface:

```
BMImage warp_perspective(
    BMImage                 &input,
    const std::tuple<
    std::pair<int,int>,
    std::pair<int,int>,
    std::pair<int,int>,
    std::pair<int,int>>     &coordinate,
    int                     output_width,
    int                     output_height,
    bm_image_format_ext     format = FORMAT_BGR_PLANAR,
    bm_image_data_format_ext dtype = DATA_TYPE_EXT_1N_BYTE,
    int                     use_bilinear = 0);
```

Parameters:

- input: BMImage

The image to be processed.

- **coordinate: std::tuple**< std::pair<int,int>, std::pair<int,int>, std::pair<int,int>, std::pair<int,int> >

The original coordinates of the four vertices of the transformed area.

Such as, ((left_top.x, left_top.y), (right_top.x, right_top.y), (left_bottom.x, left_bottom.y), (right_bottom.x, right_bottom.y))

- output_width: int

The width of the output image.

- output_height: int

The height of the output image.

- format: bm_image_format_ext

The format of the output image.

- dtype: bm_image_data_format_ext

The data type of the output image.

- use_bilinear: int

Whether to use bilinear interpolation.

Returns:

- output: BMImage

Output the transformed image.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;

int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::tuple<
        std::pair<int, int>,
        std::pair<int, int>,
        std::pair<int, int>,
        std::pair<int, int>
    > coordinate = std::make_tuple(
        std::make_pair(100, 100),
        std::make_pair(200, 100),
        std::make_pair(100, 200),
        std::make_pair(200, 200)
    )
    int output_width = 300;
```

(continues on next page)

(continued from previous page)

```
    int output_height = 300;
    bm_image_format_ext format = FORMAT_BGR_PLANAR;
    bm_image_data_format_ext dtype = DATA_TYPE_EXT_1N_BYTE;
    int use_bilinear = 1;

    sail::BMImage output = bmcv.warp_perspective(BMimg,coordinate,output_width,
↪output_height,format,dtype,use_bilinear
    );

    return 0;
}
```

4.14.30 get_bm_data_type

Convert ImgDtype to Dtype

Interface:

```
bm_data_type_t get_bm_data_type(bm_image_data_format_ext fmt);
```

Parameters:

- fmt: bm_image_data_format_ext

The type to be converted.

Returns:

- ret: bm_data_type_t

The converted type.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    bm_data_type_t ret = bmcv.get_bm_data_type(bm_image_data_format_
↪ext::DATA_TYPE_EXT_FLOAT32);
    return 0;
}
```

4.14.31 get_bm_image_data_format

Convert Dtype to ImgDtype.

Interface:

```
bm_image_data_format_ext get_bm_image_data_format(bm_data_type_t dtype);
```

Parameters:

- dtype: bm_data_type_t

The Dtype that needs to be converted.

Returns:

- ret: bm_image_data_format_ext

Returns the converted type.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    bm_image_data_format_ext ret = bmcv.get_bm_image_data_format(bm_data_type_
↪t::BM_FLOAT32);
    return 0;
}
```

4.14.32 imdecode

Load the image from memory into BMImage.

Interface:

```
BMImage imdecode(const void* data_ptr, size_t data_size);
```

Parameters:

- data_ptr: void*

The data starting address.

- data_size: bytes

The data length.

Returns:

- ret: BMImage

Returns the decoded image.

Sample:

```
#include <sail/cvwrapper.h>
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    std::ifstream image_file(image_name, std::ios::binary);
    if (!image_file) {
        std::cout << "Error opening image file." << std::endl;
        return -1;
    }
    std::vector<char> image_data_bytes(
        (std::istreambuf_iterator<char>(image_file)),
        (std::istreambuf_iterator<char>())
    );
    image_file.close();
    sail::Bmcv bmcv(handle);
    sail::BMImage src_img = bmcv.imdecode(image_data_bytes.data(), image_data_bytes.
↪size());
    return 0;
}
```

4.14.33 imencode

Encode an BMimage and return the encoded data.

Interface1:

```
bool Bmcv::imencode(std::string& ext, bm_image &img, std::vector<u_char>& buf)
```

Interface2:

```
bool Bmcv::imencode(std::string& ext, BMImage &img, std::vector<u_char>& buf)
```

Parameters:

- ext: string

Input parameter. Image encoding format, supported formats include ".jpg", ".png", etc.

- image: bm_image/BMImage

Input parameter. Input bm_image/BMImage, only FORMAT_BGR_PACKED, DATA_TYPE_EXT_1N_BYTE pictures are supported.

- buf: std::vector<u_char>

Output parameter. Data that is encoded and placed in system memory.

返回值说明:

- ret: bool

Returns 0 if encoding is successful and 1 if encoding fails.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::vector<u_char> encoded_data;
    std::string ext = ".jpg";
    bool success = bmcv.imencode(ext, BMimg, encoded_data);
    //bool success = bmcv.imencode(ext, BMimg.data(), encoded_data);  接口形式1:bm_
↪image
    return 0;
}
```

4.14.34 fft

Implement the Fast Fourier Transform of Tensor.

Note: For details about whether this operator in current SDK supports BM1688, check the page "BMCV API" in 《Multimedia User Guide》.

Interface:

```
std::vector<Tensor> fft(bool forward, Tensor &input_real);

std::vector<Tensor> fft(bool forward, Tensor &input_real, Tensor &input_imag);
```

Parameters:

- forward: bool

Whether to perform forward migration.

- input_real: Tensor

The real part of the input.

- input_imag: Tensor

The imaginary part of the input.

Returns:

- ret: std::vector<Tensor>

Returns the real and imaginary parts of the output.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    std::vector<int> shape = {512, 512};
    sail::Tensor input_real(shape);
    bool forward = true;
    //std::vector<sail::Tensor> result_real = bmcv.fft(forward, input_real);
    sail::Tensor input_imag(shape);
    std::vector<sail::Tensor> result_complex  = bmcv.fft(forward, input_real,input_imag);
    return 0;
}
```

4.14.35 convert_yuv420p_to_gray

Convert pictures in YUV420P format to grayscale images.

Interface 1:

```
int convert_yuv420p_to_gray(BMImage& input, BMImage& output);
```

Parameters 1:

- input : BMImage

Input parameters. The image to be converted.

- output : BMImage

Output parameters. Converted image.

Interface 2:

Convert pictures in YUV420P format to grayscale images.

```
int convert_yuv420p_to_gray_(bm_image& input, bm_image& output);
```

Parameters 2:

- input : bm_image

The image to be converted.

- output : bm_image

The converted image.

Sample:

```cpp
#include <sail/cvwrapper.h>
using namespace std;
int main() {
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path1";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    sail::BMImage img;
    int ret = bmcv.convert_yuv420p_to_gray(BMimg, img);
    return 0;
}
```

4.14.36 polylines

Draw one or more line segments on an image, so that the function of drawing polygons can be realized, and the color and width of the line can be specified.

Interface:

```cpp
int polylines(
    BMImage &img,
    std::vector<std::vector<std::pair<int,int>>> &pts,
    bool isClosed,
    std::tuple<unsigned char, unsigned char, unsigned char> color,
    int thickness = 1,
    int shift = 0);
```

Parameters:

- img : BMImage

Input BMImage.

- pts : std::vector<std::vector<std::pair<int,int>>>

The starting point and end point coordinates of the line segment, multiple coordinate points can be entered. The upper left corner of the image is the origin, extending to the right in the x direction and extending down in the y direction.

- isClosed : bool

Whether the graph is closed.

- color : std::tuple<unsigned char, unsigned char, unsigned char>

The color of the line is the value of the three RGB channels.

- thickness : int

The width of the lines is recommended to be even for YUV format images.

- shift : int

Polygon scaling multiple. Default is not scaling. The scaling factor is(1/2)^shift。

Returns:

- ret: int

returns 0 if success.

Sample:

```
#include <sail/cvwrapper.h>
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::vector<std::vector<std::pair<int, int>>> pts = {
        {{100, 100}, {150, 100}, {150, 150}, {100, 150}},
        {{200, 200}, {250, 200}, {250, 250}, {200, 250}}

    };
    bool isClosed = true;
    int thickness = 2;
    std::tuple<unsigned char, unsigned char, unsigned char> color = std::make_tuple(255, 0,
↪ 0);
    int shift = 0;
    int result = bmcv.polylines(BMimg, pts, isClosed, color, thickness, shift);
    if (result == 0) {
        std::cout << "Polylines drawn successfully." << std::endl;
    } else {
        std::cout << "Failed to draw polylines." << std::endl;
    }
    return 0;
}
```

4.14.37 mosaic

Print one or more mosaics on an image.

Interface:

```
int mosaic(
  int mosaic_num,
  BMImage &img,
  vector<vector<int>> rects,
  int is_expand);
```

Parameters:

- mosaic_num : int

Number of mosaics, length of list in rects.

- img : BMImage

Input BMImage.

- rects : vector<vector<int>>

Multiple Mosaic positions, the parameters in each element in the list are [Mosaic at X-axis start point, Mosaic at Y-axis start point, Mosaic width, Mosaic height].

- is_expand : int

Whether to expand the column. A value of 0 means that the column is not expanded, and a value of 1 means that a macro block (8 pixels) is expanded around the original Mosaic.

Returns:

- ret: int

returns 0 if success.

Sample:

```
#include <sail/cvwrapper.h>
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::vector<std::vector<int>> rects = {
        {100, 100, 50, 50},
        {200, 200, 60, 60}
    };
    int mosaic_num = rects.size(); /
    int is_expand = 0;
    int result = bmcv.mosaic(mosaic_num, BMimg, rects, is_expand);
    if (result == 0) {
        std::cout << "Mosaic applied successfully." << std::endl;
    } else {
        std::.cout << "Failed to apply mosaic." << std::endl;
    }
    return 0;
}
```

4.14.38 transpose

Transpose of image width and height.

Interface1:

```
BMImage Bmcv::transpose(BMImage &src);
```

Parameters1:

- src : BMImage

Input BMImage.

Returns2:

- output: BMImage:

output BMImage.

Interface2:

```
int Bmcv::transpose(
    BMImage &src,
    BMImage &dst);
```

Parameters2:

- src : BMImage

Input BMImage.

- dst : BMImage

output BMImage.

Returns2:

- ret : int

returns 0 if success.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_img.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg_input = decoder.read(handle);
    sail::BMImage BMimg_output;
    sail::Bmcv bmcv(handle);
    int ret = bmcv.transpose(BMimg_input,BMimg_output);
    if(ret != 0){
```

(continues on next page)

(continued from previous page)

```
        std::cout << "gaussian_blur failed" << std::endl;
        return -1;
    }
    bmcv.imwrite("output.jpg",BMimg_output);

    return 0;
}
```

4.14.39 watermark_superpose

Implement adding multiple watermarks to images.

接口形式:

```
int Bmcv::watermark_superpose(
BMImage &img,
string water_name,
int bitmap_type,
int pitch,
vector<vector<int>> rects,
const std::tuple<int, int, int> &color);
```

参数说明:

- Image: BMImage

Input image

- Watername: string

Watermark file path

- Bitmap_type: int

Input parameters. Watermark type, a value of 0 indicates that the watermark is an 8-bit data type (with transparency information), and a value of 1 indicates that the watermark is a 1-bit data type (without transparency information).

- Pitch: int

Input parameters. The number of bytes per line in a watermark file can be understood as the width of the watermark.

- Rects: vector

Input parameters. Watermark position, including the starting point and width/height of each watermark.

- Color: const std:: tuple<int, int, int>

Input parameters. The color of the watermark.

Return value description:

- Ret: int

Whether the return was successful

Sample:

```
#include <sail/cvwrapper.h>
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::string water_name = "your_watermark_path";
    int bitmap_type = 0;
    int pitch =117;
    std::vector<std::vector<int>> rects = {
        {10, 10, 117, 79},
        {200, 150, 117, 79}
    };
    std:: vector<int> color = {128,128,128};
    int result = bmcv.watermark_superpose(BMimg, water_name, bitmap_type, pitch, rects,
↪ color);
    if (result == 0) {
        std::cout << "Watermarks added successfully." << std::endl;
    } else {
        std::cout << "Failed to add watermarks." << std::endl;
    }
    return 0;
}
```

4.14.40 gaussian_blur

This interface is used for image Gaussian filtering. **Note: The previous SDK does not support BM1684X. For details about whether the current SDK supports BM1684X, check the page "BMCV API" in《Multimedia User Guide》. ***

Interface1:

```
int gaussian_blur(
    BMImage                 &input,
    BMImage                 &output,
    int                     kw,
    int                     kh,
    float                   sigmaX,
    float                   sigmaY = 0.0f);
```

Parameters1:

- input : BMImage

Input BMImage.

- output : BMImage

Output BMImage.

- kw : int

The size of kernel in the width direction.

- kh : int

The size of kernel in the height direction.

- sigmaX : float

Gaussian kernel standard deviation in the X direction.

- sigmaY : float

Gaussian kernel standard deviation in the Y direction.Default is 0, which means that it is the same standard deviation as the Gaussian kernel in the X direction.

Returns1:

- ret: int

returns 0 if success.

Interface2:

```
BMImage gaussian_blur(
    BMImage                 &input,
    int                     kw,
    int                     kh,
    float                   sigmaX,
    float                   sigmaY = 0.0f);
```

Parameters2:

- input : BMImage

Input BMImage.

- kw : int

The size of kernel in the width direction.

- kh : int

The size of kernel in the height direction.

- sigmaX : float

Gaussian kernel standard deviation in the X direction.

- sigmaY : float

Gaussian kernel standard deviation in the Y direction.Default is 0, which means that it is the same standard deviation as the Gaussian kernel in the X direction.

Returns2:

- output : BMImage

Returns a Gaussian filtered image.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_img.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg_input = decoder.read(handle);
    sail::BMImage BMimg_output;
    sail::Bmcv bmcv(handle);
    int ret = bmcv.gaussian_blur(BMimg_input,BMimg_output,3, 3, 0.1);
    if(ret != 0){
        std::cout << "gaussian_blur failed" << std::endl;
        return -1;
    }
    bmcv.imwrite("output.jpg",BMimg_output);

    return 0;
}
```

4.14.41 Sobel

Sobel operator for edge detection.

Note: For details about whether this operator in current SDK supports BM1684X/BM1688, check the page "BMCV API" in 《Multimedia User Guide》.

Interface1:

```
int Sobel(
    BMImage &input,
    BMImage &output,
    int dx,
    int dy,
    int ksize = 3,
    float scale = 1,
    float delta = 0);
```

Parameters1:

- input

Input BMImage

- output

Output BMImage

- dx

Order of the derivative x.

- dy

Order of the derivative y

- ksize

ize of the extended Sobel kernel; it must be -1, 1, 3, 5, or 7. -1 means 3x3 Scharr filter will be used.

- scale

Optional scale factor for the computed derivative values; by default, no scaling is applied

- delta

Optional delta value that is added to the results prior to storing them in dst.

Returns1:

- ret: int

returns 0 if success.

Interface2:

```
BMImage Sobel(
    BMImage &input,
    int dx,
    int dy,
    int ksize = 3,
    float scale = 1,
    float delta = 0);
```

Parameters2:

- input

Input BMImage

- dx

Order of the derivative x.

- dy

Order of the derivative y

- ksize

ize of the extended Sobel kernel; it must be -1, 1, 3, 5, or 7. -1 means 3x3 Scharr filter will be used.

- scale

Optional scale factor for the computed derivative values; by default, no scaling is applied

- delta

Optional delta value that is added to the results prior to storing them in dst.

Returns2:

- output: BMImage

returns porcessed BMImage.

Sample:

```
#include <sail/cvwrapper.h>

int main() {
    int dev_id = 0;
    sail::Handle handle(dev_id);
    std::string image_name = "your_img.jpg";
    sail::Decoder decoder(image_name, true, dev_id);
    sail::BMImage BMimg_input = decoder.read(handle);
    sail::BMImage BMimg_output;
    sail::Bmcv bmcv(handle);
    int ret = bmcv.Sobel(BMimg_input,BMimg_output,1,1);
    if(ret != 0){
        std::cout << "Sobel failed" << std::endl;
        return -1;
    }
    bmcv.imwrite("output.jpg",BMimg_output);

    return 0;
}
```

4.14.42 drawLines

This function can be used to draw one or several line segments on an image, which can be used to draw polygons. It supports specifying the color and thickness of the lines.

Note: For details about whether this operator in current SDK supports BM1684X, check the page "BMCV API" in 《Multimedia User Guide》 .

Interface:

```
int Bmcv::drawLines(
BMImage &image,
std::vector<std::pair<int,int>> &start_points,
std::vector<std::pair<int,int>> &end_points,
```

(continues on next page)

(continued from previous page)

```
int line_num,
std::tuple<unsigned char, unsigned char, unsigned char> color,
int thickness
);
```

Parameters:

- image : BMImage

Input image, needs to be one of the supported formats.

- start_points : std::vector<std::pair<int,int>>

List of coordinates for the starting points of the line segments.

- end_points : std::vector<std::pair<int,int>>

List of coordinates for the ending points of the line segments. The size of start_points and end_points must be the same and match the line_num parameter.

- line_num : int

The number of line segments to draw.

- color : std::tuple<unsigned char, unsigned char, unsigned char>

The color of the line segments, corresponding to the RGB channels.

- thickness : int

The thickness of the line segments.

Returns:

- ret: int

returns 0 if success.

Sample:

```
#include <sail/cvwrapper.h>
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    std::string image_name = "your_image_path";
    sail::Decoder decoder(image_name, true, tpu_id);
    sail::BMImage BMimg = decoder.read(handle);
    sail::Bmcv bmcv(handle);
    std::vector<std::pair<int, int>> start_points = {{100, 100}, {200, 200}};
    std::vector<std::pair<int, int>> end_points = {{150, 150}, {250, 250}};
    int line_num = 2;
    std::tuple<unsigned char, unsigned char, unsigned char> color = std::make_tuple(255, 0,
↪ 0);
    int thickness = 2;
```

(continues on next page)

(continued from previous page)

```
    sail::BMImage BMimg2;
    BMimg2 = bmcv.vpp_convert_format(BMimg,FORMAT_YUV420P);
    int ret = bmcv.drawLines(BMimg2, start_points, end_points, line_num, color, thickness);

    return 0;
}
```

4.14.43 stft

Short-Time Fourier Transform(STFT)

Note: For details about whether this operator in current SDK supports BM1684X, check the page "BMCV API" in 《Multimedia User Guide》.

Interface1:

```
std::tuple<Tensor, Tensor> stft(
    Tensor &input_real,
    Tensor &input_imag,
    bool realInput,
    bool normalize,
    int n_fft,
    int hop_len,
    int pad_mode,
    int win_mode
    );
```

参数说明:

- **input_real: numpy.ndarray or Tensor** The real part of the input signal.
- **input_imag: numpy.ndarray or Tensor** The imaginary part of the input signal.
- **real_input: bool** A flag indicating whether to use only real input.
- **normalize: bool** A flag indicating whether to normalize the output.
- **n_fft: int** The number of FFT points used in the STFT computation.
- **hop_len: int** The step size for sliding the window.
- **pad_mode: int** The padding mode for the input signal.
- **win_mode: int** The type of window function.

Returns:

- **result: tuple[Tensor, Tensor]** Returns the real part and imaginary part of the output.

Sample:

```cpp
#include <sail/cvwrapper.h>
using namespace std;
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    std::vector<int> shape = {2, 4096};
    sail::Tensor input_real(shape);
    sail::Tensor input_imag(shape);
    bool real_input = false;
    bool normalize = true;
    int n_fft = 1024;
    int hop_len = 256;
    int pad_mode = 0;  // 填充模式示例
    int win_mode = 1;  // 窗口类型示例
    std::tuple<sail::Tensor, sail::Tensor> result = bmcv.stft(input_real, input_imag,F
↪realInput, normalize, n_fft, hop_len, pad_mode, win_mode);
    return 0;
}
```

4.14.44 istft

Inverse Short-Time Fourier Transform(ISTFT)

Note: For details about whether this operator in current SDK supports BM1684X, check the page "BMCV API" in 《Multimedia User Guide》.

Interface1:

```cpp
std::tuple<Tensor, Tensor> istft(
    Tensor &input_real,
    Tensor &input_imag,
    bool realInput,
    bool normalize,
    int L,
    int hop_len,
    int pad_mode,
    int win_mode
    );
```

Parameters:

- **input_real: numpy.ndarray or Tensor** The real part of the input signal.
- **input_imag: numpy.ndarray or Tensor** The imaginary part of the input signal.
- **real_input: bool** Indicates whether the output signal is real; false for complex, true for real.
- **normalize: bool** Whether to normalize the output.

- **L: int** The length of the original time-domain signal.
- **hop_len: int** The step size for sliding the window; must match the value used during STFT computation.
- **pad_mode: int** The padding mode for the input signal; must match the value used during STFT computation.
- **win_mode: int** The type of window function; must match the value used during STFT computation.

Returns:

- **result: tuple[Tensor, Tensor]** Returns the real part and imaginary part of the output.

Sample:

```
#include <sail/cvwrapper.h>
using namespace std;
int main()
{
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);
    std::vector<int> shape = {2, 513, 17};
    sail::Tensor input_real(shape);
    sail::Tensor input_imag(shape);
    bool real_input = false;
    bool normalize = true;
    int L = 4096;
    int hop_len = 256;
    int pad_mode = 0;
    int win_mode = 1;
    std::tuple<sail::Tensor, sail::Tensor> result = bmcv.istft(input_real, input_imag,F
↪realInput, normalize, L, hop_len, pad_mode, win_mode);
    return 0;
}
```

4.14.45 faiss_indexflatL2

Calculate squared L2 distance between query vectors and database vectors, output the top topK L2sqr-values and the corresponding indices.

Interface:

```
std::tuple<Tensor, Tensor> faiss_indexflatL2(
    Tensor &query_vecs,
    Tensor &query_vecs_L2norm,
    Tensor &database_vecs,
    Tensor& database_vecs_L2norm,
    int vec_dims,
    int query_vecs_nums,
```

(continues on next page)

(continued from previous page)

```
    int database_vecs_nums,
    int topK
    );
```

Parameters:

- **query_vecs: Tensor** The query vectors, the supported data type is only sail.Dtype.BM_FLOAT32.
- **query_vecs_L2norm: Tensor** Calculates the sum of the square values of the elements in each row of the query vector, the data type is sail.Dtype.BM_FLOAT32.
- **database_vecs: Tensor** The database vectors, the supported data type is only sail.Dtype.BM_FLOAT32.
- **database_vecs_L2norm: Tensor** Calculates the sum of the square values of the elements in each row of the database vector, the data type is sail.Dtype.BM_FLOAT32..
- **vec_dims: int** The dimension of the query vectors and database vectors.
- **query_vecs_nums: int** The numbers of the query vectors.
- **database_vecs_nums: int** The numbers of the database vectors.
- **topK: int** Get top topK values.

Returns:

- **result: tuple[Tensor, Tensor]** Returns the square value of the top topK best-matched L2 distance and its corresponding index.

***Sample:**

```
#include <iostream>
#include <vector>
#include <sail/cvwrapper.h>

int main(){
    // 1. database_vecs
    std::vector<std::vector<float>> db_vecs = {
        {-2.0f, 0.0f, 4.0f},
        {-5.0f, 3.0f, -1.0f},
        {1.0f, 2.0f, 4.0f},
        {0.0f, 5.0f, -3.0f},
        {2.0f, 1.0f, -4.0f}
    };

    // 2. query_vecs
    std::vector<std::vector<float>> query_vecs = {
        {1.0f, 2.0f, 3.0f}
    };
```

(continues on next page)

(continued from previous page)

```
    // 3. test bmcv.faiss_indexflatL2
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);

    std::vector<float> db_vecs_l2norm;
    for (const auto& vec : db_vecs) {
        float sum = 0.0f;
        for (const auto& val : vec) {
            sum += val * val;
        }
        db_vecs_l2norm.push_back(sum);
    }

    std::vector<float> query_vecs_l2norm;
    for (const auto& vec : query_vecs) {
        float sum = 0.0f;
        for (const auto& val : vec) {
            sum += val * val;
        }
        query_vecs_l2norm.push_back(sum);
    }

    sail::Tensor database_vecs_tensor = Tensor(handle, {db_vecs.size(), db_vecs[0].size()},F
↪BM_FLOAT32, true, true);
    database_vecs_tensor.reset_sys_data(db_vecs.data(), {db_vecs.size(), db_vecs[0].size()}
↪);
    database_vecs_tensor.sync_s2d();

    sail::Tensor database_vecs_L2norm_tensor = Tensor(handle, {1, db_vecs.size()}, BM_
↪FLOAT32, true, true);
    database_vecs_L2norm_tensor.reset_sys_data(db_vecs_l2norm, {1, db_vecs.size()});
    database_vecs_L2norm_tensor.sync_s2d();

    sail::Tensor query_vecs_tensor = Tensor(handle, {query_vecs.size(), query_vecs[0].size()}
↪, BM_FLOAT32, true, true);
    query_vecs_tensor.reset_sys_data(query_vecs.data(), {query_vecs.size(), query_vecs[0].
↪size()});
    query_vecs_tensor.sync_s2d();

    sail::Tensor query_vecs_L2norm_tensor = Tensor(handle, {1, query_vecs.size()}, BM_
↪FLOAT32, true, true);
    query_vecs_L2norm_tensor.reset_sys_data(query_vecs_l2norm.data(), {1, query_vecs.
↪size()});
    query_vecs_L2norm_tensor.sync_s2d();

    std::tuple<sail::Tensor, sail::Tensor> result = bmcv.faiss_indexflatL2(query_vecs_tensor,
↪ query_vecs_L2norm_tensor, database_vecs_tensor, database_vecs_L2norm_tensor, 3,F
↪1, 5, 3);

    return 0;
```

(continues on next page)

(continued from previous page)

```
}
```

4.14.46 faiss_indexflatIP

Calculate inner product distance between query vectors and database vectors, output the top K IP-values and the corresponding indices.

Interface:

```
std::tuple<Tensor, Tensor> faiss_indexflatIP(
    Tensor &query_vecs,
    Tensor &database_vecs,
    int vec_dims,
    int query_vecs_nums,
    int database_vecs_nums,
    int topK
    );
```

Parameters:

- **query_vecs: Tensor** The query vectors, the supported data type is only sail.Dtype.BM_FLOAT32.
- **database_vecs: Tensor** The database vectors, the supported data type is only sail.Dtype.BM_FLOAT32.
- **vec_dims: int** The dimension of the query vectors and database vectors.
- **query_vecs_nums: int** The numbers of the query vectors.
- **database_vecs_nums: int** The numbers of the database vectors.
- **topK: int** Get top topK values.

Returns:

- **result: tuple[Tensor, Tensor]** Returns the top topK best-matched inner product distance values and their corresponding indexes.

Sample:

```
#include <iostream>
#include <vector>
#include <sail/cvwrapper.h>

int main(){
    // 1. database_vecs
    std::vector<std::vector<float>> db_vecs = {
        {-2.0f, 0.0f, 4.0f},
        {-5.0f, 3.0f, -1.0f},
        {1.0f, 2.0f, 4.0f},
```

(continues on next page)

(continued from previous page)

```
        {0.0f, 5.0f, -3.0f},
        {2.0f, 1.0f, -4.0f}
    };

    // 2. query_vecs
    std::vector<std::vector<float>> query_vecs = {
        {1.0f, 2.0f, 3.0f}
    };

    // 3. test bmcv.faiss_indexflatL2
    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);

    sail::Tensor database_vecs_tensor = Tensor(handle, {db_vecs.size(), db_vecs[0].size()},F
↪BM_FLOAT32, true, true);
    database_vecs_tensor.reset_sys_data(db_vecs.data(), {db_vecs.size(), db_vecs[0].size()}
↪);
    database_vecs_tensor.sync_s2d();

    sail::Tensor query_vecs_tensor = Tensor(handle, {query_vecs.size(), query_vecs[0].size()}
↪, BM_FLOAT32, true, true);
    query_vecs_tensor.reset_sys_data(query_vecs.data(), {query_vecs.size(), query_vecs[0].
↪size()});
    query_vecs_tensor.sync_s2d();

    std::tuple<sail::Tensor, sail::Tensor> result = bmcv.faiss_indexflatIP(query_vecs_tensor,
↪ database_vecs_tensor, 3, 1, 5, 3);

    return 0;
}
```

4.14.47 faiss_indexPQ_encode

Perform PQ quantization encoding on the input vector and output the encoded vector.

Interface:

```
int faiss_indexPQ_encode(
    Tensor &input_vecs,
    Tensor &centroids_vecs,
    Tensor &encoded_vecs,
    int encode_vecs_num,
    int vec_dims,
    int slice_num,
    int centroids_num,
    int IP_metric
);
```

Parameters:

- **input_vecs: Tensor** Input vector to be encoded, data types are supported only numpy.float32 or sail.Dtype.BM_FLOAT32, encode_vecs_num * vec_dims.
- **centroids_vecs: Tensor** centroids vector, data types are supported only numpy.float32 or sail.Dtype.BM_FLOAT32, slice_num * centroids_num * (vec_dims / slice_num).
- **encoded_vecs: Tensor** Output the encoded vector. The data type is BM_UINT8 and the size is encode_vecs_num * slice_num.
- **encode_vecs_num: int** The number of vectors to be encoded.
- **vec_dims: int** Dimension of the vector to be encoded.
- **slice_num: int** The number of slices of the original vector dimensions, for example the original vector dimension is 512, slice_num = 8, and each subvector dimension is 64.
- **centroids_num: int** The number of centroids.
- **IP_metric: int** 0 indicates that the distance is calculated using L2, and 1 indicates that the distance is calculated using IP.

Returns:

- **result: int** Returns 0 on success.

Sample:

```cpp
#include <iostream>
#include <vector>
#include <sail/cvwrapper.h>

int main()
{
    int encode_vecs_num = 3;
    int vec_dims = 64;
    int db_vecs_num = 10000;
    int slice_num = 8;
    int centroids_num = 256;
    int subvec_dims = vec_dims / slice_num;

    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);

    // centroids Tensor
    std::vector<int> centroids_shape = {slice_num, centroids_num, subvec_dim};
    sail::Tensor centroids_tensor(handle, centroids_shape, BM_FLOAT32, false, true);

    // input Tensor
    std::vector<int> input_shape = {encode_vecs_num, vec_dims};
    sail::Tensor input_tensor(handle, input_shape, BM_FLOAT32, false, true);
```

(continues on next page)

(continued from previous page)

```
    // encoded Tensor
    std::vector<int> encoded_shape = {encode_vecs_num, slice_num};
    sail::Tensor encoded_tensor(handle, encoded_shape, BM_UINT8, true, true);

    int ret = 0;
    ret = bmcv.faiss_indexPQ_encode(input_tensor,
                                    centroids_tensor,
                                    encoded_tensor,
                                    encode_vecs_num,
                                    vec_dims,
                                    slice_num,
                                    centroids_num,
                                    0);

    return ret;
}
```

4.14.48 faiss_indexPQ_ADC

The distance table is calculated by the query vector and the clustering center (centroid) vector, and the encoded database vector searches the table to calculate the distance and sorts it, output the topK distances and the corresponding indices.

Interface:

```
std::tuple<Tensor, Tensor> faiss_indexPQ_ADC (
    Tensor &nxquery_vecs,
    Tensor &centroids_vecs,
    Tensor &nycodes_vecs,
    int vec_dims,
    int slice_num,
    int centroids_num,
    int database_vecs_num,
    int query_vecs_num,
    int topK,
    int IP_metric
);
```

Parameters:

- **nxquery_vecs: Tensor** query vectors, data types are supported only sail.Dtype.BM_FLOAT32, and the size is query_vecs_nums * vec_dims.
- **centroids_vecs: Tensor** centroids vectors, data types are supported only sail.Dtype.BM_FLOAT32, and the size is slice_num * centroids_num * (vec_dims / slice_num).
- **nycodes_vecs: Tensor** Encoded database vector, data types are supported only sail.Dtype.BM_UINT8, and the size is database_vecs_nums * slice_num.

- **vec_dims: int** dimension of the query vector.
- **slice_num: int** The number of slices of the original vector dimensions, for example the original vector dimension is 512, slice_num = 8, and each subvector dimension is 64.
- **centroids_num: int** The number of centroids.
- **database_vecs_nums: int** The number of database vectors.
- **query_vecs_nums: int** The number of query vectors.
- **topK: int** Get top topK values.
- **IP_metric: int** 0 indicates that the distance is calculated using L2, and 1 indicates that the distance is calculated using IP.

Returns:

- **result: tuple[Tensor, Tensor]** Returns the top topK best-matched distance values and their corresponding indexes.

Sample:

```
#include <iostream>
#include <vector>
#include <sail/cvwrapper.h>

int main()
{
    int vec_dims = 512;
    int slice_num = 8;
    int centroids_num = 256;
    int database_vecs_num = 10000;
    int query_vecs_num = 1;
    int subvec_dims = vec_dims / slice_num;
    int topK = 5;

    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);

    // nxquery Tensor
    std::vector<int> nxquery_shape = {query_vecs_num, vec_dims};
    sail::Tensor nxquery_tensor(handle, nxquery_shape, BM_FLOAT32, false, true);

    // centroids Tensor
    std::vector<int> centroids_shape = {slice_num, centroids_num, subvec_dim};
    sail::Tensor centroids_tensor(handle, centroids_shape, BM_FLOAT32, false, true);

    // nycodes Tensor
    std::vector<int> nycodes_shape = {database_vecs_nums, slice_num};
    sail::Tensor nycodes_tensor(handle, nycodes_shape, BM_UINT8, true, true);
```

(continues on next page)

(continued from previous page)

```
    std::tuple<sail::Tensor, sail::Tensor> results = bmcv.faiss_indexPQ_ADC(nxquery_
↪tensor,
                                centroids_tensor,
                                nycodes_tensor,
                                vec_dims,
                                slice_num,
                                centroids_num,
                                database_vecs_num,
                                query_vecs_num,
                                topK,
                                0);

    return results;
}
```

4.14.49 faiss_indexPQ_SDC

The Symmetric Distance Computation (SDC) lookup table is used to speed up the distance calculation between PQ encodings, output the topK distances and the corresponding indices.

Interface:

```
std::tuple<Tensor, Tensor> faiss_indexPQ_SDC (
    Tensor &nxcodes_vecs,
    Tensor &nycodes_vecs,
    Tensor &sdc_table,
    int slice_num,
    int centroids_num,
    int database_vecs_num,
    int query_vecs_num,
    int topK,
    int IP_metric
);
```

Parameters:

- **nxcodes_vecs: Tensor** Encoded query vectors, data types are supported only sail.Dtype.BM_UINT8, and the size is query_vecs_nums * slice_num.
- **nycodes_vecs: Tensor** Encoded database vector, data types are supported only sail.Dtype.BM_UINT8, and the size is database_vecs_nums * slice_num.
- **sdc_table: Tensor** The Symmetric Distance Computation (SDC) lookup table, data types are supported only sail.Dtype.BM_FLOAT32, and the size is slice_num * centroids_num * centroids_num.
- **slice_num: int** The number of slices of the original vector dimensions, for example the original vector dimension is 512, slice_num = 8, and each subvector dimension is 64.
- **centroids_num: int** The number of centroids.

- **database_vecs_nums: int** The number of database vectors.
- **query_vecs_nums: int** The number of query vectors.
- **topK: int** Get top topK values.
- **IP_metric: int** 0 indicates that the distance is calculated using L2, and 1 indicates that the distance is calculated using IP.

Returns:

- **result: tuple[Tensor, Tensor]** Returns the top topK best-matched distance values and their corresponding indexes.

Sample:

```
#include <iostream>
#include <vector>
#include <sail/cvwrapper.h>

int main()
{
    int vec_dims = 512;
    int slice_num = 8;
    int centroids_num = 256;
    int database_vecs_num = 10000;
    int query_vecs_num = 1;
    int subvec_dims = vec_dims / slice_num;
    int topK = 5;

    int tpu_id = 0;
    sail::Handle handle(tpu_id);
    sail::Bmcv bmcv(handle);

    // nxcodes Tensor
    std::vector<int> nxcodes_shape = {query_vecs_num, slice_num};
    sail::Tensor nxcodes_tensor(handle, nxcodes_shape, BM_UINT8, false, true);

    // nycodes Tensor
    std::vector<int> nycodes_shape = {database_vecs_nums, slice_num};
    sail::Tensor nycodes_tensor(handle, nycodes_shape, BM_UINT8, false, true);

    // sdc_table Tensor
    std::vector<int> sdc_shape = {slice_num, centroids_num, centroids_num};
    sail::Tensor sdc_tensor(handle, sdc_shape, BM_FLOAT32, false, true);


    std::tuple<sail::Tensor, sail::Tensor> results = bmcv.faiss_indexPQ_SDC(nxcodes_
↪tensor,
                                                                            nycodes_tensor,
                                                                            sdc_tensor,
                                                                            slice_num,
                                                                            centroids_num,
                                                                            database_vecs_num,
```

(continues on next page)

(continued from previous page)

```
                                                    query_vecs_num,
                                                    topK,
                                                    0);

    return results;
}
```

4.15 Blend

Picture stitching. Multiple pictures can be stitched together. Only BM1688 and CV186AH SoC mode support this interface.

4.15.1 Constructor Blend()

Initialize Blend

Interface:

```
Blend(int src_h, std::vector<std::vector<short>> ovlap_attr, std::vector<std::vector
↪<short>> bd_attr, std::vector<std::vector<string>> wgt_phy_mem, bm_stitch_wgt_
↪mode wgt_mode)
```

Parameters:

- src_h: int

Input images. The height of the input images must be consistent.

- ovlap_attr: std::vector<std::vector<short>>

The left and right boundaries of the images overlap area.

- bd_attr: std::vector<std::vector<short>>

The left and right black border properties of the input images. This option is not currently supported, so just leave it blank.

- wgt_phy_mem: std::vector<std::vector<string>>

Weights file for input images. Reference[Get weight files of images], get image weight files。

- wgt_mode: bm_stitch_wgt_mode

The stitching mode of the input image. Currently the following two stitching modes are supported, usually BM_STITCH_WGT_YUV_SHARE is selected.

- BM_STITCH_WGT_YUV_SHARE: YUV share alpha and beta(1 alpha + 1 beta)
- BM_STITCH_WGT_UV_SHARE: UV share alpha and beta(2 alpha + 2 beta)

4.15.2 process

Perform image stitching.

Interface:

```
int process(std::vector<BMImage*> &input, BMImage &output)
```

Returns

- ret: int

Returns whether the image stitching is successful, 0 means success, other values mean failure.

Parameters:

- input: std::vector<BMImage*>

Input BMImage to be stitched.

- output: BMImage

Output stitched BMImage.

Interface:

```
BMImage process(std::vector<BMImage*> &input)
```

Returns

- output: BMImage

Return stitched BMImage.

Parameters:

- input: std::vector<BMImage*>

Input BMImage to be stitched.

Sample:

```
#include "cvwrapper.h"

int main(){
    Handle handle(0);
    Decoder decoder("./left.jpg", False, 0);
    BMImage image_left;
    decoder.read(handle, image_left);

    Decoder decoder("./right.jpg", False, 0);
    BMImage image_right;
    decoder.read(handle, image_left);

    Blend blend_tmp(2240, {{2112},{2239}}, {}, {{"data/wgt/c01_alpha_444p_m2__0_
↪2240x128.bin","data/wgt/c01_beta_444p_m2__0_2240x128.bin"}}, BM_STITCH_
↪WGT_YUV_SHARE);
```

(continues on next page)

(continued from previous page)

```
    BMImage bmimg = blend_tmp.process({image_left, image_right});
    sail::Bmcv bmcv(handle);
    int ret = bmcv.imwrite("result.jpg", img);
}
```

4.16 MultiDecoder

Multi-channel decoding interface supports decoding multiple channels of video at the same time.

4.16.1 Constructor

Interface:

```
MultiDecoder(
    int queue_size=10,
    int tpu_id=0,
    int discard_mode=0);
```

Parameters:

- queue_size: int

Input parameter. For each channel of video, the length of the decoding cache image queue.

- tpu_id: int

Input parameter. The tpu id used, the default is 0.

- discard_mode: int

Input parameter. The data discarding strategy after the cache reaches the maximum size. 0 means not to put data into the cache; 1 means to take out the picture at the head of the queue from the queue, discard it, and then cache the decoded picture. Default is 0.

4.16.2 set_read_timeout

Set the timeout for reading images, which takes effect on the read and read_ interfaces. If the image is not obtained after the timeout, the result will be returned.

Interface:

```
void set_read_timeout(int time_second);
```

Parameters:

- timeout: int

Input parameter. Timeout time, unit is seconds.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    multiDecoder.set_read_timeout(100);
    return 0;
}
```

4.16.3 add_channel

Add a channel.

Interface:

```
int add_channel(
    const std::string&  file_path,
    int                 frame_skip_num=0);
```

Parameters:

- file_path: string

Input parameter. The path or link to the video.

- frame_skip_num: int

Input parameter. The number of active frame drops in the decoding cache. The default is 0, which means no active frame drops.

Returns

Returns the unique channel number corresponding to the video. Type is int.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
```

(continues on next page)

(continued from previous page)

```
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
    return 0;
}
```

4.16.4 del_channel

Delete an already added video channel.

Interface:

```
int del_channel(int channel_idx);
```

Parameters:

- channel_idx: int

Input parameter. The channel number of the video to be deleted.

Returns:

Returns 0 on success, other values indicate failure.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
    int ret = multiDecoder.del_channel(0);
    if (ret!=0) {
        cout << "Failed!" << endl;
        return -1;
```

(continues on next page)

(continued from previous page)

```
    }
    return 0;
}
```

4.16.5 clear_queue

Clear the image cache of the specified channel.

Interface:

```
int clear_queue(int channel_idx);
```

Parameters:

- channel_idx: int

Input parameter. The channel number of the video to be deleted.

Returns:

Returns 0 on success, other values indicate failure.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
    int ret = multiDecoder.clear_queue(0);
    if (ret!=0) {
        cout << "Failed!" << endl;
        return -1;
    }
    return 0;
}
```

4.16.6 read

Get an image from the specified video channel.

Interface 1:

```
int read(
    int         channel_idx,
    BMImage&    image,
    int         read_mode=0);
```

Parameters 1:

- channel_idx: int

Input parameter. Specified video channel number.

- image: BMImage

Output parameter. Decoded picture.

- read_mode: int

Input parameter. The mode for getting images, 0 means no waiting, read one directly from the cache, and it will return regardless of whether it is read or not. Others indicate waiting until the image is obtained before returning.

Returns 1:

Returns 0 on success, other values indicate failure.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }

    int count = 0;
    while (true) {
        count++;
        for (int idx : channel_list) {
```

(continues on next page)

(continued from previous page)

```
            sail::BMImage bmimg;
            int ret = multiDecoder.read(idx, bmimg, 1);
        }
        if (count == 20) {
            break;
        }
    }
    return 0;
}
```

Interface 2:

```
BMImage read(int channel_idx);
```

Parameters 2:

- channel_idx: int

Input parameter. Specified video channel number.

返回值说明2:

Returns the decoded image, type is BMImage.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }

    int count = 0;
    while (true) {
        count++;
        for (int idx : channel_list) {
            sail::BMImage bmimg = multiDecoder.read(idx);
        }
        if (count == 20) {
            break;
        }
```

(continues on next page)

(continued from previous page)

```
    }
    return 0;
}
```

4.16.7 read_

Get an image from the specified video channel, usually used with BMImageArray.

Interface 1:

```
int read_(
    int         channel_idx,
    bm_image&   image,
    int         read_mode=0);
```

Parameters 1:

- channel_idx: int

Input parameter. Specified video channel number.

- image: bm_image

Output parameter. Decoded image.

- read_mode: int

Input parameter. The mode for getting images, 0 means no waiting, read one directly from the cache, and it will return regardless of whether it is read or not. Others indicate waiting until the image is obtained before returning.

Returns 1:

Returns 0 on success, other values indicate failure.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    vector<vector<cv::Mat>> frame_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
```

(continues on next page)

(continued from previous page)

```
        if(idx<0) return -1;
        channel_list.push_back(idx);
        frame_list.push_back(vector<cv::Mat>());
    }

    int count = 0;
    while (true) {
        count++;
        for (int idx : channel_list) {
            sail::BMImage image;
            sail::bm_image bmimg = image.data()
            int ret = multiDecoder.read_(idx,(id,1);
        }
        if (count == 20) {
            break;
        }
    }
    return 0;
}
```

Interface 2:

```
bm_image read_(int channel_idx);
```

Parameters 2:

- channel_idx: int

Input parameter. Specified video channel number.

Returns 2:

Returns the decoded image, type bm_image.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
```

(continues on next page)

(continued from previous page)

```
    int count = 0;
    while (true) {
        count++;
        for (int idx : channel_list) {
            bm_image bmimg =  multiDecoder.read_(idx);
        }
        if (count == 20) {
            break;
        }
    }
    return 0;
}
```

4.16.8 reconnect

Reconnect video in the corresponding channel.

Interface:

```
int reconnect(int channel_idx);
```

Parameters:

- channel_idx: int

Input parameter. Enter the channel number of the image.

Returns

Returns 0 on success, other values indicate failure.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>
using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
    }
    int ret = multiDecoder.reconnect(0);
    if (ret!=0) {
        cout << "Failed!" << endl;
```

(continues on next page)

(continued from previous page)

```
        return -1;
    }
    return 0;
}
```

4.16.9 get_frame_shape

Get the image shape of the corresponding channel.

Interface:

```
std::vector<int> get_frame_shape(int channel_idx);
```

Parameters:

Input parameters. Enter the channel number of the image.

Returns:

Returns a list consisting of 1, number of channels, image height, and image width.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
        vector<int> shape = multiDecoder.get_frame_shape(idx);
    }
    return 0;
}
```

4.16.10 set_local_flag

Set whether the video is a local video. If not called, the video is represented as a network video stream.

Interface:

```
void set_local_flag(bool flag);
```

Parameters:

- flag: bool

Standard bit. If it is True, each video channel will be decoded at a fixed rate of 25 frames per second.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>
using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    multiDecoder.set_local_flag(true);
    return 0;
}
```

4.16.11 get_channel_fps

Get the video fps of the specified channel

Interface

```
float get_channel_fps(int channel_idx):
```

Parameters

- channel_idx: int

The specified channel index

Returns

Returns the video fps of the specified channel

Sample:

```cpp
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
        float fps = multiDecoder.get_channel_fps(idx);
    }
    return 0;
}
```

4.16.12 get_drop_num

Obtain the number of dropped frames.

Interface:

```cpp
size_t get_drop_num(int channel_idx);
```

Parameters:

- channel_idx: int

The channel index of the input image.

Sample:

```cpp
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>

using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
```

(continues on next page)

(continued from previous page)

```
        size_t ret = multiDecoder.get_drop_num(idx);
    }
    return 0;
}
```

reset_drop_num

Set the number of dropped frames to num.

Interface:

```
size_t reset_drop_num(int channel_idx);
```

Parameters:

- channel_idx: int

The channel index of the input image.

Sample:

```
#include <sail/cvwrapper.h>
#include <sail/decoder_multi.h>
using namespace std;

int main() {
    int queue_size = 16;
    int dev_id = 0;
    int discard_mode = 0;
    sail::MultiDecoder multiDecoder(queue_size, dev_id, discard_mode);
    vector<int> channel_list;
    for (int i = 0; i < 4; i++) {
        int idx = multiDecoder.add_channel("your_video_path");
        if(idx<0) return -1;
        channel_list.push_back(idx);
        multiDecoder.reset_drop_num(idx);
    }
    return 0;
}
```

4.17 SAIL C++ high performance interface

4.17.1 ImagePreProcess

General preprocessing interface, internally implemented using thread pool.

Constructor

interface:

```
ImagePreProcess(
    int batch_size,
    sail_resize_type resize_mode,
    int tpu_id=0,
    int queue_in_size=20,
    int queue_out_size=20,
    bool use_mat_flag=false);
```

Parameters:

- batch_size: int

Input parameter. The batch size of the output results.

- resize_mode: sail_resize_type

Input parameter. Internal scale transformation method.

- tpu_id: int

Input parameter. The tpu id used, the default is 0.

- queue_in_size: int

Input parameter. Enter the maximum length of the image queue buffer. The default is 20.

- queue_out_size: int

Input parameter. The maximum length of the output Tensor queue cache, the default is 20.

- use_mat_output: bool

Input parameter. Whether to use Mat of OPENCV as the output of the image, the default is False, not used.

SetResizeImageAtrr

Set the properties of the image resizing.

Interface:

```
void SetResizeImageAtrr(
    int output_width,
    int output_height,
    bool bgr2rgb,
    bm_image_data_format_ext  dtype);
```

parameters:

- output_width: int

Input parameter. Image width after scale transformation.

- output_height: int

Input parameter. Image height after scale transformation.

- bgr2rgb: bool

Input parameter. Whether to convert images with BGR to GRB.

- dtype: ImgDtype

Input parameter. The data type after image scale conversion, the current version only supports BM_FLOAT32, BM_INT8, BM_UINT8. Can be set according to the input data type of the model.

SetPaddingAtrr

Setting the properties of Padding only takes effect when resize_mode is BM_PADDING_VPP_NEAREST, BM_PADDING_TPU_NEAREST, BM_PADDING_TPU_LINEAR, or BM_PADDING_TPU_BICUBIC.

interface:

```
void SetPaddingAtrr(
    int padding_b=114,
    int padding_g=114,
    int padding_r=114,
    int align=0);
```

Parameters: * padding_b: int

Input parameter. The b channel pixel value to be pdding, default is 114.

- padding_g: int

Input parameter. The g channel pixel value to be pdding, default is 114.

- padding_r: int

Input parameter. The r channel pixel value to be pdding, default is 114.

- align: int

Input parameter. Image padding is position, 0 means filling from the upper left corner, 1 means center filling, default is 0.

SetConvertAtrr

Set the properties of the linear transformation.

Interface:

```
int SetConvertAtrr(
  const std::tuple<
     std::pair<float, float>,
     std::pair<float, float>,
     std::pair<float, float>> &alpha_beta);
```

Parameters:

- alpha_beta: (a0, b0), (a1, b1), (a2, b2)。输入参数。

  a0 describes the linear transformation coefficient of the 0th channel;

  b0 describes the linear transformation offset of the 0th channel;

  a1 describes the linear transformation coefficient of the first channel;

  b1 describes the linear transformation offset of the first channel;

  a2 describes the linear transformation coefficient of the second channel;

  b2 describes the linear transformation offset of the second channel;

Returns:

If the setting is successful, 0 is returned. If other values are set, the setting fails.

PushImage

Send data.

interface:

```
int PushImage(
   int channel_idx,
   int image_idx,
   BMImage &image);
```

Parameters:

- channel_idx: int

Input parameter. Enter the channel number of the image.

- image_idx: int

Input parameter. Enter the number of the image.

- image: BMImage

Input parameter. The input image.

返回值说明:

Returns 0 if set successfully, other values indicate failure.

GetBatchData

Get the result of processing.

Interface:

```
std::tuple<sail::Tensor,
    std::vector<BMImage>,
    std::vector<int>,
    std::vector<int>,
    std::vector<std::vector<int>>> GetBatchData();
```

Returns: tuple[data, images, channels, image_idxs, padding_attrs]

- data: Tensor

  The processed result Tensor.

- images: std::vector<BMImage>

  Original image sequence.

- channels: std::vector<int>

  The channel sequence of the original image.

- image_idxs: std::vector<int>

  Numbered sequence of original images.

- padding_attrs: std::vector<std::vector<int> >

  Attribute list of the filled image, filled starting point coordinate x, starting point coordinate y, width after scale transformation, height after scale transformation

set_print_flag

Set the flag for printing logs. If this interface is not called, the log will not be printed.

Interface:

```
void set_print_flag(bool print_flag);
```

Returns:

- flag: bool

The printing flag, False means not printing, True means printing.

4.17.2 TensorPTRWithName

Tensor with name

```
struct TensorPTRWithName
{
    TensorPTRWithName(): name(""),data(NULL) { }
    std::string name;
    sail::Tensor* data;
};
```

Parameters

- name: string

The name of the tensor.

- data: Tensor*

The data of the tensor.

4.17.3 EngineImagePreProcess

The image inference interface with preprocessing function uses the thread pool internally, which is more efficient in the Python environment.

Constructor

Interface:

```
EngineImagePreProcess(const std::string& bmodel_path,
                      int tpu_id,
                      bool use_mat_output=false,
                      std::vector<int> core_list = {});
```

Parameters:

- bmodel_path: string

Input parameter. The path to input the model.

- tpu_id: int

Input parameter. The tpu id used.

- use_mat_output: bool

Input parameter. Whether to use Mat of OPENCV as the output of the image, the default is False, not used.

- use_mat_output: bool

When using the processor and BModel that support multi-core inference, you can choose multiple cores to use for inference. By default, N cores starting from core0 are used for inference, and N is determined by the current bmodel. For processors and bmodel models that only support single-core inference, only a single core used for inference is supported, and the input list length of the parameter must be 1, if the length of the incoming list is greater than 1, the inference will be automatically on the core0. Default is empty. If this parameter is not specified, the inference is performed by N cores starting from core 0.

InitImagePreProcess

Initialize the image preprocessing module.

Interface:

```
int InitImagePreProcess(
    sail_resize_type resize_mode,
    bool bgr2rgb=false,
    int queue_in_size=20,
    int queue_out_size=20);
```

Parameters:

- resize_mode: sail_resize_type

Input parameter. Internal scale transformation method.

- bgr2rgb: bool

Input parameter. Whether to convert images with BGR to RGB.

- queue_in_size: int

Input parameter. Enter the maximum length of the image queue buffer. The default is 20. Must not be less than the batch_size of bmodel, if not, the value will be set to batch_size.

- queue_out_size: int

Input parameter. The maximum length of the preprocessing result Tensor queue cache, the default is 20. Must not be less than the batch_size of bmodel, if not, the value will be set to batch_size.

Returns:

Returns 0 on success, fails on other values.

SetPaddingAtrr

Set the properties of Padding. This setting only takes effect when resize_mode is BM_PADDING_VPP_NEAREST, BM_PADDING_TPU_NEAREST, BM_PADDING_TPU_LINEAR, or BM_PADDING_TPU_BICUBIC.

Interface:

```
int SetPaddingAtrr(
    int padding_b=114,
    int padding_g=114,
    int padding_r=114,
    int align=0);
```

Parameters: * padding_b: int

Input parameter. The b channel pixel value to be padding, default is 114.

- padding_g: int

Input parameter. The g channel pixel value to be padding, default is 114.

- padding_r: int

Input parameter. The r channel pixel value to be padding, default is 114.

- align: int

Input parameter. The position of the image padding, 0 means padding from the upper left corner, 1 means padding in the center, the default is 0.

Returns:

Returns 0 on success, fails on other values.

SetConvertAtrr

Set the properties of the linear transformation.

Interface:

```
int SetConvertAtrr(
    const std::tuple<
        std::pair<float, float>,
        std::pair<float, float>,
        std::pair<float, float>> &alpha_beta);
```

Parameters:

- alpha_beta: (a0, b0), (a1, b1), (a2, b2)。输入参数。

  a0 describes the linear transformation coefficient of the 0th channel;

  b0 describes the linear transformation offset of the 0th channel;

a1 describes the linear transformation coefficient of the first channel;

b1 describes the linear transformation offset of the first channel;

a2 describes the linear transformation coefficient of the second channel;

b2 describes the linear transformation offset of the second channel;

Returns:

If the setting is successful, 0 is returned. If other values are set, the setting fails.

PushImage

Send image data

Interface:

```
int PushImage(
    int channel_idx,
    int image_idx,
    BMImage &image);
```

Parameters: * channel_idx: int

Input parameter. The channel id used to input the image.

- image_idx: int

Input parameter. The number of the input image.

- image: BMImage

Input parameter. The input image.

Returns:

Returns 0 on success, fails on other values.

GetBatchData

Obtain the inference results of a batch. When calling this interface, since the returned result type is BMImage, use_mat_output must be False. It is worth noting that output tensors must be manually released.

Interface:

```
std::tuple<std::map<std::string,sail::Tensor*>,
    std::vector<BMImage>,
    std::vector<int>,
    std::vector<int>,
    std::vector<std::vector<int>>> GetBatchData();
```

Returns:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: std::map<std::string,sail::Tensor*>

The result of inference.

- ost_images: std::vector<BMImage>

The original image sequence.

- channels: std::vector<int>

The channel sequence of the original image corresponding to the result.

- image_idxs: std::vector<int>

The number sequence of the original image corresponding to the result.

- padding_attrs: std::vector<std::vector<int> >

The attribute list of the filled image, the starting point coordinate x of the filling, the starting point coordinate y, the width after scale transformation, and the height after scale transformation.

GetBatchData_CV

Obtain the inference result of a batch. When calling this interface, since the returned result type is cv::Mat, use_mat_output must be True. It is worth noting that output tensors must be manually released.

Interface:

```
std::tuple<std::map<std::string,sail::Tensor*>,
    std::vector<cv::Mat>,
    std::vector<int>,
    std::vector<int>,
    std::vector<std::vector<int>>> GetBatchData_CV();
```

Returns:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: std::map<std::string,sail::Tensor*>

The result of inference.

- ost_images: std::vector<cv::Mat>

The original image sequence.

- channels: std::vector<int>

The channel sequence of the original image corresponding to the result.

- image_idxs: std::vector<int>

The number sequence of the original image corresponding to the result.

- padding_attrs: std::vector<std::vector<int> >

The attribute list of the filled image, the starting point coordinate x of the filling, the starting point coordinate y, the width after scale transformation, and the height after scale transformation.

get_graph_name

Get the name of the operation graph of model.

Interface:

```
std::string get_graph_name();
```

Returns:

Returns the name of the first operational graph of the model.

get_input_width

Get the width of the model input.

Interface:

```
int get_input_width();
```

Returns:

Returns the width of the model input.

get_input_height

Get the height of the model input.

Interface:

```
int get_input_height();
```

Returns:

Returns the width of the model input.

get_output_names

Get the name of the model output Tensor.

Interface:

```
std::vector<std::string> get_output_names();
```

Returns:

Returns the names of all output Tensors of the model.

get_output_shape

Get the shape of the specified output Tensor

Interface:

```
std::vector<int> get_output_shape(const std::string& tensor_name);
```

Parameters:

- tensor_name: string

The name of the specified output Tensor.

Returns:

Returns the shape of the specified output Tensor.

4.17.4 Yolov5 post-processing acceleration interfaces

Post-processing acceleration interfaces for single-output and three-output YOLOv5 models.

algo_yolov5_post_1output

For post-processing interfaces with a single output YOLOv5 model, internally implemented using thread pools.

__init__

Interface:

```
algo_yolov5_post_1output(const std::vector<int>& shape,
                         int network_w=640,
                         int network_h=640,
                         int max_queue_size=20,
                         bool input_use_multiclass_nms=true,
                         bool agnostic=false);
```

Parameters:

- shape: std::vector<int>

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- input_use_multiclass_nms: bool

Each detection box has multiple categories.

- agnostic: bool

Algorithm without class-specific NMS

push_data

Input data. The value of batchsize other than 1 is supported.

Interface1:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    TensorPTRWithName input_data,
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

Parameters1:

- channel_idx: std::vector<int>

The channel number of the input image.

- image_idx: std::vector<int>

The sequence number of the input image.

- input_data: TensorPTRWithName

The input data.

- dete_threshold: std::vector<float>

Detection threshold sequence.

- nms_threshold: std::vector<float>

nms threshold.

- ost_w: std::vector<int>

The width of original image.

- ost_h: std::vector<int>

The height of original image.

- padding_attrs: std::vector<std::vector<int> >

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns1:

Return 0 if successful, otherwise failed.

Interface2:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    TensorPTRWithName input_data,
    std::vector<std::vector<float>> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

Parameters2:

- channel_idx: std::vector<int>

The channel number of the input image.

- image_idx: std::vector<int>

The sequence number of the input image.

- input_data: TensorPTRWithName

The input data.

- dete_threshold: std::vector<std::vector<float>>

Detection threshold sequence.

- nms_threshold: std::vector<float>

nms threshold.

- ost_w: std::vector<int>

The width of original image.

- ost_h: std::vector<int>

The height of original image.

- padding_attrs: std::vector<std::vector<int> >

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns2:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

algo_yolov5_post_3output

For post-processing interfaces with a triple output YOLOv5 model, internally implemented using thread pools.

__init__

Interface:

```
algo_yolov5_post_3output(const std::vector<std::vector<int>>& shape,
                         int network_w=640,
                         int network_h=640,
                         int max_queue_size=20
                         bool input_use_multiclass_nms=true,
                         bool agnostic=false);
```

Parameters:

- shape: std::vector<std::vector<int>>

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- input_use_multiclass_nms: bool

Each detection box has multiple categories.

- agnostic: bool

Algorithm without class-specific NMS

push_data

Support input with arbitrary batchsize.

Interface1:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    std::vector<TensorPTRWithName> input_data,
```

(continues on next page)

(continued from previous page)

```
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

Parameters1:

- channel_idx: std::vector<int>

The channel number of the input image.

- image_idx: std::vector<int>

The sequence number of the input image.

- input_data: std::vector<TensorPTRWithName>

The input data, including three outputs.

- dete_threshold: std::vector<float>

Detection threshold sequence.

- nms_threshold: std::vector<float>

nms threshold.

- ost_w: std::vector<int>

The width of original image.

- ost_h: std::vector<int>

The height of original image.

- padding_attrs: std::vector<std::vector<int>>

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns1:

Return 0 if successful, otherwise failed.

Interface2:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    std::vector<TensorPTRWithName> input_data,
    std::vector<std::vector<float>> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

Parameters2:

- channel_idx: std::vector<int>

The channel number of the input image.

- image_idx: std::vector<int>

The sequence number of the input image.

- input_data: std::vector<TensorPTRWithName>

The input data, including three outputs.

- dete_threshold: std::vector<std::vector<float>>

Detection threshold sequence.

- nms_threshold: std::vector<float>

nms threshold.

- ost_w: std::vector<int>

The width of original image.

- ost_h: std::vector<int>

The height of original image.

- padding_attrs: std::vector<std::vector<int>>

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns2:

Return 0 if successful, otherwise failed.

get_result_npy

获取最终的检测结果

Interface:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

reset_anchors

Update anchor sizes

Interface:

```
int reset_anchors(std::vector<std::vector<std::vector<int>>> anchors_new);
```

Parameters:

- anchors_new: std::vector<std::vector<std::vector<int> > >

List of anchor sizes to update

Returns:

Return 0 if successful, otherwise failed.

algo_yolov5_post_cpu_opt

The post-processing is accelerated for the 3-output or 1-output yolov5 model.

__init__

Interface:

```
algo_yolov5_post_cpu_opt(const std::vector<std::vector<int>>& shape,
                         int network_w=640, int network_h=640);
```

Parameters:

- shape: std::vector<std::vector<int> >

Input parameter. Shape of input data.

- network_w: int

Input parameter. Input width of the model, default is 640.

- network_h: int

Input parameter. Input height of the model, default is 640.

process

Processing interface.

Interface 1:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(std::vector
↪<TensorPTRWithName> &input_data,
      std::vector<int> &ost_w,
      std::vector<int> &ost_h,
      std::vector<float> &dete_threshold,
      std::vector<float> &nms_threshold,
      bool input_keep_aspect_ratio,
      bool input_use_multiclass_nms);
```

Parameters 1:

- input_data: std::vector<TensorPTRWithName>

Input parameter. Input data with three outputs or one output.

- ost_w: std::vector<int>

Input parameter. Width of original images.

- ost_h: std::vector<int>

Input parameter. Height of original images.

- dete_threshold: std::vector<float>

Input parameter. Detection threshold.

- nms_threshold: std::vector<float>

Input parameter. NMS threshold.

- input_keep_aspect_ratio: bool

Input parameter. Whether to keep aspect ratio in images.

- input_use_multiclass_nms: bool

Input parameter. Whether to use multiclass nms.

Interface 2:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(std::map
↪<std::string, Tensor&>& input_data,
        std::vector<int> &ost_w,
        std::vector<int> &ost_h,
        std::vector<float> &dete_threshold,
        std::vector<float> &nms_threshold,
        bool input_keep_aspect_ratio,
        bool input_use_multiclass_nms);
```

Parameters 2:

- input_data: std::map<std::string, Tensor&>

Input parameter. Input data with three outputs or one output.

- ost_w: std::vector<int>

Input parameter. Width of original images.

- ost_h: std::vector<int>

Input parameter. Height of original images.

- dete_threshold: std::vector<float>

Input parameter. Detection threshold.

- nms_threshold: std::vector<float>

Input parameter. NMS threshold.

- input_keep_aspect_ratio: bool

Input parameter. Whether to keep aspect ratio for boxes.

- input_use_multiclass_nms: bool

Input parameter. Whether to multiclass nms.

Returns:

std::vector<std::vector<std::tuple<left, top, right, bottom, class_id, score> > >

- left: int

The leftmost x-coordinate of the detection result.

- top: int

The topmost y-coordinate of the detection result.

- right: int

The rightmost x-coordinate of the detection result.

- bottom: int

The bottommost y-coordinate of the detection result.

- class_id: int

The class label of the detection result.

- score: float

The score of the detection result.

Interface 3:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(std::vector
↪<TensorPTRWithName> &input_data,
        std::vector<int> &ost_w,
        std::vector<int> &ost_h,
        std::vector<std::vector<float>> &dete_threshold,
        std::vector<float> &nms_threshold,
        bool input_keep_aspect_ratio,
        bool input_use_multiclass_nms);
```

Parameters 3:

- input_data: std::vector<TensorPTRWithName>

Input parameter. Input data with three outputs or one output.

- ost_w: std::vector<int>

Input parameter. Width of original images.

- ost_h: std::vector<int>

Input parameter. Height of original images.

- dete_threshold: std::vector<std::vector<float>>

Input parameter. Detection threshold.

- nms_threshold: std::vector<float>

Input parameter. NMS threshold.

- input_keep_aspect_ratio: bool

Input parameter. Whether to keep aspect ratio in images.

- input_use_multiclass_nms: bool

Input parameter. Whether to use multiclass nms.

Interface 4:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(std::map
↪<std::string, Tensor&>& input_data,
        std::vector<int> &ost_w,
        std::vector<int> &ost_h,
        std::vector<std::vector<float>> &dete_threshold,
        std::vector<float> &nms_threshold,
        bool input_keep_aspect_ratio,
        bool input_use_multiclass_nms);
```

Parameters 4:

- input_data: std::map<std::string, Tensor&>

Input parameter. Input data with three outputs or one output.

- ost_w: std::vector<int>

Input parameter. Width of original images.

- ost_h: std::vector<int>

Input parameter. Height of original images.

- dete_threshold: std::vector<std::vector<float>>

Input parameter. Detection threshold.

- nms_threshold: std::vector<float>

Input parameter. NMS threshold.

- input_keep_aspect_ratio: bool

Input parameter. Whether to keep aspect ratio for boxes.

- input_use_multiclass_nms: bool

Input parameter. Whether to multiclass nms.

Returns:

std::vector<std::vector<std::tuple<left, top, right, bottom, class_id, score> > >

- left: int

The leftmost x-coordinate of the detection result.

- top: int

The topmost y-coordinate of the detection result.

- right: int

The rightmost x-coordinate of the detection result.

- bottom: int

The bottommost y-coordinate of the detection result.

- class_id: int

The class label of the detection result.

- score: float

The score of the detection result.

reset_anchors

Update the size of the anchor.

Interface:

```
int reset_anchors(std::vector<std::vector<std::vector<int>>> anchors_new);
```

Parameters:

- anchors_new: std::vector<std::vector<std::vector<int> > >

List of anchor sizes to be updated.

Returns:

A return value of 0 indicates success, while other values indicate failure.

algo_yolov5_post_cpu_opt_async

For accelerated post-processing interfaces in cpu, internally implemented using thread pools.

__init__

Interface:

```
algo_yolov5_post_cpu_opt_async(const std::vector<std::vector<int>>& shape,
                    int network_w=640,
                    int network_h=640,
                    int max_queue_size=20,
                    bool use_multiclass_nms=true);
```

Parameters:

- shape: std::vector<std::vector<int>>

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- use_multiclass_nms: bool

Whether to use multi-class NMS, which defaults to true.

push_data

Support input with arbitrary batchsize.

Interface1:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    std::vector<TensorPTRWithName> input_data,
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

Parameters1:

- channel_idx: std::vector<int>

The channel number of the input image.

- image_idx: std::vector<int>

The sequence number of the input image.

- input_data: std::vector<TensorPTRWithName>

The input data, including three outputs.

- dete_threshold: std::vector<float>

Detection threshold sequence.

- nms_threshold: std::vector<float>

nms threshold.

- ost_w: std::vector<int>

The width of original image.

- ost_h: std::vector<int>

The height of original image.

- padding_attrs: std::vector<std::vector<int>>

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns1:

Return 0 if successful, otherwise failed.

Interface2:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    std::vector<TensorPTRWithName> input_data,
    std::vector<std::vector<float>> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

Parameters2:

- channel_idx: std::vector<int>

The channel number of the input image.

- image_idx: std::vector<int>

The sequence number of the input image.

- input_data: std::vector<TensorPTRWithName>

The input data, including three outputs.

- dete_threshold: std::vector<std::vector<float>>

Detection threshold sequence.

- nms_threshold: std::vector<float>

nms threshold.

- ost_w: std::vector<int>

The width of original image.

- ost_h: std::vector<int>

The height of original image.

- padding_attrs: std::vector<std::vector<int>>

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns2:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

reset_anchors

Update anchor sizes

Interface:

```
int reset_anchors(std::vector<std::vector<std::vector<int>>> anchors_new);
```

Parameters:

- anchors_new: std::vector<std::vector<std::vector<int> > >

List of anchor sizes to update

Returns:

Return 0 if successful, otherwise failed.

tpu_kernel_api_yolov5_detect_out

For the 3-output yolov5 model, Tensor Computing Processor Kernel is used to accelerate post-processing. Currently, only BM1684x is supported, and the version of libsophon must be no less than 0.4.6 (v23.03.01).

Constructor

Interface:

```
tpu_kernel_api_yolov5_detect_out(int device_id,
                                 const std::vector<std::vector<int>>& shapes,
                                 int network_w=640,
                                 int network_h=640,
                                 std::string module_file = "/opt/sophon/libsophon-current/lib/
↪tpu_module/libbm1684x_kernel_module.so");
```

Paramters:

- device_id: int

Input parameter. The used device ID.

- shape: std::vector<std::vector<int> >

Input parameter. The shape of the input data.

- network_w: int

Input parameter. The input width of the model, default is 640.

- network_h: int

Input parameter. The input width of the model, default is 640.

- module_file: string

Input parameter. Kernel module file path, the default is "/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so" .

process

Processing interface

Interface 1:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(
                        std::vector<TensorPTRWithName>& input,
                        float dete_threshold,
                        float nms_threshold,
                        bool release_input = false);
```

Paramters1:

- input_data: std::vector<TensorPTRWithName>

Input parameter. Input data, including three outputs.

- dete_threshold: float

Input parameter. The detection threshold.

- nms_threshold: float

Input parameters. The nms threshold sequence.

- release_input: bool

Input parameters. Release input memory, default is false.

Interface 2:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(
                          std::map<std::string, Tensor&>& input,
                          float dete_threshold,
                          float nms_threshold,
                          bool release_input = false);
```

Paramters 2:

- input_data: std::map<std::string, Tensor&>

Input parameter. Input data, including three outputs.

- dete_threshold: float

Input parameter. The detection threshold.

- nms_threshold: float

Input parameters. The nms threshold sequence.

- release_input: bool

Input parameters. Release input memory, default is false.

Returns:

std::vector<std::vector<std::tuple<left, top, right, bottom, class_id, score> > >

- left: int

The leftmost x coordinate of the detection result.

- top: int

The uppermost y coordinate of the detection result.

- right: int

The rightmost x coordinate of the detection result.

- bottom: int

The lowest y coordinate of the detection result.

- class_id: int

The category ID of the test result.

- score: float

The score of the test result.

reset_anchors

Update anchor size.

Interface:

```
int reset_anchors(std::vector<std::vector<std::vector<int>>> anchors_new);
```

Paramters:

- anchors_new: std::vector<std::vector<std::vector<int> > >

The list of anchor sizes to update.

Returns:

Returns 0 on success, other values indicate failure.

tpu_kernel_api_yolov5_out_without_decode

For the 1-output yolov5 model, Tensor Computing Processor Kernel is used to accelerate post-processing. Currently, only BM1684x is supported, and the version of libsophon must be no less than 0.4.6 (v23.03.01).

Constructor

Interface:

```
tpu_kernel_api_yolov5_out_without_decode(int device_id,
                                const std::vector<int>& shapes,
                                int network_w=640,
                                int network_h=640,
                                std::string module_file = "/opt/sophon/libsophon-current/lib/
↪tpu_module/libbm1684x_kernel_module.so");
```

Paramters:

- device_id: int

Input parameter. The used device ID.

- shape: std::vector<int>

Input parameter. The shape of the input data.

- network_w: int

Input parameter. The input width of the model, default is 640.

- network_h: int

Input parameter. The input width of the model, default is 640.

- module_file: string

Input parameter. Kernel module file path, the default is "/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so" .

process

Processing interface

Interface 1:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(
                            TensorPTRWithName& input,
                            float dete_threshold,
                            float nms_threshold);
```

Paramters1:

- input_data: TensorPTRWithName

Input parameter. Input data, including one output.

- dete_threshold: float

Input parameter. The detection threshold.

- nms_threshold: float

Input parameters. The nms threshold sequence.

Interface 2:

```
std::vector<std::vector<std::tuple<int, int, int, int ,int, float>>> process(
                            std::map<std::string, Tensor&>& input,
                            float dete_threshold,
                            float nms_threshold);
```

Paramters 2:

- input_data: std::map<std::string, Tensor&>

Input parameter. Input data, including one output.

- dete_threshold: float

Input parameter. The detection threshold.

- nms_threshold: float

Input parameters. The nms threshold sequence.

Returns:

std::vector<std::vector<std::tuple<left, top, right, bottom, class_id, score> > >

- left: int

The leftmost x coordinate of the detection result.

- top: int

The uppermost y coordinate of the detection result.

- right: int

The rightmost x coordinate of the detection result.

- bottom: int

The lowest y coordinate of the detection result.

- class_id: int

The category ID of the test result.

- score: float

The score of the test result.

4.17.5 Yolox post-processing acceleration interfaces

algo_yolox_post

For post-processing interfaces with yolox model, internally implemented using thread pools.

Constructor

Interface:

```
algo_yolox_post(const std::vector<int>& shape,
                int network_w=640,
                int network_h=640,
                int max_queue_size=20);
```

Parameters:

- shape: std::vector<int>

Input parameters. The shape of the input data.

- network_w: int

Input parameters. The input width of the model, which defaults to 640.

- network_h: int

Input parameters. The input height of the model, which defaults to 640.

- max_queue_size: int

Input parameters. Maximum length of cached data.

push_data

Input data. The value of batchsize other than 1 is supported.

Interface:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    TensorPTRWithName input_data,
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

Parameters:

- channel_idx: std::vector<int>

Input parameters. The channel number of the input image.

- image_idx: std::vector<int>

Input parameters. The sequence number of the input image.

- input_data: TensorPTRWithName

Input parameters. The input data.

- dete_threshold: std::vector<float>

Input parameters. Detection threshold sequence.

- nms_threshold: std::vector<float>

Input parameters. nms threshold.

- ost_w: std::vector<int>

Input parameters. The width of original image.

- ost_h: std::vector<int>

Input parameters. The height of original image.

- padding_attrs: std::vector<std::vector<int> >

Input parameters. The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

4.17.6 Yolov8 post-processing acceleration interfaces

algo_yolov8_post_1output_async

For post-processing interfaces with a single output YOLOv8 model, internally implemented using thread pools.

__init__

Interface:

```
algo_yolov8_post_1output_async(const std::vector<int>& shape,
                    int network_w=640,
                    int network_h=640,
                    int max_queue_size=20,
                    bool input_use_multiclass_nms=true,
                    bool agnostic=false);
```

Parameters:

- shape: std::vector<int>

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- input_use_multiclass_nms: bool

Each detection box has multiple categories.

- agnostic: bool

Algorithm without class-specific NMS

push_data

Input data. The value of batchsize other than 1 is supported.

接口形式:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    TensorPTRWithName input_data,
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

参数说明:

- channel_idx: std::vector<int>

The channel number of the input image.

- image_idx: std::vector<int>

The sequence number of the input image.

- input_data: TensorPTRWithName

The input data.

- dete_threshold: std::vector<float>

Detection threshold sequence.

- nms_threshold: std::vector<float>

nms threshold.

- ost_w: std::vector<int>

The width of original image.

- ost_h: std::vector<int>

The height of original image.

- padding_attrs: std::vector<std::vector<int> >

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

algo_yolov8_post_cpu_opt_1output_async

For post-processing interfaces with a single output YOLOv8 model, internally implemented using thread pools.

__init__

Interface:

```
algo_yolov8_post_cpu_opt_1output_async(const std::vector<int>& shape,
                    int network_w=640,
                    int network_h=640,
                    int max_queue_size=20,
                    bool input_use_multiclass_nms=true,
                    bool agnostic=false);
```

Parameters:

- shape: std::vector<int>

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- input_use_multiclass_nms: bool

Each detection box has multiple categories.

- agnostic: bool

Algorithm without class-specific NMS

push_data

Input data. The value of batchsize other than 1 is supported.

接口形式:

```
int push_data(
    std::vector<int> channel_idx,
    std::vector<int> image_idx,
    TensorPTRWithName input_data,
    std::vector<float> dete_threshold,
    std::vector<float> nms_threshold,
    std::vector<int> ost_w,
    std::vector<int> ost_h,
    std::vector<std::vector<int>> padding_attr);
```

参数说明:

- channel_idx: std::vector<int>

The channel number of the input image.

- image_idx: std::vector<int>

The sequence number of the input image.

- input_data: TensorPTRWithName

The input data.

- dete_threshold: std::vector<float>

Detection threshold sequence.

- nms_threshold: std::vector<float>

nms threshold.

- ost_w: std::vector<int>

The width of original image.

- ost_h: std::vector<int>

The height of original image.

- padding_attrs: std::vector<std::vector<int> >

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
std::tuple<std::vector<std::tuple<int, int, int, int ,int, float>>,int,int> get_result_npy();
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

4.17.7 sort

sort_tracker_controller

SORT algorithm to track target

sort_tracker_controller

Interface:

```
sort_tracker_controller(float max_iou_distance = 0.7, int max_age = 30, int n_init = 3);
```

Parameters:

- max_iou_distance: float

Input. Maximum Intersection over Union (IoU) distance threshold used in the tracker.

- max_age: int

Input. The maximum number of frames that a tracked object can exist in the tracker.

- n_init: int

Input. The threshold for the number of initialization frames in the tracker.

process

Processing interface.

Interface:

```
td::vector<std::tuple<int, int, int, int, int, float, int>> process(const std::vector<std::tuple
↪<int, int, int, int ,int, float>>& detected_objects_short);
```

Parameters:

- detected_objects: std::vector<std::tuple<left, top, right, bottom, class_id, score>>

Input. Detected objects.

返回值说明:

- tracked_objects: std::vector<std::tuple<left, top, right, bottom, class_id, score, track_id>>

Output。Tracked objects.

sort_tracker_controller_async

SORT algorithm asynchronous processing interface

Constructor

interface:

```
sort_tracker_controller(float max_iou_distance = 0.7, int max_age = 30, int n_init = 3,
↪int input_queue_size = 10,int result_queue_size = 10);
```

Parameters:

- max_iou_distance: float

Input. Maximum Intersection over Union (IoU) distance threshold used in the tracker.

- max_age: int

Input. The maximum number of frames that a tracked object can exist in the tracker.

- n_init: int

Input. The threshold for the number of initialization frames in the tracker.

- input_queue_size: int

Input. Buffer size of the input queue.

- result_queue_size: int

Input. Buffer size of the result queue.

push_data

Asynchronous processing interface, pushes input parameters to the internal task queue, used in conjunction with get_result

interface :

```
int push_data(const std::vector<std::tuple<int, int, int, int ,int, float>>& detected_
↪objects_short);
```

Parameters :

- detected_objects_short: std::vector<std::tuple<left, top, right, bottom, class_id, score>>

Input parameter. Detected object frame.

Returns :

int

Returns 0 on success and others on failure.

get_result

Asynchronous processing interface, obtains the information of the target to be tracked, and is used in conjunction with push_data.

Interface:

```
std::vector<std::tuple<int, int, int, int, int, float, int>> get_result();
```

Parameters:

- vector<std::tuple<left, top, right, bottom, class_id, score, track_id>>

Output parameters. The object being tracked.

4.17.8 deepsort

deepsort_tracker_controller

针对DeepSORT算法，通过处理检测的结果和提取的特征，实现对目标的跟踪。

构造函数

接口形式:

```
deepsort_tracker_controller(float max_cosine_distance,
                            int nn_budget,
                            int k_feature_dim,
                            float max_iou_distance = 0.7,
                            int max_age = 30,
                            int n_init = 3);
```

参数说明:

- max_cosine_distance: float

输入参数。用于相似度计算的最大余弦距离阈值。

- nn_budget: int

输入参数。用于最近邻搜索的最大数量限制。

- k_feature_dim: int

输入参数。被检测的目标的特征维度。

- max_iou_distance: float

输入参数。模用于跟踪器中的最大交并比（IoU）距离阈值。

- max_age: int

输入参数。跟踪目标在跟踪器中存在的最大帧数。

- n_init: int

输入参数。跟踪器中的初始化帧数阈值。

process

处理接口。

接口形式1:

```
int process(const vector<DeteObjRect>& detected_objects,
        vector<Tensor>& feature,
        vector<TrackObjRect>& tracked_objects);
```

参数说明1:

- detected_objects: vector<DeteObjRect>

输入参数。检测出的物体框。

- feature: vector<Tensor>

输入参数。检测出的物体的特征。

- tracked_objects: vector<TrackObjRect>

输出参数。被跟踪的物体。

返回值说明:

int

成功返回0，失败返回其他。

deepsort_tracker_controller_async

DeepSORT algorithm asynchronous processing interface

Constructor

interface:

```
deepsort_tracker_controller(float max_cosine_distance,
                            int nn_budget,
                            int k_feature_dim,
                            float max_iou_distance = 0.7,
                            int max_age = 30,
                            int n_init = 3,
                            int queue_size = 10);
```

Parameters:

- max_cosine_distance: float

Input parameter. As the maximum cosine distance threshold when calculating similarity.

- nn_budget: int

Input parameter. Maximum number limit used for nearest neighbor searches.

- k_feature_dim: int

Input parameter. Feature dimensions of the detected target.

- max_iou_distance: float

Input parameter. The IoU distance threshold used in trackers.

- max_age: int

Input parameter. The maximum number of frames that the tracking target will exist in the tracker.

- n_init: int

Input parameter. Initialization frame threshold in the tracker.

- queue_size: int

Input parameter. The length of the result buffer queue.

push_data

Asynchronous processing interface, pushes input parameters to the internal task queue, used in conjunction with get_result

interface 1:

```
int push_data(const vector<DeteObjRect>& detected_objects,
              vector<Tensor>& feature);
```

Parameters 1:

- detected_objects: vector<DeteObjRect>

Input parameter. Detected object frame.

- feature: vector<Tensor>

Input parameter. The characteristics of detected objects.

Returns 1:

int

Returns 0 on success and others on failure.

interface 2:

```
int push_data(const vector<DeteObjRect>& detected_objects,
              vector<vector<float>>& feature);
```

Parameters 2:

- detected_objects: vector<DeteObjRect>

Input parameter. The detected object frame.

- feature: vector<float>

Input parameter.The characteristics of detected objects.

Return 2:

int

Returns 0 on success and others on failure.

get_result

Asynchronous processing interface, obtains the information of the target to be tracked, and is used in conjunction with push_data.

Interface:

```
vector<TrackObjRect> get_result();
```

Parameters:

- vector<TrackObjRect>

Output parameters. The object being tracked.

4.17.9 bytetrack

bytetrack_tracker_controller

For the ByteTrack algorithm, tracking of targets is achieved by processing the detection results.

Constructor

Interface:

```
bytetrack_tracker_controller(int frame_rate = 30,
                             int track_buffer = 30);
```

Parameters:

- frame_rate: int

Input. Used to control the maximum number of frames allowed to disappear for tracked objects.

- track_buffer: int

Input. Used to control the maximum number of frames allowed to disappear for tracked objects.

process

Processing interface.

Interface:

```
int process(const vector<DeteObjRect>& detected_objects,
        vector<TrackObjRect>& tracked_objects);
```

Parameters:

- detected_objects: vector<DeteObjRect>

Input. Detected objects.

- tracked_objects: vector<TrackObjRect>

Output. Tracked objects.

Returns:

int

Returns 0 on success and others on failure.

4.17.10 openpose

openpose 使用tpukernel对part nms后处理加速

tpu_kernel_api_openpose_part_nms

For the OpenPose model, Tensor Computing Processor Kernel is used to accelerate part nms post-processing. Currently, only BM1684x is supported, and the version of libsophon must be no less than 0.4.6 (v23.03.01).

Constructor

Interface:

```
tpu_kernel_api_openpose_part_nms(int device_id,
                                 int network_c,
                                 std::string module_file = "/opt/sophon/libsophon-current/lib/
↪tpu_module/libbm1684x_kernel_module.so");
```

Paramters:

- device_id: int

Input parameter. The used device ID.

- network_c: int

Input parameter. The input channel of the model, corresponding to the number of keypoint channels.

- module_file: string

Input parameter. Kernel module file path, the default is "/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so" .

process

Processing interface

Interface 1:

```
std::tuple<std::vector<std::vector<int>>, std::vector<std::vector<float>>, std::vector
↪<std::vector<int>>> process(
                        TensorPTRWithName& input_data,
                        std::vector<int>& shape,
                        std::vector<float>& threshold,
                        std::vector<int>& max_peak_num);
```

Paramters1:

- input_data: TensorPTRWithName

Input parameter. Input data.

- shape: std::vector<int>

Input. Input data width and height.

- threshold: std::vector<float>

Input. Detection threshold.

- max_peak_num: std::vector<int>

Input. The max number of detected peaks.

Interface 2:

```
std::tuple<std::vector<std::vector<int>>, std::vector<std::vector<float>>, std::vector
↪<std::vector<int>>> process(
                          std::map<std::string, Tensor&>& input_data,
                          std::vector<int>& shape,
                          std::vector<float>& threshold,
                          std::vector<int>& max_peak_num);
```

Paramters 2:

- input_data: std::map<std::string, Tensor&>

Input parameter. Input data.

- shape: std::vector<int>

Input. Input data width and height.

- threshold: std::vector<float>

Input. Detection threshold.

- max_peak_num: std::vector<int>

Input. The max number of detected peaks.

Returns:

std::tuple<std::vector<std::vector<int>>, std::vector<std::vector<float>>, std::vector<std::vector<int>>>

- 1st output: std::vector<std::vector<int>>

The number of the detected peaks in each channel.

- 2nd: std::vector<std::vector<float>>

The scores of all detected peaks.

- 3rd: std::vector<std::vector<int>>

The flatten coordinates of all detected peaks.

reset_network_c

Update the channel number of the input.

Interface:

```
int reset_network_c(int network_c_new);
```

Paramters:

- network_c_new: int

The number of channels to be updated.

Returns:

Returns 0 on success, other values indicate failure.

CHAPTER 5

SAIL Python API

5.1 Basic function

5.1.1 get_available_tpu_num

Get the number of available Tensor Computing Processors.

Interface:

```
def get_available_tpu_num()->int
```

Returns:

- tpu_num : int

Number of available Tensor Computing Processors

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    tpu_len = sail.get_available_tpu_num()
    print('available tpu:',tpu_len)
```

5.1.2 set_print_flag

Print main process time use.

Interface:

```
def set_print_flag(print_flag: bool)
```

Parameters:

- print_flag : bool

If print_flag is true, print main process time use, Otherwise not print.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    set_print_flag(True)
```

5.1.3 set_dump_io_flag

Dump input date and output date.

Interface:

```
def set_dump_io_flag(dump_io_flag: bool)
```

Parameters:

- dump_io_flag : bool

If dump_io_flag is true, dump input data and output data, Otherwise not dump.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    set_dump_io_flag(True)
```

5.1.4 set_loglevel

Set the logging level to the specified level. Lower log levels are typically used in production environments to reduce performance overhead and the volume of log data, while higher log levels are suitable for development and debugging in order to capture more detailed information.

Interface:

```
def set_loglevel(sail.LogLevel loglevel) -> int
```

Parameters:

- loglevel: LogLevel

The Target log level as a sail.LogLevel enum value. The optional values include TRACE, DEBUG, INFO, WARN, ERR, CRITICAL, OFF, and the default level is INFO.

Returns:

return: int

Returning 0 indicates the log level was set successfully, whereas returning -1 indicates a failure due to an unknown log level.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    ret = sail.set_loglevel(sail.LogLevel.TRACE)
    if (ret == 0):
        print("Set log level successfully")
    else:
        print("Unknown log level, set failed.")
```

5.1.5 set_decoder_env

Set the parameters for the Decoder (including MutlDecoder) through environment variables. These must be set before the Decoder is constructed; otherwise, the default values will be used. This is mainly applicable to video decoding.

Interface:

```
def set_decoder_env(env_name: str, env_value: str)
```

Parameters:

- env_name: str

The property name to set for the Decoder. The available property names are as follows:

- 'rtsp_transport' : The transport protocol used for RTSP. The default is TCP.
- 'extra_frame_buffer_num' : The maximum number of cached frames for the Decoder. The default is 5.
- 'stimeout' : Raise error timeout, in milliseconds. The default is 20000000, i.e., 20 seconds.
- 'skip_non_idr' : Decoding frame skip mode. 0, no skip; 1, skip Non-RAP frames; 2, skip non-reference frames. The default is 0.
- 'fflags' : format flags, like "nobuffer" . Read ffmpeg official docs for more details.
- 'rtsp_flags' : Set RTSP flags. The default is prefer_tcp.
- 'refcounted_frames' : When set to 1, the decoded images need to be manually released by the program; when set to 0, they are automatically released by the Decoder.

- 'probesize' : the max size of the data to analyze to get stream information. 5000000 by default.
- 'analyzeduration' : How many microseconds are analyzed to probe the input. 5000000 microseconds by default.
- 'buffer_size' : The maximum socket buffer size in bytes.
- 'max_delay' : Maximum demuxing delay in microseconds.
- env_value: str

Environment value.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    sail.set_decoder_env("extra_frame_buffer_num", "3") # Decrease buffer num for lower memory usage
    sail.set_decoder_env("probesize", "1024") # Decrease probesize for lower latency
    sail.set_decoder_env("skip_non_idr", "2") # skip non-reference frames
    dev_id = 0
    handle = sail.Handle(dev_id)
    image_name = "your_video.mp4"
    decoder = sail.Decoder(image_name, True, dev_id)
    bmimg = decoder.read(handle)
```

5.1.6 base64_encode

Encode byte data into base64 and return the encoded data as bytes. BM1688 and CV186AH PCIE mode are not supported.

Interface:

```
def base64_encode(handle: Handle, input_bytes: bytes) -> bytes:
```

Parameters:

- handle: Handle

The handle of the device, created using sail.Handle(dev_id).

- input_bytes: bytes

The byte data to be encoded.

Returns:

- bytes

The byte data encoded in base64.

Sample

```
import sophon.sail as sail
import numpy as np

if __name__ == "__main__":
    arr = np.array([[1, 2, 3], [4, 5, 6]])
    # to bytes
    arr_bytes = arr.tobytes()
    # get handle
    handle = sail.Handle(0)
    # base64 encode
    base64_encoded_arr = sail.base64_encode(handle,arr_bytes)
```

5.1.7 base64_decode

Decode the byte-encoded data in base64 and return the decoded byte data. BM1688 and CV186AH PCIE mode are not supported.

Interface:

```
def base64_decode(handle: Handle, encode_bytes: bytes) -> bytes:
```

Parameters:

- handle: Handle

The handle of the device, created using sail.Handle(dev_id).

- encode_bytes: bytes

The byte-encoded data in base64.

Returns:

- bytes

The byte data decoded from base64.

Sample

```
import sophon.sail as sail
import numpy as np

if __name__ == "__main__":
    arr = np.array([[1, 2, 3], [4, 5, 6]])
    shape = arr.shape
    # tobytes
    arr_bytes = arr.tobytes()
    # get handle
    handle = sail.Handle(0)
    # base64 encode
    base64_encoded_arr = sail.base64_encode(handle,arr_bytes)
```

(continues on next page)

(continued from previous page)

```
# decode
base64_decode_arr = sail.base64_decode(handle,base64_encoded_arr)
# byte to numpy
res_arr = np.frombuffer(arr_bytes, dtype=np.int64).reshape(shape)
```

5.1.8 base64_encode_array

Encode a numpy.array into base64, generating byte-encoded data. BM1688 and CV186AH PCIE mode are not supported.

Interface:

```
def base64_encode_array(handle: Handle, input_arr: numpy.ndarray) -> bytes:
```

Parameters:

- handle: Handle

The handle of the device, created using sail.Handle(dev_id).

- input_arr: numpy.ndarray

The numpy.ndarray data to be encoded.

Returns

- bytes

The byte data encoded in base64.

5.1.9 base64_decode_asarray

Decode base64 to generate numpy.array data.

Interface:

```
def base64_decode_asarray(handle: Handle, encode_arr_bytes: bytes, array_type:str =
↪"uint8") -> numpy.ndarray:
```

Parameters:

- handle: Handle

The handle of the device, created using sail.Handle(dev_id).

- encode_arr_bytes: bytes

The byte data of the numpy.ndarray encoded in base64.

- array_type: str

The data type of numpy.ndarray, defaulting to uint8, supports float, uint8, int8, int16, int32, int64.

Returns

- numpy.array

The one-dimensional numpy.array array decoded from base64.

Sample

```
import sophon.sail as sail
import numpy as np

if __name__ == "__main__":
    arr = np.array([[1,2,3],[4,5,6]],dtype=np.uint8)
    # base64 encode
    base64_encoded = sail.base64_encode_array(handle,arr)
    # base64 decode
    res_array = sail.base64_decode_asarray(handle,base64_encoded).reshape(shape)
```

5.1.10 get_tpu_util

Get the processor utilization of the specified device

Interface:

```
def get_tpu_util(dev_id: int) -> int
```

Parameter:

- dev_id: int

Device ID.

Return:

Returns the processor percent utilization of the device corresponding to the ID.

Sample

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("dev {} tpu-util is {} %".format(dev_id,sail.get_tpu_util(dev_id)))
```

5.1.11 get_vpu_util

Get the VPU utilization of the specified device

Interface:

```
def get_vpu_util(dev_id: int) -> list
```

Parameter:

- dev_id: int

Device ID.

Return:

The vpu of bm1684 is 5-core, and the return value is a list of length 5. The vpu of bm1684x is 3-core, and the return value is a list of length 3. Each integer in the List is the percent utilization of the corresponding core.

Sample

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_vpu_util",sail.get_vpu_util(dev_id))
```

5.1.12 get_vpp_util

Get the VPP utilization of the specified device

Interface:

```
def get_vpp_util(dev_id: int) -> list
```

Parameter:

- dev_id: int

Device ID.

Return:

The vpp of bm1684 and bm1684x are both 2-core, and the return value is a list of length 2. Each integer in the List is the percent utilization of the corresponding core.

Sample

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_vpp_util",sail.get_vpp_util(dev_id))
```

5.1.13 get_board_temp

Get the temperature of the board.

Interface:

```
def get_board_temp(dev_id: int) -> int
```

Parameter:

- dev_id: int

Device ID.

Return:

The board temperature for the corresponding card, with the default unit in Celsius (°C)

Sample

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_board_temp",sail.get_board_temp(dev_id))
```

5.1.14 get_chip_temp

Get the temperature of the processor.

Interface:

```
def get_chip_temp(dev_id: int) -> int
```

Parameter:

- dev_id: int

Device ID.

Return:

The processor temperature for the corresponding card, with the default unit in Celsius (°C)

Sample

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_chip_temp",sail.get_chip_temp(dev_id))
```

5.1.15 get_dev_stat

Get device memory information.

Interface:

```
def get_dev_stat(dev_id: int) -> list
```

Parameter:

- dev_id: int

Device ID.

Return:

A list of memory information for the corresponding device: [mem_total, mem_used, tpu_util].

Sample

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    print("get_dev_stat",sail.get_dev_stat(dev_id))
```

5.2 SAIL enum type

5.2.1 sail.Data type

Interface:

```
# Data type for float32
sail.Dtype.BM_FLOAT32
# Data type for int8
sail.Dtype.BM_INT8
# Data type for uint8
sail.Dtype.BM_UINT8
# Data type for int32
sail.Dtype.BM_INT32
# Data type for uint32
sail.Dtype.BM_UINT32
```

5.2.2 sail.Format

接口形式:

```
sail.Format.FORMAT_YUV420P
sail.Format.FORMAT_YUV422P
sail.Format.FORMAT_YUV444P
sail.Format.FORMAT_NV12
sail.Format.FORMAT_NV21
sail.Format.FORMAT_NV16
sail.Format.FORMAT_NV61
sail.Format.FORMAT_NV24
sail.Format.FORMAT_RGB_PLANAR
sail.Format.FORMAT_BGR_PLANAR
sail.Format.FORMAT_RGB_PACKED
sail.Format.FORMAT_BGR_PACKED
sail.Format.FORMAT_RGBP_SEPARATE
sail.Format.FORMAT_BGRP_SEPARATE
sail.Format.FORMAT_GRAY
sail.Format.FORMAT_COMPRESSED
```

5.2.3 sail.ImgDtype

接口形式:

```
sail.ImgDtype.DATA_TYPE_EXT_FLOAT32
sail.ImgDtype.DATA_TYPE_EXT_1N_BYTE
sail.ImgDtype.DATA_TYPE_EXT_4N_BYTE
sail.ImgDtype.DATA_TYPE_EXT_1N_BYTE_SIGNED
sail.ImgDtype.DATA_TYPE_EXT_4N_BYTE_SIGNED
```

5.2.4 sail.IOMode

接口形式:

```
# Input tensors are in system memory while output tensors are in device memory
sail.IOMode.SYSI
# Input tensors are in device memory while output tensors are in system memory.
sail.IOMode.SYSO
# Both input and output tensors are in system memory.
sail.IOMode.SYSIO
# Both input and output tensors are in device memory.
sail.IOMode.DEVIO
```

5.2.5 sail.bmcv_resize_algorithm

The resize method of bmcv.

Interface:

```
sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST
sail.bmcv_resize_algorithm.BMCV_INTER_LINEAR
sail.bmcv_resize_algorithm.BMCV_INTER_BICUBIC
```

Parameters:

- sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST

Using nearest neighbor interpolation method while resizing.

- sail.bmcv_resize_algorithm.BMCV_INTER_LINEAR

Using linear interpolation method while resizing.

- sail.bmcv_resize_algorithm.BMCV_INTER_BICUBIC

Using double cubic interpolation method while resizing.

5.2.6 sail.sail_resize_type

图像预处理对应的预处理方法。

接口形式: .. code-block:: python

sail_resize_type.BM_RESIZE_VPP_NEAREST sail_resize_type.BM_RESIZE_TPU_NEAREST
sail_resize_type.BM_RESIZE_TPU_LINEAR sail_resize_type.BM_RESIZE_TPU_BICUBIC
sail_resize_type.BM_PADDING_VPP_NEAREST
sail_resize_type.BM_PADDING_TPU_NEAREST
sail_resize_type.BM_PADDING_TPU_LINEAR sail_resize_type.BM_PADDING_TPU_BICUBI

参数说明:

- BM_RESIZE_VPP_NEAREST

Use VPP, the nearest neighbor method for image scaling

- BM_RESIZE_TPU_NEAREST

Use Tensor Computing Processor, the nearest neighbor method for image scaling

- BM_RESIZE_TPU_LINEAR

Use Tensor Computing Processor, the linear interpolation method for image scaling.

- BM_RESIZE_TPU_BICUBIC

Use Tensor Computing Processor, double cubic interpolation method for image scaling

- BM_PADDING_VPP_NEAREST

Use VPP, the nearest neighbor method for image scaling with padding

- BM_PADDING_TPU_NEAREST

Use Tensor Computing Processor, the nearest neighbor method for image scaling with padding

- BM_PADDING_TPU_LINEAR

Use Tensor Computing Processor, the linear interpolation method for image scaling with padding

- BM_PADDING_TPU_BICUBIC

Use Tensor Computing Processor, double cubic interpolation method for image scaling with padding

5.3 sail.Handle

5.3.1 __init__

Interface:

```
def __init__(self, tpu_id: int)
```

Parameters

- tpu_id : int

create handle with tpu Id

5.3.2 get_device_id

Get Tensor Computing Processor id of this handle.

Interface:

```
def get_device_id(self)-> int
```

Returns

- tpu_id : int

Tensor Computing Processor id of this handle.

5.3.3 get_sn

Get serial number of this handle.

Interface:

```
def get_sn(self)-> str
```

Returns

- serial_number : str

serial number of this handle.

5.3.4 get_target

Get Tensor Computing Processor type of this handle.

Interface:

```
def get_target(self)-> str
```

Returns

- tpu_chip_type : str

Tensor Computing Processor type of this handle.

5.4 sail.Tensor

5.4.1 __init__

Constructor allocates memory of the tensor. If synchronization between system memory and device memory is required, sync_d2s or sync_s2d needs to be executed.

Interface:

```
def __init__(self,
             handle: sail.Handle,
             data: numpy.array,
             own_sys_data: bool=True,
             own_dev_data: bool=True)
```

Parameters

- handle : sail.Handle

Handle instance

- array_data : numpy.array

Tensor ndarray data, dtype can be np.float32, np.int8 or np.uint8

- own_sys_data: bool, default: True

  Indicates whether the Tensor owns system memory.

  - If True, the Tensor allocates system memory and copies the input data into it.
  - If False, the Tensor does not allocate system memory.

- own_dev_data: bool, default: True

  Indicates whether the Tensor owns device memory.

  - If True, the Tensor allocates device memory (regardless of own_sys_data value)
  - If False, the Tensor does not allocate device memory
  - Data copying to device memory occurs **only when** both:
    - own_sys_data=False
    - own_dev_data=True

Interface:

```
def __init__(self,
             handle: sail.Handle,
             shape: list[int],
             dtype: sail.Dtype,
             own_sys_data: bool,
             own_dev_data: bool)
```

Parameters

- handle : sail.Handle

Handle instance

- shape : tuple

Tensor shape

- dytpe : sail.Dtype

Data type

- own_sys_data : bool

Indicator of whether own system memory

- own_dev_data : bool

Indicator of whether own device memory

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
```

(continues on next page)

(continued from previous page)

```
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor2 = sail.Tensor(handle,[1,2],sail.Dtype.BM_FLOAT32,true,true)
```

Interface:

This initialization method creates a new Tensor based on an existing source Tensor and reuses a portion of the source Tensor' s device memory without copying device memory. It is suitable for scenarios such as LLM inference where memory reuse is required.

During the use of this Tensor, it is necessary to ensure that the source Tensor is not released.

```
def __init__(self, src: Tensor, shape: list[int], offset: int)
```

Parameters:

- src: sail.Tensor

The source Tensor used to create the new Tensor.

- shape: list[int]

The shape of the new Tensor, a sequence of integers.

The number of elements corresponding to the new shape must not exceed the number of elements in the source Tensor.

- offset: int

The offset of the Tensor' s device memory relative to the source Tensor' s device memory, in bytes of the dtype.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    height = 1080
    width = 1920
    data_type = sail.Dtype.BM_INT32
    src_shape = [1, 3, height, width]
    src_tensor = sail.Tensor(handle, src_shape, data_type, False, True)

    dst_shape = [1, 1, height, width]
    offset = height * width
    dst_tensor = sail.Tensor(src_tensor, dst_shape, offset)
```

5.4.2 shape

Get shape of the tensor.

Interface:

```
def shape(self)-> list
```

Returns

- tensor_shape : list

Shape of the tensor

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    print(input_tensor1.shape())
```

5.4.3 dtype

Get data_type of the tensor.

Interface:

```
def dtype(self)-> sail.Dtype
```

Returns

- data_type : sail.Dtype

return data_type of the tensor.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    print(input_tensor1.dtype())
```

5.4.4 asnumpy

Get system data of the tensor. If synchronization between system memory and device memory is required, sync_d2s needs to be executed.

Interface:

```
def asnumpy(self)-> numpy.array
```

Returns

- data : numpy.array

System data of the tensor, dtype can be np.float32, np.int8 or np.uint8 with respective to the dtype of the tensor.

Interface:

```
def asnumpy(self, shape: tuple)-> numpy.array
```

Parameters

- shape : tuple

Tensor shape want to get

Returns

- data : numpy.array

System data of the tensor, dtype can be np.float32, np.int8 or np.uint8 with respective to the dtype of the tensor.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_ = input_tensor1.asnumpy()
    input__ = input_tensor1.asnumpy((3,1))
```

5.4.5 update_data

Update system data of the tensor, if there is no system memory assigned, update the device memory.

Interface:

```
def update_data(self, data: numpy.array)
```

Parameters

data : numpy.array

Data to update. The data type of the updated data should be the same as the tensor, The data size should not exceed the tensor size, and the tensor shape will not be changed.

Note: If the data is of the numpy.float16 type, you should use numpy.view(numpy.uint16) and pass it to this API.

example:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)

    tensor_fp32 = sail.Tensor(handle, [1,3,640,640], sail.BM_FLOAT32, True, True)
    np_fp32 = np.ones(tensor_fp32.shape(),dtype=np.float32)
    tensor_fp32.update_data(np_fp32)

    tensor_fp16 = sail.Tensor(handle, [1,3,640,640], sail.BM_FLOAT16, True, True)
    np_fp16 = np.ones(tensor_fp16.shape(),dtype=np.float16)
    tensor_fp16.update_data(np_fp16.view(np.uint16))
```

5.4.6 scale_from

Scale data to tensor in system memory.

Interface:

```
def scale_from(self, data: numpy.array, scale: float32)
```

Parameters

- data : numpy.array with dtype of float32

Data.

- scale : float32

Scale value.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor1.scale_from(input,0.1)
```

5.4.7 scale_to

Scale tensor to data in system memory.

Interface:

```
def scale_to(self, scale: float32)-> numpy.array
```

Parameters

- scale : float32

Scale value.

Returns

- data : numpy.array with dtype of float32

Data.

Interface:

```
def scale_to(self, scale: float32, shape: tuple)-> numpy.array
```

Parameters

- scale : float32

Scale value.

- shape : tuple

Tensor shape wanted to get

Returns

- data : numpy.array with dtype of float32

Data.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor1_ = input_tensor1.scale_to(0.1)
    input_tensor1__ = input_tensor1.scale_to(0.1,(3,1))
```

5.4.8 reshape

Reset shape of the tensor.

Interface:

```
def reshape(self, shape: list)
```

Parameters

- shape : list

New shape of the tensor

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor1_ = input_tensor1.reshape([3,1])
```

5.4.9 own_sys_dat

Judge if the tensor owns data pointer in system memory.

Interface:

```
def own_sys_data(self)-> bool
```

Returns

- judge_ret : bool

True for owns data pointer in system memory.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    print(input_tensor1.own_sys_data())
```

5.4.10 own_dev_data

Judge if the tensor owns data in device memory.

Interface:

```
def own_dev_data(self)-> bool
```

Returns

- judge_ret : bool

True for owns data in device memory.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    print(input_tensor1.own_dev_data())
```

5.4.11 sync_s2d

Copy data from system memory to device memory without or with specified size.

Interface:

```
def sync_s2d(self)
```

Interface:

```
def sync_s2d(self, size: int)
```

Parameters

- size : int

Byte size to be copied

Interface:

```
def sync_s2d(self, src: sail.Tensor, offset_src: int, offset_dst: int, len: int)->None
```

Parameters

- src: sail.Tensor

Specifies the Tensor to be copied from.

- offset_src: int

Specifies the number of elements to offset in the source Tensor from where to start copying.

- offset_dst: int

Specifies the number of elements to offset in the destination Tensor from where to start copying.

- len: int

Specifies the length of the copy, i.e., the number of elements to copy.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input = np.array([1, 2, 3])

    input_tensor1 = sail.Tensor(handle,input)
    input_tensor2 = sail.Tensor(handle,[1,2], sail.Dtype.BM_FLOAT32, True, True)
    input_tensor2.sync_s2d()
    input_tensor2.sync_s2d(1)
    input_tensor2.sync_s2d(input_tensor1,0,0,2)
```

5.4.12 sync_d2s

Copy data from device memory to system memory without or with specified size.

Interface:

```
def sync_d2s(self)
```

Interface:

```
def sync_d2s(self, size: int)
```

Parameters

- size : int

Byte size to be copied

Interface:

```
def sync_d2s(self, src: sail.Tensor, offset_src: int, offset_dst: int, len: int)->None
```

Parameters

- src: sail.Tensor

Specifies the Tensor to be copied from.

- offset_src: int

Specifies the number of elements to offset in the source Tensor from where to start copying.

- offset_dst: int

Specifies the number of elements to offset in the destination Tensor from where to start copying.

- len: int

Specifies the length of the copy, i.e., the number of elements to copy.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)

    input_tensor1 = sail.Tensor(handle,[1,3],sail.Dtype.BM_FLOAT32,False,True)
    input_tensor2 = sail.Tensor(handle,[1,3],sail.Dtype.BM_FLOAT32,True,True)

    input_tensor1.ones()
    input_tensor2.sync_d2s()
    input_tensor2.sync_d2s(2)
    input_tensor2.sync_d2s(input_tensor1,0,0,2)
```

5.4.13 sync_d2d

Copies the data from another Tensor’ s device memory to this Tensor’ s device memory.

Interface:

```
def sync_d2d(self, src: sail.Tensor, offset_src: int, offset_dst: int, len: int)->None
```

Parameters

- src: sail.Tensor

Specifies the Tensor to be copied from.

- offset_src: int

Specifies the number of elements to offset in the source Tensor from where to start copying.

- offset_dst: int

Specifies the number of elements to offset in the destination Tensor from where to start copying.

- len: int

Specifies the length of the copy, i.e., the number of elements to copy.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    handle_ = sail.Handle(1)
    input_tensor1 = sail.Tensor(handle,[1,3],sail.Dtype.BM_FLOAT32,False,True)
    input_tensor2 = sail.Tensor(handle_,[1,3],sail.Dtype.BM_FLOAT32,True,True)

    input_tensor1.ones()
    input_tensor2.sync_d2d(input_tensor1,0,0,2)
```

5.4.14 sync_d2d_stride

Copies the data from another Tensor' s device memory to this Tensor' s device memory in stride.

Interface:

```
def sync_d2d_stride(self, src: sail.Tensor, stride_src: int, count: int)->None
```

Parameters:

- src: sail.Tensor

Specifies the Tensor to be copied from.

- stride_src: int

Specifies the stride of the source Tensor.

- stride_dst: int

Specifies the stride of the destination Tensor.stride_dst must be 1, EXCEPT: stride_dst == 4 && stride_src == 1 && Tensor_type_size == 1

- count: int

Specifies the count of elements to copy.Ensure count * stride_src <= tensor_src_size, count * stride_dst <= tensor_dst_size.

5.4.15 dump_data

Dump Tensor data to file. If synchronization between system memory and device memory is required, sync_d2s needs to be executed.

Interface:

```
def sync_d2s(self, file_name:str, bin:bool = False)
```

Parameters

- file_name : str

file path to dump tensor

- bin : bool

binary format, default False.

Sample:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    data = np.ones([1,20], dtype=int)
    ts = sail.Tensor(handle, data)
    ts.scale_from(data, 0.1)
    ts.dump_data("./temp.txt")
    ret_data = np.loadtxt("./temp.txt")
    print(ts.asnumpy(), ret_data)
```

5.4.16 memory_set

Fill memory with a scalar, it will be automatically converted to tensor' s dtype.

Interface:

```
def memory_set(self, c: any)->None
```

Parameters:

- c: any

the value to fill.

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    handle = sail.Handle(0)
    input = 1
    input_tensor1 = sail.Tensor(handle,[1],sail.Dtype.BM_FLOAT32,True,True)

    input_tensor1.memory_set(input)
```

5.4.17 zeros

fill memory with zeros.

Interface:

```
def zeros(self)->None
```

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input_tensor1 = sail.Tensor(handle,(1,3),sail.Dtype.BM_FLOAT32,False,True)

    input_tensor1.zeros()
```

5.4.18 ones

fill memory with ones.

Interface:

```
def ones(self)->None
```

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input_tensor1 = sail.Tensor(handle,(1,3),sail.Dtype.BM_FLOAT32,False,True)

    input_tensor1.ones()
```


5.4.19 size

Return the number of elements contained in the Tensor.

Interface:

```
def size(self)->int
```

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input_tensor1 = sail.Tensor(handle,(1,3),sail.Dtype.BM_FLOAT32,False,True)

    print(input_tensor1.size())
```

5.4.20 element_size

Returns the size in bytes of an individual element.

Interface:

```
def element_size(self)->int
```

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
    handle = sail.Handle(0)
    input_tensor1 = sail.Tensor(handle,(1,3),sail.Dtype.BM_FLOAT32,False,True)

    print(input_tensor1.element_size())
```

5.4.21 nbytes

Return the total number of bytes occupied by all elements of Tensor.

Interface:

```
def nbytes(self)->int
```

Sample:

```
import sophon.sail as sail
import numpy as np
if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
handle = sail.Handle(0)
input_tensor1 = sail.Tensor(handle,(1,3),sail.Dtype.BM_FLOAT32,False,True)

print(input_tensor1.nbytes())
```

5.5 sail.PaddingAtrr

5.5.1 __init__

Interface:

```
def __init__(self):
```

Interface:

```
def __init__(self,
             stx: int,
             sty: int,
             width: int,
             height: int,
             r: int,
             g: int,
             b: int)
```

Parameters

- stx : int

Offset x information relative to the origin of dst image

- sty : int

Offset y information relative to the origin of dst image

- width : int

The width after resize

- height : int

The height after resize

- r : int

Pixel value information of R channel

- g : int

Pixel value information of G channel

- b : int

Pixel value information of B channel

5.5.2 set_stx

set offset stx.

Interface:

```
def set_stx(self, stx: int)
```

Parameters

- stx : int

Offset x information relative to the origin of dst image

5.5.3 set_sty

set offset sty.

Interface:

```
def set_sty(self, sty: int)
```

Parameters

- sty : int

Offset y information relative to the origin of dst image

5.5.4 set_w

set width.

Interface:

```
def set_w(self, width: int)
```

Parameters

- width : int

The width after resize

5.5.5 set_h

set height.

Interface:

```
def set_h(self, height: int)
```

Parameters

- height : int

The height after resize

5.5.6 set_r

set R.

Interface:

```
def set_r(self, r: int)
```

Parameters

- r : int

Pixel value information of R channel

5.5.7 set_g

set G.

Interface:

```
def set_g(self, g: int):
```

Parameters

- g : int

Pixel value information of G channel

5.5.8 set_b

set B

Interface:

```
def set_b(self, b: int)
```

Parameters

b : int

Pixel value information of B channel

5.6 sail.Engine

5.6.1 __init__

Constructor without or with bmodel loaded.

Interface:

```
def __init__(tpu_id: int)
```

Parameters

- tpu_id : int

Tensor Computing Processor ID. You can use bm-smi to see available IDs

Interface:

```
def __init__(self, handle: Handle)
```

Parameters

- hanle : Handle

A Handle instance

Interface:

```
def __init__(self,
             bmodel_path: str,
             tpu_id: int,
             mode: sail.IOMode)
```

Parameters

- bmodel_path : str

Path to bmodel

- tpu_id : int

Tensor Computing Processor ID. You can use bm-smi to see available IDs

- mode : sail.IOMode

Specify the input/output tensors are in system memory or device memory

Interface:

```
def __init__(self,
             bmodel_bytes:str,
             bmodel_size: int,
             tpu_id: int,
             mode: sail.IOMode)
```

Parameters

· bmodel_bytes : bytes

Bytes of bmodel in system memory

· bmodel_size : int

Bmodel byte size

· tpu_id : int

Tensor Computing Processor ID. You can use bm-smi to see available IDs

· mode : sail.IOMode

Specify the input/output tensors are in system memory or device memory

Sample:

```
import sophon.sail as sail
import os
if __name__ == '__main__':
    engine1 = sail.Engine(0)
    bmodel_path = "your_bmodel.bmodel"
    handle = sail.Handle(0)
    engine2 = sail.Engine(handle)

    engine3 = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    file = open(bmodel_path,"rb")
    datas = file.read()
    file_size = os.path.getsize(bmodel_path)
    engine4 = sail.Engine(datas,file_size,0,sail.IOMode.SYSI)
```

5.6.2 get_handle

Get Handle instance.

Interface:

```
def get_handle(self)->sail.Handle
```

Returns

· handle: sail.Handle

Handle instance

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    engine1 = sail.Engine(0)
    handle = engine1.get_handle()
```

5.6.3 load

Load bmodel from file.

Interface:

```
def load(self, bmodel_path: str)->bool
```

Parameters

· bmodel_path : str

Path to bmodel

Interface:

```
def load(self, bmodel_bytes: bytes, bmodel_size: int)->bool
```

Parameters

· bmodel_bytes : bytes

Bytes of bmodel in system memory

· bmodel_size : int

Bmodel byte size

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine1 = sail.Engine(0)
    engine1.load(bmodel_path)
```

5.6.4 get_graph_names

Get all graph names in the loaded bmodels.

Interface:

```
def get_graph_names(self)-> list
```

Returns

· graph_names : list

Graph names list in loaded context

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine1 = sail.Engine(0)
    engine1.load(bmodel_path)
    graph_names = engine1.get_graph_names()
```

5.6.5 set_io_mode

Set IOMode for a graph.

Interface:

```
def set_io_mode(self, graph_name: str, mode: sail.IOMode)
```

Parameters

- graph_name: str

The specified graph name

- mode : sail.IOMode

Specified io mode

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    engine.set_io_mode(graph_name,sail.IOMode.SYSI)
```

5.6.6 graph_is_dynamic

Determine whether a selected computational map is dynamic.

Interface:

```
def graph_is_dynamic(self, graph_name: str) -> list
```

Parameters

- graph_name : str

Specified graph name

Returns

- is_dynamic : bool

A boolean value indicating whether the selected computation graph is dynamic or not.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    is_dynamic = engine.graph_is_dynamic(graph_name)
```

5.6.7 get_input_names

Get all input tensor names of the specified graph.

Interface:

```
def get_input_names(self, graph_name: str) -> list
```

Parameters

- graph_name : str

Specified graph name

Returns

- input_names : list

All the input tensor names of the graph

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    input_names = engine.get_input_names(graph_name)
```

5.6.8 get_output_names

Get all output tensor names of the specified graph.

Interface:

```
def get_output_names(self, graph_name: str)-> list
```

Parameters

- graph_name : str

Specified graph name

Returns

- input_names : list

All the output tensor names of the graph

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    output_names = engine.get_output_names(graph_name)
```

5.6.9 get_max_input_shapes

Get max shapes of input tensors in a graph. For static models, the max shape is fixed and it should not be changed. For dynamic models, the tensor shape should be smaller than or equal to the max shape.

Interface:

```
def get_max_input_shapes(self, graph_name: str)-> dict {str : list}
```

Parameters

- graph_name : str

The specified graph name

Returns

- max_shapes : dict {str : list}

Max shape of the input tensors

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    max_input_shapes = engine.get_max_input_shapes(graph_name)
```

5.6.10 get_input_shape

Get the maximum dimension shape of an input tensor in a graph. There are cases that there are multiple input shapes in one input name, This API only returns the maximum dimension one for the memory allocation in order to get the best performance.

Interface:

```
def get_input_shape(self, graph_name: str, tensor_name: str)-> list
```

Parameters

· graph_name : str

The specified graph name

· tensor_name : str

The specified input tensor name

Returns

· tensor_shape : list

The maxmim dimension shape of the tensor

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    input_name = engine.get_input_names(graph_name)[0]
    input_shape = engine.get_input_shape(graph_name,input_name)
```

5.6.11 get_max_output_shapes

Get max shapes of input tensors in a graph. For static models, the max shape is fixed and it should not be changed. For dynamic models, the tensor shape should be smaller than or equal to the max shape.

Interface:

```
def get_max_output_shapes(self, graph_name: str)-> dict {str : list}
```

Parameters

· graph_name : str

The specified graph name

Returns

- max_shapes : dict {str : list}

Max shape of the output tensors

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    max_output_shapes = engine.get_max_output_shapes(graph_name)
```

5.6.12 get_output_shape

Get the shape of an output tensor in a graph.

Interface:

```
def get_output_shape(self, graph_name: str, tensor_name: str)-> list
```

Parameters

- graph_name : str

The specified graph name

- tensor_name : str

The specified output tensor name

Returns

tensor_shape : list

The shape of the tensor

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    output_name = engine.get_output_names(graph_name)[0]
    input_shape = engine.get_output_shape(graph_name,output_name)
```

5.6.13 get_input_dtype

Get scale of an input tensor. Only used for int8 models.

Interface:

```
def get_input_dtype(self, graph_name: str, tensor_name: str)-> sail.Dtype
```

Parameters

- graph_name : str

The specified graph name

- tensor_name : str

The specified output tensor name

Returns

- scale: sail.Dtype

Data type of the input tensor

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    input_name = engine.get_input_names(graph_name)[0]
    input_dtype = engine.get_input_dtype(graph_name,input_name)
```

5.6.14 get_output_dtype

Get scale of an output tensor. Only used for int8 models.

Interface:

```
def get_output_dtype(self, graph_name: str, tensor_name: str)-> sail.Dtype
```

Parameters

- graph_name : str

The specified graph name

- tensor_name : str

The specified output tensor name

Returns

- scale: sail.Dtype

Data type of the output tensor

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    output_name = engine.get_output_names(graph_name)[0]
    input_shape = engine.get_output_dtype(graph_name,output_name)
```

5.6.15 get_input_scale

Get scale of an input tensor. Only used for int8 models.

Interface:

```
def get_input_scale(self, graph_name: str, tensor_name: str)-> float32
```

Parameters

- graph_name : str

The specified graph name

- tensor_name : str

The specified output tensor name

Returns

- scale: float32

Scale of the input tensor

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    input_name = engine.get_input_names(graph_name)[0]
    input_scale = engine.get_input_scale(graph_name,input_name)
```

5.6.16 get_output_scale

Get scale of an output tensor. Only used for int8 models.

Interface:

```
def get_output_scale(self, graph_name: str, tensor_name: str)-> float32
```

Parameters

- graph_name : str

The specified graph name

- tensor_name : str

The specified output tensor name

Returns

- scale: float32

Scale of the output tensor

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    output_name = engine.get_output_names(graph_name)[0]
    output_scale = engine.get_output_scale(graph_name,output_name)
```

5.6.17 process

Inference with provided system data of input tensors, with or without input shapes and output tensors.

Interface:

```
def process(self,
            graph_name: str,
            input_tensors: dict {str : numpy.array},
            core_list: list[int])-> dict {str : numpy.array}
```

Parameters

- graph_name : str

The specified graph name

- input_tensors : dict {str : numpy.array}

Data of all input tensors in system memory

- core_list : list[int]

This parameter is only valid for processors that support multi-core inference, and the core used for inference can be selected. Set bmodel as the corresponding kernel number N, and if corelist is empty, use N cores starting from core0 for inference; If the length of corelist is greater than N, use the corresponding top N cores in corelist for inference. This parameter can be ignored for processors that only support single core inference. **Returns**

- output_tensors : dict {str : numpy.array}

Data of all output tensors in system memory

Interface:

```
def process(self,
            graph_name: str,
            input_tensors: dict {str : sail.Tensor},
            output_tensors: dict {str : sail.Tensor},
            core_list: list[int])
```

Parameters

- graph_name : str

The specified graph name

- input_tensors : dict {str : sail.Tensor}

Input tensors managed by user

- output_tensors : dict {str : sail.Tensor}

Output tensors managed by user

- core_list : list[int]

This parameter is only valid for processors that support multi-core inference, and the core used for inference can be selected. Set bmodel as the corresponding kernel number N, and if corelist is empty, use N cores starting from core0 for inference; If the length of corelist is greater than N, use the corresponding top N cores in corelist for inference. This parameter can be ignored for processors that only support single core inference.

Interface:

```
def process(self,
            graph_name: str,
            input_tensors: dict {str : sail.Tensor},
            input_shapes: dict {str : list},
            output_tensors: dict {str : sail.Tensor},
            core_list: list[int])
```

Parameters

- graph_name : str

The specified graph name

- input_tensors : dict {str : sail.Tensor}

Input tensors managed by user

- input_shapes : dict {str : list}

Shapes of all input tensors

- output_tensors : dict {str : sail.Tensor}

Output tensors managed by user

- core_list : list[int]

This parameter is only valid for processors that support multi-core inference, and the core used for inference can be selected. Set bmodel as the corresponding kernel number N, and if corelist is empty, use N cores starting from core0 for inference; If the length of corelist is greater than N, use the corresponding top N cores in corelist for inference. This parameter can be ignored for processors that only support single core inference.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    # prepare tensor map
    input_tensors_map = engine.create_input_tensors_map(graph_name)
    data_dict = {key: tensor.asnumpy() for key, tensor in input_tensors_map.items()}
    # inference type1
    output_tensors_map = engine.process(graph_name, data_dict)

    # inference type2
    output_tensors_map_ = engine.create_output_tensors_map(graph_name)
    engine.process(graph_name, input_tensors_map, output_tensors_map_)
```

5.6.18 get_device_id

Get device id of this engine

Interface:

```
def get_device_id(self)-> int
```

Returns

- tpu_id : int

Tensor Computing Processor id of this engine

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    dev_id = engine.get_device_id()
```

5.6.19 create_input_tensors_map

Create input tensors map, according to and bmodel.

Interface:

```
def create_input_tensors_map(self,
                             graph_name: str,
                             create_mode: int)-> dict[str,Tensor]
```

Parameters:

- graph_name : str

The specified graph name.

- create_mode: int

Tensor Create mode, case 0: only allocate system memory; case 1: only allocate device memory; case other: according to engine IOMode.

Returns

- output: dict[str,Tensor]

Output result.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    # prepare tensor map
    input_tensors_map = engine.create_input_tensors_map(graph_name)
```

5.6.20 create_output_tensors_map

Create output tensors map, according to and bmodel.

Interface:

```
def create_output_tensors_map(self,
                              graph_name: str,
                              create_mode: int)-> dict[str,Tensor]
```

Parameters:

- graph_name : str

The specified graph name.

- create_mode: int

Tensor Create mode, case 0: only allocate system memory; case 1: only allocate device memory; case other: according to engine IOMode.

Returns

- output: dict[str,Tensor]

Output result.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.Engine(bmodel_path,0,sail.IOMode.SYSI)
    graph_name = engine.get_graph_names()[0]
    # prepare tensor map
    output_tensors_map = engine.create_output_tensors_map(graph_name)
```

5.7 sail.MultiEngine

5.7.1 MultiEngine

Interface:

```
def __init__(self,
             bmodel_path: str,
             device_ids: list[int],
             sys_out: bool = True,
             graph_idx: int = 0)
```

Parameters

- bmodel_path : str

Path to bmodel

- device_ids : list[int]

Tensor Computing Processor ID. You can use bm-smi to see available IDs

- sys_out : bool, default: True

The flag of copy result to system memory.

- graph_idx : int, default: 0

The specified graph index

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
```

5.7.2 set_print_flag

Print debug messages.

Interface:

```
def set_print_flag(self, print_flag: bool)
```

Parameters

- print_flag : bool

if print_flag is true, print debug messages

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    engine.set_print_flag(true)
```

5.7.3 set_print_time

Print main process time use.

Interface:

```
def set_print_time(self, print_flag: bool)
```

Parameters

- print_flag : bool

if print_flag is true, print main process time use, Otherwise not print.

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    engine.set_print_time(true)
```

5.7.4 get_device_ids

Get device ids of this MultiEngine.

Interface:

```
def get_device_ids(self)-> list[int]
```

Returns

- device_ids : list[int]

Tensor Computing Processor ids of this MultiEngine.

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    ids = engine.get_device_ids()
```

5.7.5 get_graph_names

Get all graph names in the loaded bmodels.

Interface:

```
def get_graph_names(self)-> list
```

Returns

- graph_names : list

Graph names list in loaded context

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
```

5.7.6 get_input_names

Get all input tensor names of the specified graph.

Interface:

```
def get_input_names(self, graph_name: str)-> list
```

Parameters

- graph_name : str

Specified graph name

Returns

- input_names : list

All the input tensor names of the graph

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
    input_names = engine.get_input_names(graph_names[0])
```

5.7.7 get_output_names

Get all output tensor names of the specified graph.

Interface:

```
def get_output_names(self, graph_name: str)-> list
```

Parameters

- graph_name : str

Specified graph name

Returns

- input_names : list

All the output tensor names of the graph

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
    output_names = engine.get_output_names(graph_names[0])
```

5.7.8 get_input_shape

Get the maximum dimension shape of an input tensor in a graph. There are cases that there are multiple input shapes in one input name, This API only returns the maximum dimension one for the memory allocation in order to get the best performance.

Interface:

```
def get_input_shape(self, graph_name: str, tensor_name: str)-> list
```

Parameters

- graph_name : str

The specified graph name

- tensor_name : str

The specified input tensor name

Returns

- tensor_shape : list

The maxmim dimension shape of the tensor

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
    input_names = engine.get_input_names(graph_names[0])
    input_shape = engine.get_input_shape(graph_names[0],input_names[0])
```

5.7.9 get_output_shape

Get the shape of an output tensor in a graph.

Interface:

```
def get_output_shape(self, graph_name: str, tensor_name: str)-> list
```

Parameters

- graph_name : str

The specified graph name

- tensor_name : str

The specified output tensor name

Returns

- tensor_shape : list

The shape of the tensor

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    dev_id = [0,1]
    bmodel_path = "your_bmodel.bmodel"
    engine = sail.MultiEngine(bmodel_path,dev_id)
    graph_names = engine.get_graph_names()
    output_names = engine.get_output_names(graph_names[0])
    output_shape = engine.get_output_shape(graph_names[0], output_names[0])
```


5.7.10 process

Inference with provided system data of input tensors.

Interface:

```
def process(self, input_tensors: dict {str : numpy.array})-> dict {str : numpy.array}

def process(self, input_tensors: list[dict{str: sophon.sail.Tensor}] )->dict {str : Tensor}
```

Parameters

- input_tensors : dict {str : numpy.array}

Data of all input tensors in system memory

Returns

- output_tensors : dict {str : numpy.array}

Data of all output tensors in system memory

示例代码:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    dev_id = [0,1]
    handle = sail.Handle(0)
    bmodel_path = 'your_bmodel.bmodel'
    engine = sail.MultiEngine(bmodel_path, dev_id)
    graph_name = engine.get_graph_names()[0]
    input_names = engine.get_input_names(graph_name)
    output_names = engine.get_output_names(graph_name)

    input_tensors_map = {}

    # form 1
    input_numpy_map = {}
    for input_name in input_names:
        data = np.ones(engine.get_input_shape(graph_name,input_name),dtype=np.float32)
        input_numpy_map = {input_name:data}
    output_tensors_map = engine.process(input_numpy_map)
    print(output_tensors_map)

    # form 2
    for input_name in input_names:
        data = np.ones(engine.get_input_shape(graph_name,input_name),dtype=np.float32)
        tensor = sail.Tensor(handle,data)
        input_tensors_map[input_name] = tensor
    input_tensors_vector = [input_tensors_map]
    output_tensors_map = engine.process(input_tensors_vector)
    print(output_tensors_map)
```

5.8 sail.bm_image

Functions to get bm_image attributes.

5.8.1 width

Get width of image.

Interface:

```
def width(self) -> int
```

Returns:

- width : int

The width of image.

5.8.2 height

Get height of image.

Interface:

```
def height(self) -> int
```

Returns:

- height : int

The height of image.

5.8.3 format

Get format of image.

Interface:

```
def format(self) -> bm_image_format_ext
```

Returns:

- format : bm_image_format_ext

Get the format of image.

5.8.4 dtype

Get dtype of image.

Interface:

```
def dtype(self) -> bm_image_data_format_ext
```

Returns:

- dtype : bm_image_data_format_ext

The dtype of image.

示例代码:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image.jpg"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmimg = BMimg.data()# here is a sail.bm_image
    print(bmimg.width(),bmimg.height(),bmimg.format(),bmimg.dtype())
```

5.9 sail.BMImage

5.9.1 __init__

Interface:

```
def __init__(self)


def __init__(self,
             handle: sail.Handle,
             h: int,
             w: int,
             format: bm_image_format_ext,
             dtype: sail.bm_image_data_format_ext)

def __init__(self, handle: sail.Handle,
             buffer: bytes | np.array,
             h: int,
             w: int,
             format: sail.Format,
             dtype: sail.ImgDtype = DATA_TYPE_EXT_1N_BYTE,
             strides: list[int] = None,
             offset: int = 0)
```

Parameters

- handle : sail.Handle

Handle instance

- h: int

The height of img

- w: int

The width of img

- format : bm_image_format_ext

The pixel format of img. Supported formats are listed in sail.Format .

- dtype: sail.bm_image_data_format_ext

The data type of img. Supported data types are listed in ImgDtype .

- buffer: bytes | np.array

The buffer used to create a BMImage, which contains pixel values

- strides

The stride of the image when creating an image with a buffer. The unit is in bytes. The default is empty, indicating that it is the same as the data width of one row. If specified, ensure the number of elements in the list matches the number of image planes.

- offset

The offset of valid data relative to the start address of the buffer when creating an image with a buffer. The unit is in bytes, and the default is 0.

5.9.2 width

Get the img width.

Interface:

```
def width(self)-> int
```

Returns

- width : int

The width of img

5.9.3 height

Get the img height.

Interface:

```
def height(self)-> int
```

Returns

- height : int

The height of img

5.9.4 format

Get the img format.

Interface:

```
def format(self)-> bm_image_format_ext
```

Returns

- format : bm_image_format_ext

The format of img

5.9.5 dtype

Get the img dtype.

Interface:

```
def dtype(self)-> bm_image_data_format_ext
```

Returns

- dtype: bm_image_data_format_ext

The data type of img

5.9.6 data

Get inner bm_image.

Interface:

```
def data(self)-> bm_image
```

Returns

- img : bm_image

the data of img

5.9.7 get_device_id

Get device id of this image.

Interface:

```
def get_device_id(self)-> int
```

Returns

- device_id : int

Tensor Computing Processor ids of this image.

5.9.8 get_handle

Get Handle of BMImage.

Interface:

```
def get_handle(self):
```

Return:

- Handle : Handle

Return the Handle of BMImage.

5.9.9 asmat

Convert to cv Mat

Interface:

```
def asmat(self)-> numpy.ndarray[numpy.uint8]
```

Returns

- image : numpy.ndarray[numpy.uint8]

only support uint8

5.9.10 asnumpy

Convert BMImage’s data to numpy.ndarray, with original pixel format.

Supported pixel formats are listed in sail.Format .

Supported data type are DATA_TYPE_EXT_1N_BYTE、DATA_TYPE_EXT_1N_BYTE_SIGNED和

The returned ndarray’s shape is corresponding to BMImage’s pixel format.

pixel format	ndarray’s shape
FORMAT_BGR_PACKED / FORMAT_BGR_PACKED	(height, width, 3)
FORMAT_ARGB_PACKED / FORMAT_ABGR_PACKED	(height, width, 4)
FORMAT_GRAY	(1, height, width)
FORMAT_BGR_PLANAR / FORMAT_RGB_PLANAR	(3, height, width)
FORMAT_YUV444P	(3, height, width)
else	(numel,)

In above table, numel means the number of elements in BMImage. For example, if pixel format is YUV420P or NV12, numel = height * width * 1.5 ; if pixel format is BGR_PACKED or BGR_PLANAR, numel = height * width * 3 .

接口形式:

```
def asnumpy(self) -> numpy.ndarray
```

返回值说明:

- image : numpy.ndarray

return data in BMImage.

示例代码:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    devid = 0
    handle = sail.Handle(devid)
    height = 1080
    width = 1920
    dtype = sail.ImgDtype.DATA_TYPE_EXT_1N_BYTE
    np_dtype = np.uint8

    # example for BGR_PLANAR
    format = sail.Format.FORMAT_BGR_PLANAR
    numel = int(height * width * 3)
```

(continues on next page)

(continued from previous page)

```
    rawdata = np.random.randint(0, 255, (numel,), np_dtype)
    img = sail.BMImage(handle, rawdata, height, width, format, dtype)
    out_ndarray = img.asnumpy()
    assert out_ndarray.shape == (3, height, width)

    # example for YUV420P
    format = sail.Format.FORMAT_YUV420P
    numel = int(height * width * 1.5)
    rawdata = np.random.randint(0, 255, (numel,), np_dtype)
    img = sail.BMImage(handle, rawdata, height, width, format, dtype)
    out_ndarray = img.asnumpy()
    assert out_ndarray.shape == (numel,)
```

5.9.11 get_plane_num

Get plane number of this image

Interface:

```
def get_plane_num(self) -> int:
```

5.9.12 align

Align the bm_image to 64 bytes

Interface:

```
def align(self) -> int:
```

Returns

- ret : int

return if BMImage aligned,-1 failed,0 successed

5.9.13 check_align

Check if the bm_image aligned

Interface:

```
def check_align(self) -> bool:
```

Returns

- ret : bool

return if BMImage aligned,1 aligned,0 unaligned

5.9.14 unalign

Unalign the bm_image to source bm_image

Interface:

```
def unalign(self) -> int:
```

Returns

- ret : int

return if BMImage unaligned,-1 failed,0 successed

5.9.15 check_contiguous_memory

Check if the bm_image' s memory contiguous

Interface:

```
def check_contiguous_memory(self) -> bool:
```

Returns

- ret : bool

return if BMImage memory contiguous,1 contiguous,0 uncontiguous

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    file_path = "your_image.jpg" # 请替换为您的文件路径
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path, False, dev_id)
    BMimg = sail.BMImage()
    ret = decoder.read(handle, BMimg)

    # get bm_image
    bm_image = BMimg.data()

    # get BMimg width,height,dtype,format,device_id,plane_num,handle
    print(BMimg.width(), BMimg.height(), BMimg.format(), BMimg.dtype(), BMimg.get_
↪device_id(), BMimg.get_plane_num(), BMimg.get_handle())

    # get mat
    np_data = BMimg.asmat()

    # align BMimg
    ret = BMimg.align()
```

(continues on next page)

(continued from previous page)

```
    if ret == 0:
        print("align success")
    else:
        print("align failed")

    print(BMimg.check_align())

    # unalign BMimg
    ret = BMimg.unalign()
    if ret == 0:
        print("unalign success")
    else:
        print("unalign failed")

    # check contiguous memory
    print(BMimg.check_contiguous_memory())

    # create BMImage with data from buffer
    buf = bytes([i % 256 for i in range(int(200*100*3))])
    img_fromRawdata = sail.BMImage(handle, buf, 200, 100, sail.Format.FORMAT_BGR_
↪PACKED)
```

5.9.16 get_pts_dts

Get pts and dts.

Interface:

```
def get_pts_dts() -> list
```

Returns

- result : list

the value of pts and dts.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    input_file_path = 'your_rtsp_url'
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(input_file_path, True, dev_id)
    image = sail.BMImage()
    ret = decoder.read(handle, image)
    if ret == 0:
        print("Frame read successfully into bm_image")
        pts,dts=image.get_pts_dts()
        print("pts:",pts)
```

(continues on next page)

(continued from previous page)

```
        print("dts:",dts)
    else:
        print("Failed to read frame into bm_image")
```

5.10 sail.BMImageArray

5.10.1 __init__

Interface:

```
def __init__(self)


def __init__(self,
             handle: sail.Handle,
             h: int,
             w: int,
             format: bm_image_format_ext,
             dtype: bm_image_data_format_ext)
```

Parameters

- handle : sail.Handle

Handle instance

- h : int

Height instance

- w : int

Width instance

- format : bm_image_format_ext

Format instance

- dtype : bm_image_data_format_ext

Dtype instance

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D
    images = sail.BMImageArray4D()

    # Create BMImageArray4D with parameters
    dev_id = 0
```

(continues on next page)

(continued from previous page)

```
    handle = sail.Handle(dev_id)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,dev_id)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
```

5.10.2 __getitem__

Get the bm_image from index i.

Interface:

```
def __getitem__(self, i: int)-> sail.bm_image
```

Parameters

- i : int

Index of the specified location.

Returns

- img : sail.bm_image

result bm_image

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    dev_id = 0
    handle = sail.Handle(dev_id)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,dev_id)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())

    images.copy_from(0,ori_img)
    # get the bm_image from index 0
    img_0 = images.__getitem__(0)
    print("image0 from bmimg_array:",img_0.width(),img_0.height(),img_0.dtype())
```

5.10.3 __setitem__

Copy the image to the specified index.

Interface:

```
def __setitem__(self, i: int, data: sail.bm_image)
```

Parameters

- i: int

Index of the specified location.

- data: sail.bm_image

Input image

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    dev_id = 0
    handle = sail.Handle(dev_id)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,dev_id)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
    # copy image to the specified index
    images.__setitem__(3,ori_img.data())
```

5.10.4 copy_from

Copy the image to the specified index.

Interface:

```
def copy_from(self, i: int, data: sail.BMImage):
```

Parameters

- i: int

Index of the specified location.

- data: sail.BMImage

Input image

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    dev_id = 0
    handle = sail.Handle(dev_id)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,dev_id)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
    # copy image to the specified index
    images.copy_from(0,ori_img)
```

5.10.5 attach_from

Attach the image to the specified index.(Because there is no memory copy, the original data needs to be cached)

Interface:

```
def attach_from(self, i: int, data: BMImage):
```

Parameters:

- i: int

Index of the specified location.

- data: BMImage

Input image.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    dev_id = 0
    handle = sail.Handle(dev_id)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,dev_id)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
    # Attach image to the specified index
    images.attach_from(1,ori_img)
```

5.10.6 get_device_id

Get device id of this BMImageArray.

Interface:

```
def get_device_id(self) -> int:
```

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    # Create BMImageArray4D with parameters
    dev_id = 0
    handle = sail.Handle(dev_id)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("your_image.jpg",True,dev_id)
    ori_img = decoder.read(handle)
    images = sail.BMImageArray4D(handle, ori_img.height(), ori_img.width(), ori_img.
↪format(), ori_img.dtype())
    # Get device id of this BMImageArray
    devid = images.get_device_id()
    print("device id:",devid)
```

5.11 sail.Decoder

5.11.1 __init__

Interface:

```
def __init__(self,
             file_path: str,
             compressed: bool = True,
             tpu_id: int = 0)
```

Parameters

- file_path : str

Path or rtsp url to the video/image file

- compressed : bool, default: True

Whether the format of decoded output is compressed NV12.

- tpu_id: int, default: 0

ID of Tensor Computing Processor, there may be more than one Tensor Computing Processor for PCIE mode.

5.11.2 is_opened

Judge if the source is opened successfully.

Interface:

```
def is_opened(self)-> bool
```

Returns

- judge_ret : bool

True for success and False for failure

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'
    dev_id = 0
    decoder = sail.Decoder(file_path, True, dev_id)
    ret = decoder.is_opened()
    print("Decoder opened:", ret)
```

5.11.3 read

Read an image from the Decoder.

Interface:

```
def read(self, handle: sail.Handle, image: sail.BMImage)-> int
```

Parameters

- handle : sail.Handle

Handle instance

- image : sail.BMImage

BMImage instance

Returns

- judge_ret : int

0 for success and others for failure

Interface:

```
def read(self, handle: sail.Handle)-> sail.BMImage
```

Parameters

- handle : sail.Handle

Handle instance

Returns

- image : sail.BMImage

BMImage instance

Sample1:

```
import sophon.sail as sail
if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path, True, dev_id)
    image = sail.BMImage()
    ret = decoder.read(handle, image)
    if ret == 0:
        print("Frame read successfully")
    else:
        print("Failed to read frame")
```

Sample2:

```
import sophon.sail as sail
if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path, True, dev_id)
    BMimg = decoder.read(handle)
```

5.11.4 read_

Read an image from the Decoder.

Interface:

```
def read_(self, handle: sail.Handle, image: sail.bm_image)-> int
```

Parameters

- handle : sail.Handle

Handle instance

- image : sail.bm_image

bm_image instance

Returns

- judge_ret : int

0 for success and others for failure

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path, True, dev_id)
    image = sail.BMImage()
    bm_img = image.data()
    ret = decoder.read_(handle, bm_img)
    if ret == 0:
        print("Frame read successfully into bm_image")
    else:
        print("Failed to read frame into bm_image")
```

5.11.5 get_frame_shape

Get frame shape in the Decoder.

Interface:

```
def get_frame_shape(self)-> list
```

Returns

- frame_shape : list

The shape of the frame

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    file_path = 'your_video_file_path.mp4'
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path,True,dev_id)
    print(decoder.get_frame_shape())
```

5.11.6 release

Release the Decoder.

Interface:

```
def release(self)
```

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    file_path = 'your_video_file_path.mp4'
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(file_path,True,dev_id)
    decoder.release()
```

5.11.7 reconnect

Reconnect the Decoder.

Interface:

```
def reconnect(self)
```

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    file_path = "your_video_file_path.mp4"
    decoder = sail.Decoder(file_path, True, dev_id)
    decoder.reconnect()
```

5.11.8 enable_dump

Enable the dump input video ability of the decoder (without encoding) and cache up to 1000 frames of undecoded video.

Interface:

```
def enable_dump(dump_max_seconds: int):
    """ enable input video dump without encode.
    """
```

Parameters

- dump_max_seconds : int

dump video max length.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    file_path = "your_video_file_path.mp4"
    decoder = sail.Decoder(file_path, True, dev_id)
```

(continues on next page)

(continued from previous page)

```
    dump_max_seconds = 100
    decoder.enable_dump(dump_max_seconds)
```

5.11.9 disable_dump

Disable the dump input video ability of the decoder and clear the cache queue.

Interface:

```
def disable_dump():
    """ Disable  input video dump without encode.
    """
```

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    file_path = "your_video_file_path.mp4"
    decoder = sail.Decoder(file_path, True, dev_id)
    decoder.enable_dump(100)
    decoder.disable_dump()
```

5.11.10 dump

At the time of calling this function, dump the input video for several seconds before and after. Due to the lack of encoding, it is necessary to dump the keyframes that all frames depend on within a few seconds before and after. Therefore, the dump implementation of the interface is based on gop, and the actual video duration under dump will be higher than the input parameter duration. The error depends on the gop of the input video. The larger the size and gop, the larger the error.

Interface:

```
def dump(dump_pre_seconds, dump_post_seconds, file_path)->int
    """ dump input video without encode.

    Parameters:
    ----------
    dump_pre_seconds : int
        dump video length(seconds) before dump moment
    dump_post_seconds : int
        dump video length(seconds) after dump moment
    file_path : str
        output path

    Returns
```

(continues on next page)

(continued from previous page)

```
    -------
    int, 0 for success
    """
```

Sample:

```
import sophon.sail as sail
import time

if __name__ == '__main__':
    dev_id = 0
    input_file_path = "your_rtsp_url"
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(input_file_path, True, dev_id)
    decoder.enable_dump(30)
    dump_pre_seconds = 10
    dump_post_seconds = 10
    output_file_path = "output_video_path.mp4"

    # start decode
    t_decode = time.time()
    while(True):
        if time.time() - t_decode > dump_pre_seconds:
            break
        _ = decoder.read(handle)

    # start dump
    ret = decoder.dump(dump_pre_seconds, dump_post_seconds, output_file_path)
    if ret == 0:
        print("Decoder dump start!")
    else:
        print("Decoder dump fail!")
        exit(-1)

    # continue decode
    t_dump = time.time()
    while(True):
        if time.time() - t_dump > dump_post_seconds:
            print("Decoder dump finish!")
            break
        _ = decoder.read(handle)

    time.sleep(1)
    print("exit")
```

5.11.11 get_pts_dts

Get pts and dts.

Interface:

```
def get_pts_dts() -> list
```

Returns

- result : list

the value of pts and dts.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    input_file_path = 'your_rtsp_url'
    dev_id = 0
    handle = sail.Handle(dev_id)
    decoder = sail.Decoder(input_file_path, True, dev_id)
    image = sail.BMImage()
    ret = decoder.read(handle, image)
    if ret == 0:
        print("Frame read successfully into bm_image")
        pts,dts=decoder.get_pts_dts()
        print("pts:",pts)
        print("dts:",dts)
    else:
        print("Failed to read frame into bm_image")
```

5.12 sail.Encoder

Encoder, supporting video and image encoding.

5.12.1 __init__

Encoder init

piture encoder init:

Interface:

```
def __init__(self)
```

video encoder init:

Interface:

```
def __init__(self, output_path: str, handle: sail.Handle, enc_fmt: str, pix_fmt: str, enc_
↪params: str, cache_buffer_length: int=5, abort_policy: int=0)

def __init__(self, output_path: str, device_id: int, enc_fmt: str, pix_fmt: str, enc_
↪params: str, cache_buffer_length: int=5, abort_policy: int=0)
```

Parameters:

- output_path: str

output path, support local video file and rtsp/rtmp stream.

- handle: sail.Hnadle

Handle instance. (either handle instance or device_id)

- device_id: int

Encoder device_id. (either handle instance or device_id, when specify device_id, the encoder will create a Handle internally)

- enc_fmt: str

encoder format, support h264_bm and h265_bm/hevc_bm.

- pix_fmt: str

output pixel format, support NV12 and I420. I420 is recommended.

- enc_params: str

encoder params, "width=1902:height=1080:gop=32:gop_preset=3:framerate=25:bitrate=2000" , width and height are necessary. By default, Bitrate is used to control quality, and Bitrate becomes invalid when qp is specified in the parameter.

- cache_buffer_length: int

The internal cache queue length defaults to 5. sail.Encoder internally maintains a cache queue to improve flow control fault tolerance during pushing stream.

- abort_policy: int

The reject policy for video_write. 0 for returns -1 immediately. 1 for pop queue header. 2 for clear the queue. 3 for blocking.

5.12.2 is_opened

Determine if the encoder is turned on.

Interface:

```
def is_opened(self) -> bool
```

return:

- judge_ret: bool

return True when opened, and False when failed.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    out_path = "path/to/your/output/file"
    enc_fmt = "h264_bm"
    pix_fmt = "I420"
    enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25"
    cache_buffer_length = 5
    abort_policy = 0
    encoder = sail.Encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy)
    print(encoder.is_opened())
```

5.12.3 pic_encode

Encode an image and return the encoded data.

Interface1:

```
def pic_encode(self, ext: str, image: BMImage)->numpy.array
```

Interface2:

```
def pic_encode(self, ext: str, image: bm_image)->numpy.array
```

Parameters:

- ext: str

Image encoding format. such as ".jpg" , ".png"

- image: sail.BMImage

Input image, only supports picture of FORMAT_BGR_PACKED and DATA_TYPE_EXT_1N_BYTE.

return:

- data: numpy.array

Encoded data stored in system memory.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
```

(continues on next page)

(continued from previous page)

```
    handle = sail.Handle(dev_id)
    img_path = "path/to/your/output/file"
    decoder = sail.Decoder(img_path,False,dev_id)
    img = decoder.read(handle)
    # img = decoder.read(handle).data()  //bm_image
    encoder = sail.Encoder()
    data = encoder.pic_encode(".jpg", img)
    print(data)
```

5.12.4 video_write

Send a frame of image to the video encoder. Asynchronous interface, after format conversion, is placed in the internal cache queue.

Interface1:

```
def video_write(self, image: sail.BMImage)->int
```

Interface2:

```
def video_write(self, image: sail.bm_image)->int
```

Parameters:

- image: sail.BMImage

On the BM1684, when the pixel format (pix_fmt) of the encoder is set to I420, the shape of the image to be encoded can differ from the encoder' s width and height. However, when the pixel format is NV12, the image shape must match the encoder' s dimensions. In this case, a format conversion is performed internally using bmcv_image_storage_convert, which may utilize NPU resources.

On the BM1684X, the shape of the image to be encoded can differ from the encoder' s width and height. The internal resizing and format conversion are handled by bmcv_image_vpp_convert.

Return:

- judge_ret: int

Successfully returned 0, internal cache queue full returned -1. encode failed returns -2. push stream failed returns -3. unknown abort policy returns -4.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    img_path = "your_img_path"
```

(continues on next page)

(continued from previous page)

```
    decoder = sail.Decoder(img_path,False,dev_id)
    img = decoder.read(handle)
    out_path = "path/to/your/output/file"
    enc_fmt = "h264_bm"
    pix_fmt = "I420"
    enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25"
    cache_buffer_length = 5
    abort_policy = 0
    encoder = sail.Encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy)
    ret = encoder.video_write(img)
    # ret = encoder.video_write(img.data())  #  sail.bm_image
    print(ret)
```

5.12.5 release

release encoder

Interface:

```
def release(self)->None
```

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    dev_id = 0
    handle = sail.Handle(dev_id)
    out_path = "path/to/your/output/file"
    enc_fmt = "h264_bm"
    pix_fmt = "I420"
    enc_params = "width=1920:height=1080:bitrate=2000:gop=32:gop_
↪preset=2:framerate=25"
    cache_buffer_length = 5
    abort_policy = 0
    encoder = sail.Encoder(out_path, handle, enc_fmt, pix_fmt, enc_params, cache_buffer_
↪length, abort_policy)
    encoder.release()
```

5.13 sail.Decoder_RawStream

Original stream decoder for H264/H265 decoding。

5.13.1 __init__

Interface:

```
def __init__(self, tpu_id: int, decformat: str)
```

Parameters:

- tpu_id: int

The Tensor Computing Processor id that used, which defaults to 0.

- decformat: str

Input image format, supports h264 and h265.

5.13.2 read

Read a frame of image from Decoder.

Interface:

```
def read(self, data: bytes, image: BMImage, continue_frame: bool = False) -> int
```

Parameters:

- data: bytes

Input parameter. The binary data of the original stream.

- image: sail.BMImage

Output parameter. Read data into the BMImage.

- continue_frame: bool

Input parameter. Whether to read frames continuously, the default is False.

Returns:

- judge_ret: int

Returns 0 if the read is successful and other values if failed.

5.13.3 read_

Read a frame of image from Decoder.

Interface:

```
read_(self, data_bytes: bytes, image: bm_image, continue_frame: bool = False)
```

Parameters:

- data: bytes

Input parameter. The binary data of the original stream.

- image: sail.bm_image

Output parameter. Read data into the BMImage.

- continue_frame: bool

Input parameter. Whether to read frames continuously, the default is False.

Returns:

- judge_ret: int

Returns 0 if the read is successful and other values if failed.

5.13.4 release

Release the Decoder.

Interface:

```
def release(self)
```

5.14 sail.Bmcv

5.14.1 __init__

Interface:

```
def __init__(self, handle: sail.Handle)
```

Parameters: * handle : sail.Handle

Handle instance

5.14.2 bm_image_to_tensor

Convert image to tensor.

Interface1:

```
def bm_image_to_tensor(self,
                       image: sail.BMImage | sail.BMImageArray
                       ) -> sail.Tensor
```

Parameters: * image : sail.BMImage | sail.BMImageArray

BMImage/BMImageArray instance

Returns: Return sail.Tensor instance

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    tensor = bmcv.bm_image_to_tensor(BMimg)# here is a sail.Tensor
```

Interface2:

```
def bm_image_to_tensor(self,
                       image: sail.BMImage | sail.BMImageArray,
                       tensor: sail.Tensor)
```

Parameters: * image : sail.BMImage | sail.BMImageArray

BMImage/BMImageArray instance

- tensor : sail.Tensor

Tensor instance

Sample: .. code-block:: python

import sophon.sail as sail

if __name__ == '__main__' : tpu_id = 0 handle = sail.Handle(tpu_id) image_name = "your_image_path" decoder = sail.Decoder(image_name,True,tpu_id) BMimg = decoder.read(handle)# here is a sail.BMImage bmcv = sail.Bmcv(handle) tensor = sail.Tensor(handle,(1920,1080),sail.Dtype.BM_FLOAT32,True,True) bmcv.bm_image_to_tensor(BMimg,tensor)

5.14.3 tensor_to_bm_image

Convert tensor to image.

Interface:

```
def tensor_to_bm_image(self,
                       tensor: sail.Tensor,
                       bgr2rgb: bool = False,
                       layout: str = 'nchw') -> sail.BMImage
```

Parameters: * tensor : sail.Tensor

Tensor instance

- bgr2rgb : bool

Swap color channel, default: False

- layout : str

Layout of the input tensor

Returns: Return BMImage instance

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    tensor = bmcv.bm_image_to_tensor(BMimg)# here is a sail.Tensor
    BMimg2 = bmcv.tensor_to_bm_image(tensor)
```

Interface:

```
def tensor_to_bm_image(self,
                       tensor: sail.Tensor,
                       format: sail.Format) -> sail.BMImage
```

Parameters: * tensor : sail.Tensor

Tensor instance

- format : sail.Format

Format of the BMImage

Returns: Return BMImage instance

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    tensor = bmcv.bm_image_to_tensor(BMimg)# here is a sail.Tensor
    BMimg2 = bmcv.tensor_to_bm_image(tensor, sail.Format.FORMAT_BGR_PLANAR)
```

Interface:

```
def tensor_to_bm_image(self,
                       tensor: sail.Tensor,
                       img: sail.BMImage | sail.BMImageArray,
                       bgr2rgb: bool = False,
                       layout: str = 'nchw')
```

Parameters: * tensor : sail.Tensor

Tensor instance

- img : sail.BMImage

BMImage instance

- bgr2rgb : bool

Swap color channel, default: False

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    tensor = bmcv.bm_image_to_tensor(BMimg)# here is a sail.Tensor
    BMimg2 = sail.BMImage()
    bmcv.tensor_to_bm_image(tensor,BMimg2)
```

Interface:

```
def tensor_to_bm_image(self,
                       tensor: sail.Tensor,
                       img: sail.BMImage | sail.BMImageArray,
                       format: sail.Format)
```

Parameters: * tensor : sail.Tensor

Tensor instance

- img : sail.BMImage

BMImage instance

- format : sail.Format

Format of the BMImage

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    tensor = bmcv.bm_image_to_tensor(BMimg)# here is a sail.Tensor
    BMimg2 = sail.BMImage()
    bmcv.tensor_to_bm_image(tensor,BMimg2, sail.Format.FORMAT_BGR_PLANAR)
```

5.14.4 crop_and_resize

Crop then resize an image or an image array.

Interface:

```
def crop_and_resize(self,
                    input: sail.BMImage|sail.BMImageArray,
                    crop_x0: int,
                    crop_y0: int,
                    crop_w: int,
                    crop_h: int,
                    resize_w: int,
                    resize_h: int,
                    resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST
                    )->sail.BMImage
```

Parameters: * input : sail.BMImage|sail.BMImageArray,

Input image or image array

- crop_x0 : int

Start point x of the crop window

- crop_y0 : int

Start point y of the crop window

- crop_w : int

Width of the crop window

- crop_h : int

Height of the crop window

- resize_w : int

Target width

- resize_h : int

Target height

- resize_alg : bmcv_resize_algorithm

Resize algorithm, default is BMCV_INTER_NEAREST

Returns:

- output : sail.BMImage

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg3 = bmcv.crop_and_resize(BMimg,0,0,BMimg.width(),BMimg.height(),640,640,
↪sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST)
```

5.14.5 crop

Crop an image or an image array with given window.

Interface:

```
def crop(self,
        input: sail.BMImage,
        crop_x0: int,
        crop_y0: int,
        crop_w: int,
        crop_h: int)-> sail.BMImage
```

Parameters: * input : sail.BMImage

Input image

- crop_x0 : int

Start point x of the crop window

- crop_y0 : int

Start point y of the crop window

- crop_w : int

Width of the crop window

- crop_h : int

Height of the crop window

Returns:

- output : sail.BMImage

Output image

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    crop_x0, crop_y0, crop_w, crop_h = 100, 100, 200, 200
    cropped_BMimg = bmcv.crop(BMimg, crop_x0, crop_y0, crop_w, crop_h)
```

Interface:

```
def crop(self, input, crop_x0, crop_y0, crop_w, crop_h):
```

Parameters: * input : sail.BMImageArray

Input image array

- crop_x0 : int

Start point x of the crop window

- crop_y0 : int

Start point y of the crop window

- crop_w : int

Width of the crop window

- crop_h : int

Height of the crop window

Returns:

- output : sail.BMImageArray

Output image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    rects = [
        [0, 0, 40, 40],
        [40, 40, 80, 80],
        #...more
    ]
    cropped_images_list = bmcv.crop(BMimg, rects)
```

5.14.6 resize

Resize an image or an image array

Interface:

```
def resize(self,
        input: sail.BMImage | sail.BMImageArray,
        resize_w: int,
        resize_h: int,
        resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)-> sail.BMImage
```

Parameters: * input : sail.BMImage | sail.BMImageArray

Input image or image array

- resize_w : int

Target width

- resize_h : int

Target height

- resize_alg : bmcv_resize_algorithm

Resize algorithm, default is BMCV_INTER_NEAREST

Returns:

- output : sail.BMImage | sail.BMImageArray

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg_resize = bmcv.resize(BMimg,640,640,resize_alg=sail.bmcv_resize_algorithm.
↪BMCV_INTER_NEAREST)
```

5.14.7 vpp_crop_and_resize

Crop then resize an image or an image array using vpp

Interface:

```
def vpp_crop_and_resize(self,
                        input: sail.BMImage | sail.BMImageArray,
                        crop_x0: int,
                        crop_y0: int,
                        crop_w: int,
                        crop_h: int,
                        resize_w: int,
                        resize_h: int,
                        resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)-> sail.
↪BMImage
```

Parameters: * input : sail.BMImage | sail.BMImageArray

Input image or image array

- crop_x0 : int

Start point x of the crop window

- crop_y0 : int

Start point y of the crop window

- crop_w : int

Width of the crop window

- crop_h : int

Height of the crop window

- resize_w : int

Target width

- resize_h : int

Target height

- resize_alg : bmcv_resize_algorithm

Resize algorithm, default is BMCV_INTER_NEAREST

Returns:

- output : sail.BMImage | sail.BMImageArray

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    crop_x0 = 100
    crop_y0 = 100
    crop_w = 200
    crop_h = 200
    resize_w = 300
    resize_h = 300

    resized_BMimg = bmcv.vpp_crop_and_resize(
        BMimg,
        crop_x0,
        crop_y0,
        crop_w,
        crop_h,
        resize_w,
        resize_h,
        sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST
    )
```

5.14.8 vpp_crop_and_resize_padding

Crop then resize an image or an image array using vpp.

Interface:

```
def vpp_crop_and_resize_padding(self,
                                input: sail.BMImage | sail.BMImageArray,
                                crop_x0: int,
                                crop_y0: int,
                                crop_w: int,
                                crop_h: int,
                                resize_w: int,
                                resize_h: int,
                                padding: PaddingAtrr,
                                resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)-
↪> sail.BMImage
```

Parameters:

- input : sail.BMImage | sail.BMImageArray

Input image or image array

- crop_x0 : int

Start point x of the crop window

- crop_y0 : int

Start point y of the crop window

- crop_w : int

Width of the crop window

- crop_h : int

Height of the crop window

- resize_w : int

Target width

- resize_h : int

Target height

- padding : PaddingAtrr

padding info

- resize_alg : bmcv_resize_algorithm

Resize algorithm, default is BMCV_INTER_NEAREST

Returns: * output : sail.BMImage | sail.BMImageArray

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    paddingatt = sail.PaddingAtrr()
    paddingatt.set_stx(0)
    paddingatt.set_sty(0)
    paddingatt.set_w(640)
    paddingatt.set_h(640)
    paddingatt.set_r(114)
    paddingatt.set_g(114)
    paddingatt.set_b(114)
    BMimg4 = bmcv.vpp_crop_and_resize_padding(BMimg,0,0,BMimg.width(),BMimg.
↪height(),640,640,paddingatt)
```

5.14.9 vpp_crop

Crop an image or an image array with given window using vpp.

Interface:

```
def vpp_crop(self,
             input: sail.BMImage | sail.BMImageArray,
             crop_x0: int,
             crop_y0: int,
             crop_w: int,
             crop_h: int)-> sail.BMImage
```

Parameters: * input : sail.BMImage | sail.BMImageArray

Input image or image array

- crop_x0 : int

Start point x of the crop window

- crop_y0 : int

Start point y of the crop window

- crop_w : int

Width of the crop window

- crop_h : int

Height of the crop window

Returns: * output : sail.BMImage | sail.BMImageArray

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    crop_x0 = 100
    crop_y0 = 100
    crop_w = 200
    crop_h = 200
    BMimg4 = bmcv.vpp_crop(BMimg,crop_x0,crop_y0,crop_w,crop_h)
```

5.14.10 vpp_resize

Resize an image or an image array with interpolation of INTER_NEAREST using vpp.

Interface:

```
def vpp_resize(self,
           input: sail.BMImage | sail.BMImageArray,
           resize_w: int,
           resize_h: int,
           resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)-> sail.
↪BMImage | sail.BMImageArray
```

Parameters:

- input : sail.BMImage | sail.BMImageArray

Input image or image array

- resize_w : int

Target width

- resize_h : int

Target height

- resize_alg : bmcv_resize_algorithm

Resize algorithm, default is BMCV_INTER_NEAREST

Returns:

- output : sail.BMImage | sail.BMImageArray

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg_resize = bmcv.vpp_resize(BMimg,640,640,resize_alg=sail.bmcv_resize_
↪algorithm.BMCV_INTER_NEAREST)
```

Interface:

```
def vpp_resize(self,
               input: sail.BMImage | sail.BMImageArray,
               output: sail.BMImage | sail.BMImageArray,
               resize_w: int,
               resize_h: int,
               resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)
```

Parameters:

- input : sail.BMImage | sail.BMImageArray

Input image

- output : sail.BMImage | sail.BMImageArray

Output image

- resize_w : int

Target width

- resize_h : int

Target height

- resize_alg : bmcv_resize_algorithm

Resize algorithm, default is BMCV_INTER_NEAREST

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
```

(continues on next page)

(continued from previous page)

```
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg_resize = sail.BMImage()
    bmcv.vpp_resize(BMimg,BMimg_resize,640,640,resize_alg=sail.bmcv_resize_algorithm.
↪BMCV_INTER_NEAREST)
```

5.14.11 vpp_resize_padding

Resize an image or an image array with interpolation of INTER_NEAREST using vpp.

Interface:

```
def vpp_resize_padding(self,
                       input: sail.BMImage | sail.BMImageArray,
                       resize_w: int,
                       resize_h: int,
                       padding: PaddingAtrr,
                       resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST)-> sail.
↪BMImage | sail.BMImageArray
```

Parameters:

- input : sail.BMImage | sail.BMImageArray

Input image or image array

- resize_w : int

Target width

- resize_h : int

Target height

- padding : PaddingAtrr

padding info

- resize_alg : bmcv_resize_algorithm

Resize algorithm, default is BMCV_INTER_NEAREST

Returns:

- output : sail.BMImage | sail.BMImageArray

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name, True, tpu_id)
    BMimg = decoder.read(handle)  # here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    paddingatt = sail.PaddingAtrr()
    paddingatt.set_stx(0)
    paddingatt.set_sty(0)
    paddingatt.set_w(640)
    paddingatt.set_h(640)
    paddingatt.set_r(114)
    paddingatt.set_g(114)
    paddingatt.set_b(114)
    BMimg4 = bmcv.vpp_resize_padding(BMimg,640,640,paddingatt)
```

5.14.12 warp

Applies an affine transformation to an image or an image array.

Interface:

```
def warp(self,
        input: sail.BMImage | sail.BMImageArray,
        matrix: 2d list,
        use_bilinear: int = 0,
        similar_to_opencv: bool = False)-> sail.BMImage | sail.BMImageArray
```

Parameters:

- input : sail.BMImage | sail.BMImageArray

Input image or image array

- matrix: 2d list

2x3 transformation matrix

- use_bilinear: int

Whether to use bilinear interpolation, default to 0 using nearest neighbor interpolation, 1 being bilinear interpolation

- similar_to_opencv: bool

Whether to use the interface aligning the affine transformation interface of OpenCV

Returns:

- output : sail.BMImage | sail.BMImageArray

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    rotated_matrix = [[0.9996914396,-0.02484,0],[0.02484,0.9996914396,0]]
    BMimg6 = bmcv.warp(BMimg, rotated_matrix)
```

Interface:

```
def warp(self,
        input: sail.BMImage | sail.BMImageArray,
        output: sail.BMImage | sail.BMImageArray,
        matrix: 2d list,
        use_bilinear: int = 0,
        similar_to_opencv: bool = False)-> int
```

Parameters:

- input : sail.BMImage | sail.BMImageArray

Input image or image array

- output : sail.BMImage | sail.BMImageArray

Output image or image array

- matrix: 2d list

2x3 transformation matrix

- use_bilinear: int

Whether to use bilinear interpolation, default to 0 using nearest neighbor interpolation, 1 being bilinear interpolation

- similar_to_opencv: bool

Whether to use the interface aligning the affine transformation interface of OpenCV

Returns:

0 for success and others for failure

Sample:

```
import sophon.sail as sail
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    rotated_matrix = [[0.9996914396,-0.02484,0],[0.02484,0.9996914396,0]]
    output = sail.BMImage()
    ret = bmcv.warp(BMimg, output,rotated_matrix)
```

5.14.13 convert_to

Applies a linear transformation to an image or an image array.

Interface:

```
def convert_to(self,
               input: sail.BMImage | sail.BMImageArray,
               alpha_beta: tuple)-> sail.BMImage | sail.BMImageArray
```

Parameters:

- input : sail.BMImage | sail.BMImageArray

Input image or image array

- alpha_beta: tuple

(a0, b0), (a1, b1), (a2, b2) factors

Returns:

- output : sail.BMImage | sail.BMImageArray

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    BMimg5 = bmcv.convert_to(BMimg, alpha_beta)
```

Interface:

```
def convert_to(self,
               input: sail.BMImage | sail.BMImageArray,
               output: sail.BMImage | sail.BMImageArray,
               alpha_beta: tuple)
```

Parameters:

- input : sail.BMImage | sail.BMImageArray

Input image or image array

- output : sail.BMImage | sail.BMImageArray

Output image or image array

- alpha_beta: tuple

(a0, b0), (a1, b1), (a2, b2) factors

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    alpha_beta = (1.0/255,0),(1.0/255,0),(1.0/255,0)
    BMimg5 = sail.BMImage()
    bmcv.convert_to(BMimg, BMimg5,alpha_beta)
```

5.14.14 yuv2bgr

Convert an image or an image array from YUV to BGR.

Interface:

```
def yuv2bgr(self, input: sail.BMImage | sail.BMImageArray)-> sail.BMImage | sail.
↪BMImageArray
```

Parameters:

- input : sail.BMImage | sail.BMImageArray

Input image or image array

Returns:

- output : sail.BMImage | sail.BMImageArray

Output image or image array

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    BMimg5 = bmcv.yuv2bgr(BMimg)
```

5.14.15 rectangle

Draw a rectangle on input image.

Interface:

```
def rectangle(self,
              image: sail.BMImage,
              x0: int,
              y0: int,
              w: int,
              h: int,
              color: tuple,
              thickness: int = 1)-> int

def rectangle(self,
                image: sail.bm_image,
                x0: int,
                y0: int,
                w: int,
                h: int,
                color: tuple,
                thickness: int = 1)-> int
```

Parameters:

- image : sail.BMImage | sail.bm_image

Input image

- x0 : int

Start point x of rectangle

- y0 : int

Start point y of rectangle

- w : int

Width of rectangle

- h : int

Height of rectangle

- color : tuple

Color of rectangle

- thickness : int

Thickness of rectangle, default: 1

Returns:

- process_status : int

0 for success and others for failure

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.rectangle(BMimg, 20, 20, 600, 600,(0,0,255),2)
```

5.14.16 fillRectangle

Fill a rectangle on input image.

Interface:

```
def fillRectangle(self,
            image: sail.BMImage,
            x0: int,
            y0: int,
            w: int,
            h: int,
            color: tuple)-> int

def fillRectangle(self,
            image: sail.bm_image,
            x0: int,
            y0: int,
            w: int,
            h: int,
            color: tuple)-> int
```

Parameters:

- image : sail.BMImage | sail.bm_image

Input image

- x0 : int

Start point x of rectangle

- y0 : int

Start point y of rectangle

- w : int

Width of rectangle

- h : int

Height of rectangle

- color : tuple

Color of rectangle

Returns:

- process_status : int

0 for success and others for failure

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.fillRectangle(BMimg, 20, 20, 600, 600,(0,0,255))
```

5.14.17 imwrite

Save the image to the specified file.

Interface:

```
def imwrite(self, file_name: str, image: sail.BMImage)-> int

def imwrite(self, file_name: str, image: sail.BMImage, params: list) -> int

def imwrite(self, file_name: str, image: sail.bm_image)-> int
```

Parameters:

- file_name : str

Name of the file

- output : sail.BMImage | sail.bm_image

Image to be saved

- params : list

Format parameters for saving the image. Must ensure the validity of the parameters.

Returns:

- process_status : int

0 for success and others for failure

Sample:

```
import sophon.sail as sail
import cv2

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmcv.imwrite("{}_{}.jpg".format(BMimg.width(),BMimg.height()),BMimg)
    bmcv.imwrite("{}_{}_qual95.jpg".format(BMimg.width(),BMimg.height()),BMimg,[cv2.
↪IMWRITE_JPEG_QUALITY, 95])
```

5.14.18 imread

Read and decode one image files and supports hard decoding only for JPEG baseline format. For other formats, such as PNG and BMP, soft decoding is used. The returned BMImage for JPEG baseline images keeps YUV color space, and the pixel format depends on the sampling information in the file like YUV420. The returned BMImage for other formats will maintain the corresponding color space of their input.

Interface:

```
def imread(self, filename: str) -> BMImage
```

Parameters:

- filename : str

Name of file to be read.

Returns:

- output : sail.BMImage

The decoded image.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    bmcv = sail.Bmcv(handle)
    filename = "your_image_path"
    BMimg = bmcv.imread(filename)
```

5.14.19 get_handle

Get Handle instance.

Interface:

```
def get_handle(self)-> sail.Handle
```

Returns:

- handle: sail.Handle

Handle instance

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    handle1 = bmcv.get_handle()
```

5.14.20 crop_and_resize_padding

Crop then resize an image.

Interface:

```
def crop_and_resize_padding(self,
                            input: sail.BMImage,
                            crop_x0: int,
                            crop_y0: int,
```

(continues on next page)

(continued from previous page)

```
                    crop_w: int,
                    crop_h: int,
                    resize_w: int,
                    resize_h: int,
                    padding: PaddingAtrr,
                    resize_alg: bmcv_resize_algorithm = BMCV_INTER_NEAREST
                    )-> sail.BMImage
```

Parameters:

- input : sail.BMImage

Input image

- crop_x0 : int

Start point x of the crop window

- crop_y0 : int

Start point y of the crop window

- crop_w : int

Width of the crop window

- crop_h : int

Height of the crop window

- resize_w : int

Target width

- resize_h : int

Target height

- padding : PaddingAtrr

padding info

- resize_alg : bmcv_resize_algorithm

Resize algorithm, default is BMCV_INTER_NEAREST

Returns:

- output : sail.BMImage

Output image

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
tpu_id = 0
handle = sail.Handle(tpu_id)
image_name = "your_image_path"
decoder = sail.Decoder(image_name,True,tpu_id)
BMimg = decoder.read(handle)# here is a sail.BMImage
bmcv = sail.Bmcv(handle)
crop_x0 = 100
crop_y0 = 100
crop_w = 200
crop_h = 200
resize_w = 300
resize_h = 300

paddingatt = sail.PaddingAtrr()
paddingatt.set_stx(0)
paddingatt.set_sty(0)
paddingatt.set_w(300)
paddingatt.set_h(300)
paddingatt.set_r(114)
paddingatt.set_g(114)
paddingatt.set_b(114)
padded_BMimg = bmcv.crop_and_resize_padding(
    BMimg,
    crop_x0,
    crop_y0,
    crop_w,
    crop_h,
    resize_w,
    resize_h,
    paddingatt,
    sail.bmcv_resize_algorithm.BMCV_INTER_NEAREST
)
```

5.14.21 convert_format

Convert input to output format or convert an image to BGR PLANAR format.

Interface:

```
def convert_format(self, input: sail.BMImage, output: sail.BMImage)
```

Parameters:

- input : sail.BMImage

BMimage instance

- output : sail.BMImage

output image

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    output = sail.BMImage()
    bmcv.convert_format(BMimg,output)
```

Interface:

```
def convert_format(self, input: sail.BMImage)-> sail.BMImage
```

Parameters:

- input : sail.BMImage

BMimage instance

Returns:

- output : sail.BMImage

output image

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    output = bmcv.convert_format(BMimg,sail.FORMAT_BGR_PLANAR)
```

5.14.22 vpp_convert_format

Convert input to output format or convert an image to BGR PLANAR format using vpp.

Interface:

```
def vpp_convert_format(self, input: sail.BMImage, output: sail.BMImage)
```

Parameters:

- input : sail.BMImage

BMimage instance

- output : sail.BMImage

output image

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    output = sail.BMImage()
    bmcv.vpp_convert_format(BMimg,output)
```

Interface:

```
def vpp_convert_format(self, input):
```

Parameters:

- input : sail.BMImage

BMimage instance

Returns:

- output : sail.BMImage

output image

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    output = bmcv.vpp_convert_format(BMimg,sail.FORMAT_BGR_PLANAR)
```

5.14.23 putText

Draws a text on the image.

Supported pixel format for input BMImage: FORMAT_GRAY, FORMAT_YUV420P, FORMAT_YUV422P, FORMAT_YUV444P, FORMAT_NV12, FORMAT_NV21, FORMAT_NV16, FORMAT_NV61

Interface:

```
def putText(self,
            input: sail.BMImage,
            text: str,
            x: int,
            y: int,
            color: tuple,
            fontScale: float,
            thickness: int)-> int

def putText(self,
            input: sail.bm_image,
            text: str,
            x: int,
            y: int,
            color: tuple,
            fontScale: float,
            thickness: int)-> int
```

Parameters:

- input : sail.BMImage

BMimage instance

- text: str

Text to write on an image.

- x: int

Start point x

- y: int

Start point y

- color : tuple

Color of text

- fontScale : float

Size of font

- thickness : int

Thickness of text

Returns:

- process_status : int

0 for success and others for failure

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    bgr_img = decoder.read(handle)
    bmcv = sail.Bmcv(handle)
    yuv_img = bmcv.convert_format(bgr_img, sail.FORMAT_YUV420P)
    ret = bmcv.putText(yuv_img, "some text" , 20, 20, [0,0,255], 1.4, 2)
    assert ret == 0
```

5.14.24 image_add_weighted

Calculates the weighted sum of two images

Interface:

```
def image_add_weighted(self,
                       input0: sail.BMImage,
                       alpha: float,
                       input1: float,
                       beta: float,
                       gamma: float,
                       output: BMImage)
```

Parameters:

- input0 : sail.BMImage

BMimage instance.

- alpha : float

alpha instance.

- input1 : sail.BMImage

BMImage instance.

- beta: float

beta instance.

- gamma: float

gamma instance.

- output: BMImage

result BMImage, output = input1 * alpha + input2 * beta + gamma.

Sample1:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name1 = "your_image_path1"
    image_name2 = "your_image_path2"
    decoder1 = sail.Decoder(image_name1,True,tpu_id)
    decoder2 = sail.Decoder(image_name2,True,tpu_id)
    BMimg1 = decoder1.read(handle)# here is a sail.BMImage
    BMimg2 = decoder2.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmg = sail.BMImage()
    bmcv.image_add_weighted(BMimg1,0.5,BMimg2,0.5,0.5,bmg)
```

Interface:

```
def image_add_weighted(self,
                       input0: sail.BMImage,
                       alpha: float,
                       input1: sail.BMImage,
                       beta: float,
                       gamma: float)-> BMImage
```

Parameters:

- input0 : sail.BMImage

BMimage instance.

- alpha : float

alpha instance.

- input1 : sail.BMImage

BMImage instance.

- beta: float

beta instance.

- gamma: float

gamma instance.

Returns:

- output: BMImage

result BMImage, output = input1 * alpha + input2 * beta + gamma.

Sample2:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name1 = "your_image_path1"
    image_name2 = "your_image_path2"
    decoder1 = sail.Decoder(image_name1,True,tpu_id)
    decoder2 = sail.Decoder(image_name2,True,tpu_id)
    BMimg1 = decoder1.read(handle)# here is a sail.BMImage
    BMimg2 = decoder2.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmg = bmcv.image_add_weighted(BMimg1,0.5,BMimg2,0.5,0.5)
```

5.14.25 image_copy_to

Copy the input to the output.

Interface:

```
def image_copy_to(self,
                  input: BMImage|BMImageArray,
                  output: BMImage|BMImageArray,
                  start_x: int,
                  start_y: int)
```

Parameters:

- input: BMImage|BMImageArray

Input image or image array.

- output: BMImage|BMImageArray

Output image or image array.

- start_x: int

Point start x.

- start_y: int

Point start y.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name1 = "your_image_path1"
    image_name2 = "your_image_path2"
    decoder1 = sail.Decoder(image_name1,True,tpu_id)
    decoder2 = sail.Decoder(image_name2,True,tpu_id)
    BMimg1 = decoder1.read(handle)# here is a sail.BMImage
    BMimg2 = decoder2.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmcv.image_copy_to(BMimg1,BMimg2,0,0)
```

5.14.26 image_copy_to_padding

Copy the input to the output width padding.

Interface:

```
def image_copy_to_padding(self,
                          input: BMImage|BMImageArray,
                          output: BMImage|BMImageArray,
                          padding_r: int,
                          padding_g: int,
                          padding_b: int,
                          start_x: int,
                          start_y: int)
```

Parameters:

- input: BMImage|BMImageArray

Input image or image array.

- output: BMImage|BMImageArray

Output image or image array.

- padding_r: int

r value for padding.

- padding_g: int

g value for padding.

- padding_b: int

b value for padding.

- start_x: int

point start x.

- start_y: int

point start y.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name1 = "your_image_path1"
    image_name2 = "your_image_path2"
    decoder1 = sail.Decoder(image_name1,True,tpu_id)
    decoder2 = sail.Decoder(image_name2,True,tpu_id)
    BMimg1 = decoder1.read(handle)# here is a sail.BMImage
    BMimg2 = decoder2.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bmcv.image_copy_to_padding(BMimg1,BMimg2,128,128,128,0,0)
```

5.14.27 nms

Do nms use tpu.

Note: For details about whether this operator in current SDK supports BM1688, check the page "BMCV API" in 《Multimedia User Guide》 .

Interface:

```
def nms(self,
        input: [float, float, float, float, float],
        threshold: float)-> numpy.ndarray[Any, numpy.dtype[numpy.float32]]
```

Parameters:

- input: [float, float, float, float, float]

input proposal array, shape must be (self, n,5) n<56000, proposal:[left,top,right,bottom,score].

- threshold: float

nms threshold.

Returns:

return nms result, numpy.ndarray[Any, numpy.dtype[numpy.float32]]

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
```

(continues on next page)

(continued from previous page)

```
handle = sail.Handle(tpu_id)
bmcv = sail.Bmcv(handle)
input_boxes = np.array([
[50, 50, 100, 100, 0.9],
[60, 60, 110, 110, 0.85],
[200, 200, 250, 250, 0.7],
[130, 50, 180, 100, 0.8],
[205, 205, 255, 255, 0.75]
])
nms_threshold = 0.5
selected_boxes  = bmcv.nms(input_boxes, nms_threshold)
print(selected_boxes)
```

5.14.28 drawPoint

Draw Point on input image.

Interface:

```
def drawPoint(self,
              image: BMImage,
              center: Tuple[int, int],
              color: Tuple[int, int, int],
              radius: int) -> int:

def drawPoint(self,
              image: bm_image,
              center: Tuple[int, int],
              color: Tuple[int, int, int],
              radius: int) -> int:
```

Parameters:

- image: BMImage | bm_image

Input image

- center: Tuple[int, int]

center of point, (point_x, point_y)

- color: Tuple[int, int, int],

color of drawn, (b,g,r)

- radius: int

Radius of drawn

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.drawPoint(BMimg, (320, 320), (0,255,255),2)
```

5.14.29 warp_perspective

Applies a perspective transformation to an image.

Interface:

```
def warp_perspective(input: BMImage,
                     coordinate: [[int,int],[int,int],[int,int],[int,int]],
                     output_width: int,
                     output_height: int,
                     format: bm_image_format_ext = FORMAT_BGR_PLANAR,
                     dtype: bm_image_data_format_ext = DATA_TYPE_EXT_1N_BYTE,
                     use_bilinear: int = 0 ) -> BMImage:
```

Parameters:

- input: BMImage

Input image

coordinate: [[int,int],[int,int],[int,int],[int,int]]

- **Original coordinate, like(left_top.x, left_top.y), (right_top.x, right_top.y),** (left_bottom.x, left_bottom.y), (right_bottom.x, right_bottom.y)
- output_width: int

Output width

- output_height: int

Output height

- format : bm_image_format_ext

Output image format, Only FORMAT_BGR_PLANAR,FORMAT_RGB_PLANAR

- dtype: bm_image_data_format_ext

Output image dtype, Only DATA_TYPE_EXT_1N_BYTE,DATA_TYPE_EXT_4N_BYTE

- use_bilinear: int

Bilinear use flag.

Returns:

Output image

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    img = bmcv.warp_perspective(BMimg, ((100, 100), (540, 100), (100, 380), (540, 380)),640,
↪640)
```

5.14.30 get_bm_data_type

Convert bm_image_data_format_ext to bm_data_type_t

Interface:

```
def get_bm_data_type((self, format: sail.ImgDtype) -> sail.Dtype
```

Parameters:

- format: sail.ImgDtype

The type that needs to be converted.

Returns:

- ret: sail.Dtype

The converted type.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    type = bmcv.get_bm_data_type(sail.DATA_TYPE_EXT_FLOAT32)
```

5.14.31 get_bm_image_data_format

Convert bm_data_type_t to bm_image_data_format_ext

Interface:

```
def get_bm_image_data_format(self, dtype: bm_data_type_t) -> bm_image_data_
↪format_ext:
```

Parameters:

- dtype: sail.Dtype

The sail.Dtype that needs to be converted.

Returns:

- ret: sail.ImgDtype

The converted type.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    type = bmcv.get_bm_image_data_format(sail.BM_FLOAT32)
```

5.14.32 imdecode

Load image from system memory

Interface:

```
def imdecode(self, data_bytes: bytes) -> BMImage:
```

Parameters:

- data_bytes: bytes

image data bytes in system memory

Returns:

return decoded image

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    with open(image_name, 'rb') as image_file:
        image_data_bytes = image_file.read()
    bmcv = sail.Bmcv(handle)
    src_img = bmcv.imdecode(image_data_bytes)
```

5.14.33 imencode

Encode an BMimage and return the encoded data.

Interface:

```
def imencode(self, ext: str, img: BMImage) -> numpy.ndarray:
```

Parameters:

- ext: str

Input parameter. Image encoding format, supported formats include ".jpg" , ".png" , etc.

- img: BMImage

Input parameter. Input BMImage, only FORMAT_BGR_PACKED, DATA_TYPE_EXT_1N_BYTE pictures are supported.

Return:

- ret: numpy.array

The data that is encoded and placed in system memory.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.imencode(".jpg",BMimg)
```

5.14.34 fft

1d or 2d fft (only real part, or real part and imaginary part)

Note: For details about whether this operator in current SDK supports BM1688, check the page "BMCV API" in 《Multimedia User Guide》.

Interface:

```
def fft(self, forward: bool, input_real: Tensor)-> list[Tensor]:
```

Parameters:

- forward: bool

positive transfer

- input_real: Tensor

input tensor

Returns:

return list[Tensor], The real and imaginary part of output

Interface:

```
def fft(self,
        forward: bool,
        input_real: Tensor,
        input_imag: Tensor) -> list[Tensor]:
```

Parameters:

- forward: bool

positive transfer

- input_real: Tensor

input tensor real part

- input_imag: Tensor

input tensor imaginary part

Returns:

return list[Tensor], The real and imaginary part of output

5.14.35 mat_to_bm_image

transform opencv mat to sail BMImage.

Interface:

```
def mat_to_bm_image(self, mat: numpy.ndarray[numpy.uint8]) -> BMImage:
```

Parameters:

- mat : numpy

input opencv mat.

Returns:

- ret: sail.BMImage

return sail.BMImage.

Sample:

```
import sophon.sail as sail
import cv2

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    bmcv = sail.Bmcv(handle)
    opencv_mat = cv2.imread(image_name)
    sail_bm_image = bmcv.mat_to_bm_image(opencv_mat)
```

Interface:

```
def mat_to_bm_image(self, mat: numpy.ndarray[numpy.uint8], img: BMImage) -> int:
```

Parameters:

- mat : numpy

input opencv mat.

- img : sail.BMImage

output sail.BMImage.

Returns:

- ret: int

returns 0 if success.

Sample:

```
import sophon.sail as sail
import cv2

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    bmcv = sail.Bmcv(handle)
    opencv_mat = cv2.imread(image_name)
    BMimg2 = sail.BMImage()
    ret = bmcv.mat_to_bm_image(opencv_mat,BMimg2)
```

5.14.36 polylines

It can be realized to draw one or more line segments on an image, so that the function of drawing polygons can be realized, and the color and width of the line can be specified.

Interface:

```
def polylines(self, image: BMImage, pts: list[list[tuple(int, int)]], isClosed: bool, color:F
↪tuple(int, int, int), thickness: int = 1, shift: int = 0) -> int:
```

Parameters:

- img : sail.BMImage

Input BMImage.

- pts : list[list[tuple(int, int)]]

The starting point and end point coordinates of the line segment, multiple coordinate points can be entered. The upper left corner of the image is the origin, extending to the right in the x direction and extending down in the y direction.

- isClosed : bool

Whether the graph is closed.

- color : tuple(int, int, int)

The color of the line is the value of the three RGB channels.

- thickness : int

The width of the lines is recommended to be even for YUV format images.

- shift : int

Polygon scaling multiple. Default is not scaling. The scaling factor is(1/2)^shift。

Returns:

- ret: int

returns 0 if success.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg1 = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    bm = bmcv.vpp_convert_format(BMimg1,sail.FORMAT_YUV444P)
    ret = bmcv.polylines(bm,[[(10,20),(40,80)]],True,[128,128,128])
```

5.14.37 mosaic

This interface is used to print one or more mosaics on an image.

Interface:

```
def mosaic(self, mosaic_num: int, img: BMImage, rects: list[list[int,int,int,int]], is_
↪expand:int) -> int
```

Parameters:

- mosaic_num : int

Number of mosaics, length of list in rects.

- img : sail.BMImage

Input BMImage.

- rects : list[list[int,int,int,int]]

Multiple Mosaic positions, the parameters in each element in the list are [Mosaic at X-axis start point, Mosaic at Y-axis start point, Mosaic width, Mosaic height].

- is_expand : int

Whether to expand the column. A value of 0 means that the column is not expanded, and a value of 1 means that a macro block (8 pixels) is expanded around the original Mosaic.

Returns:

- ret: int

returns 0 if success.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg1 = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    ret = bmcv.mosaic(2, BMimg1, [[10,10,100,2000],[500,500,1000,100]], 1)
```

5.14.38 gaussian_blur

This interface is used for image Gaussian filtering. **Note: The previous SDK does not support BM1684X. For details about whether the current SDK supports BM1684X, check the page "BMCV API" in《Multimedia User Guide》. ***

Interface:

```
def gaussian_blur(self, input: BMImage, kw: int, kh : int, sigmaX : float, sigmaY : float = 0.
↪0) -> BMImage:
```

Parameters:

- input : sail.BMImage

Input BMImage.

- kw : int

The size of kernel in the width direction.

- kh : int

The size of kernel in the height direction.

- sigmaX : float

Gaussian kernel standard deviation in the X direction.

- sigmaY : float

Gaussian kernel standard deviation in the Y direction.Default is 0, which means that it is the same standard deviation as the Gaussian kernel in the X direction.

Returns:

- output : sail.BMImage

Returns a Gaussian filtered image.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    handle = sail.Handle(0)
```

(continues on next page)

(continued from previous page)

```
bmcv = sail.Bmcv(handle)


bmimg = sail.BMImage()
decoder = sail.Decoder("your_img.jpg",True,0)
bmimg = decoder.read(handle)

print(bmimg.format())
output = bmcv.gaussian_blur(bmimg, 3, 3, 0.1)

bmcv.imwrite("out.jpg",output)
```

5.14.39 transpose

The interface can realize the transposition of image width and height.

Interface1:

```
def transpose(self, src: sail.BMImage) -> sail.BMImage:
```

Parameters1:

- src : sail.BMImage

Input BMImage.

Returns2:

- output: sail.BMImage:

output sail.BMImage.

Interface2:

```
def transpose(self, src: sail.BMImage, dst: sail.BMImage) -> int:
```

Parameters2:

- src : sail.BMImage

Input BMImage.

- dst : sail.BMImage

output sail.BMImage.

Returns2:

- ret : int

returns 0 if success.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':

    handle = sail.Handle(0)
    bmcv = sail.Bmcv(handle)
    bmimg = sail.BMImage()
    decoder = sail.Decoder("your_img.jpg",True,0)
    bmimg = decoder.read(handle)
    img = bmcv.convert_format(bmimg,sail.Format.FORMAT_GRAY)
    print("readed")
    print(img.format())
    output = bmcv.transpose(img)

    bmcv.imwrite("out.jpg",output)
```

5.14.40 watermark_superpose

Implement adding multiple watermarks to images

接口形式:

```
def watermark_superpose(self,
image: sail.BMImage,
water_name:string,
bitmap_type: int,
pitch: int,
rects: list[list[int]],
color: tuple
        )->int
```

参数说明:

- Image: save BMImage

Input image

- Watername: string

Watermark file path

- Bitmap_type: int

Input parameters. Watermark type, a value of 0 indicates that the watermark is an 8-bit data type (with transparency information), and a value of 1 indicates that the watermark is a 1-bit data type (without transparency information).

- Pitch: int

Input parameters. The number of bytes per line in a watermark file can be understood as the width of the watermark.

- Rects: list

Input parameters. Watermark position, including the starting point and width/height of each watermark.

- Color: tuple

Input parameters. The color of the watermark.

Return value description:

- Ret: int

Whether the return was successful

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path1"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg1 = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    water_name = 'your_watermark_path'
    ret = bmcv.watermark_superpose(BMimg1,water_name,0,117,[[0,0,117,79],[0,90,117,79]],
↪[128,128,128])
    bmcv.imwrite("aafaa.jpg",BMimg1)
```

5.14.41 Sobel

Soble operator for edge detection.

Note: For details about whether this operator in current SDK supports BM1684X/BM1688, check the page "BMCV API" in 《Multimedia User Guide》.

Interface1:

```
def Sobel(self, input: BMImage, output: BMImage, dx: int, dy: int, ksize: int = 3, scale:F
↪float = 1, delta: float = 0) -> int:
```

Parameters1:

- input

Input BMImage

- output

Output BMImage

- dx

Order of the derivative x.

- dy

Order of the derivative y

- ksize

ize of the extended Sobel kernel; it must be -1, 1, 3, 5, or 7. -1 means 3x3 Scharr filter will be used.

- scale

Optional scale factor for the computed derivative values; by default, no scaling is applied

- delta

Optional delta value that is added to the results prior to storing them in dst.

Returns1:

- ret: int

returns 0 if success.

Interface2:

```
def Sobel(self, input: BMImage, dx: int, dy: int, ksize: int = 3, scale: float = 1, delta: float
↪= 0) -> BMImage:
```

Parameters2:

- input

Input BMImage

- dx

Order of the derivative x.

- dy

Order of the derivative y

- ksize

ize of the extended Sobel kernel; it must be -1, 1, 3, 5, or 7. -1 means 3x3 Scharr filter will be used.

- scale

Optional scale factor for the computed derivative values; by default, no scaling is applied

- delta

Optional delta value that is added to the results prior to storing them in dst.

Returns2:

- output: BMImage

returns porcessed BMImage.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    bmcv = sail.Bmcv(handle)
    bmimg = sail.BMImage()
    decoder = sail.Decoder("your_img.jpg",True,tpu_id)
    bmimg = decoder.read(handle)

    print(bmimg.format())
    output = bmcv.Sobel(bmimg, 1, 1)
    bmcv.imwrite("out.jpg",output)
```

5.14.42 drawLines

Draws multiple lines on a given image.

Note: For details about whether this operator in current SDK supports BM1684X, check the page "BMCV API" in 《Multimedia User Guide》.

Interface1:

```
def drawLines(self, image: BMImage, start_points: list[tuple[int, int]], end_points:F
↪list[tuple[int, int]], line_num: int, color: tuple[int, int, int], thickness: int) -> int:
```

Parameters:

- image : sail.BMImage

Input image.

- start_points : list[tuple[int, int]]

A list of starting point coordinates for the line segments, with the top-left corner of the image as the origin, extending to the right for the x-direction and down for the y-direction.

- end_points : list[tuple[int, int]]

A List of ending point coordinates for the lines, which must have the same length as the list of starting points.

- line_num : int

The number of line segments, which must be the same as the length of the starting and ending points lists.

- color : tuple[int, int, int]

The color of the line segments, in RGB format.

- thickness : int

The thickness of the line segments.

Returns:

- ret: int

Returns 0 upon success.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    image_name = "your_image_path"
    decoder = sail.Decoder(image_name,True,tpu_id)
    BMimg1 = decoder.read(handle)# here is a sail.BMImage
    bmcv = sail.Bmcv(handle)
    start_points = [(10, 10), (30, 30)]
    end_points = [(20, 20), (40, 40)]
    line_num = 2
    color = (255, 0, 0)
    thickness = 2
    bm = bmcv.vpp_convert_format(BMimg1,sail.FORMAT_YUV444P)
    ret = bmcv.drawLines(bm, start_points, end_points, line_num,color, thickness)
```

5.14.43 stft

Short-Time Fourier Transform(STFT)

Note: For details about whether this operator in current SDK supports BM1684X, check the page "BMCV API" in 《Multimedia User Guide》.

Interface1:

```
stft(self, input_real: numpy.ndarray, input_imag: numpy.ndarray, real_input: bool,F
↪normalize: bool, n_fft: int, hop_len: int,
        pad_mode: int, win_mode: int) -> tuple[numpy.ndarray, numpy.ndarray]

stft(self, input_real: Tensor, input_imag: Tensor, real_input: bool, normalize: bool, n_fft:F
↪int, hop_len: int,
        pad_mode: int, win_mode: int) -> tuple[Tensor, Tensor]
```

Parameters:

- **input_real: numpy.ndarray or Tensor** The real part of the input signal.
- **input_imag: numpy.ndarray or Tensor** The imaginary part of the input signal.
- **real_input: bool** A flag indicating whether to use only real input.
- **normalize: bool** A flag indicating whether to normalize the output.

- **n_fft: int** The number of FFT points used in the STFT computation.
- **hop_len: int** The step size for sliding the window.
- **pad_mode: int** The padding mode for the input signal. 0 indicates CONSTANT padding, and 1 indicates REFLECT padding.
- **win_mode: int** The type of window function. 0 represents the HANN window, and 1 represents the HAMM window.

Returns:

- **result: tuple[numpy.ndarray, numpy.ndarray]** Returns the real part and imaginary part of the output.

Sample:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    random_array1 = np.random.rand(2, 4096).astype('float32')
    random_array2 = np.random.rand(2, 4096).astype('float32')
    bmcv = sail.Bmcv(handle)
    input_real = sail.Tensor(handle, random_array1, True)
    input_imag = sail.Tensor(handle, random_array2, True)
    real_input = False
    normalize = True
    n_fft = 1024
    hop_len = 256
    pad_mode = 0
    win_mode = 1
    stft_R, stft_I = bmcv.stft(input_real, input_imag, real_input, normalize, n_fft, hop_
↪len, pad_mode, win_mode)
```

5.14.44 istft

Inverse Short-Time Fourier Transform(ISTFT)

Note: For details about whether this operator in current SDK supports BM1684X, check the page "BMCV API" in 《Multimedia User Guide》 .

Interface1:

```
istft(self, input_real: numpy.ndarray, input_imag: numpy.ndarray, real_input: bool,F
↪normalize: bool, L: int, hop_len: int,
    pad_mode: int, win_mode: int) -> tuple[numpy.ndarray, numpy.ndarray]:

istft(self, input_real: Tensor, input_imag: Tensor, real_input: bool, normalize: bool, L: int,F
↪hop_len: int,
    pad_mode: int, win_mode: int) -> tuple[Tensor, Tensor]:
```

Parameters:

- **input_real: numpy.ndarray or Tensor** The real part of the input signal.
- **input_imag: numpy.ndarray or Tensor** The imaginary part of the input signal.
- **real_input: bool** Indicates whether the output signal is real; false for complex, true for real.
- **normalize: bool** Whether to normalize the output.
- **L: int** The length of the original time-domain signal.
- **hop_len: int** The step size for sliding the window; must match the value used during STFT computation.
- **pad_mode: int** The padding mode for the input signal; must match the value used during STFT computation.
- **win_mode: int** The type of window function; must match the value used during STFT computation.

Returns:

- **result: tuple[numpy.ndarray, numpy.ndarray]** Returns the real part and imaginary part of the output.

Sample:

```
import sophon.sail as sail
import numpy as np

if __name__ == '__main__':
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    random_array1 = np.random.rand(2, 513, 17).astype('float32')
    random_array2 = np.random.rand(2, 513, 17).astype('float32')
    bmcv = sail.Bmcv(handle)
    input_real = sail.Tensor(handle, random_array1, True)
    input_imag = sail.Tensor(handle, random_array2, True)
    real_input = False
    normalize = True
    L = 4096
    hop_len = 256
    pad_mode = 0
    win_mode = 1
    istft_R, istft_I = bmcv.istft(input_real, input_imag, real_input, normalize, 4096, hop_
↪len, pad_mode, win_mode)
```

5.14.45 bmcv_overlay

Add a watermark with a transparent channel to the image. Only support BM1688, CV186AH。

Interface:

```
def bmcv_overlay(self, image: BMImage, overlay_info: list[list[int]], overlay_image:F
↪list[BMImage]) -> int:
```

Parameters:

- image : sail.BMImage

input/output image

- overlay_info: list[list[int]]

The position and size information of a set of watermarks in the format of [x,y,w,h]

- overlay_image: list[BMImage]

A group of watermark, only support sail.Format.FORMAT_ARGB_PACKED

Returns:

- ret: int

0 for success

Note:

You need to make sure that all rectangles in overlay_info do not overlap

Sample1:

Read RGBA images directly as watermarks

```
import sophon.sail as sail
import cv2

if __name__ == '__main__':

    handle = sail.Handle(0)

    decoder = sail.Decoder("pics/1.jpg")
    image_org = decoder.read(handle)

    bmcv = sail.Bmcv(handle)

    buffer = cv2.imread("pics/icon.png", cv2.IMREAD_UNCHANGED)
    buffer_x = 100
    buffer_y = 50
    buffer_w = 100
    buffer_h = 30
```

(continues on next page)

(continued from previous page)

```
    buffer_x1 = 500
    buffer_y1 = 200
    buffer_w1 = 60
    buffer_h1 = 70

    buffer = cv2.resize(buffer, (buffer_w, buffer_h))
    buffer1 = cv2.resize(buffer, (buffer_w1, buffer_h1))

    overlay_images = []
    overlay_info = []
    overlay_images.append(sail.BMImage(handle, buffer, buffer_h, buffer_w, sail.
↪Format.FORMAT_ARGB_PACKED, sail.ImgDtype.DATA_TYPE_EXT_1N_
↪BYTE))
    overlay_info.append([buffer_x,buffer_y,buffer_w, buffer_h])
    overlay_images.append(sail.BMImage(handle, buffer1, buffer_h1, buffer_w1, sail.
↪Format.FORMAT_ARGB_PACKED, sail.ImgDtype.DATA_TYPE_EXT_1N_
↪BYTE))
    overlay_info.append([buffer_x1,buffer_y1,buffer_w1, buffer_h1])

    ret = bmcv.bmcv_overlay(image_org, overlay_info, overlay_images)

    ret = bmcv.imwrite("overlayed.jpg", image_org)
```

Sample2:

Use other libraries to generate the watermark of Chinese text, and use overlay to draw in the designated position to realize the puttext function indirectly

```
import sophon.sail as sail
from PIL import Image, ImageDraw, ImageFont

if __name__ == '__main__':
    # get original image
    handle = sail.Handle(0)
    bmcv = sail.Bmcv(handle)
    decoder = sail.Decoder("pics/1.jpg")
    org_image = decoder.read(handle)

    # the info of font
    watermark_w, watermark_h = 60, 30
    text = ["人", "车", "狗"]
    text_color = (255, 0, 0, 255)
    font = ImageFont.truetype("fonts/simhei.ttf", watermark_h, encoding="utf-8")

    # get watermark of labels
    overlay_images = []
    for i in range(len(text)):
        blank_image = Image.new('RGBA', (watermark_w, watermark_h), (255,255,
↪255,0))
        draw = ImageDraw.Draw(blank_image)
        draw.text((0,0), text[i], fill=text_color, font=font)
```

(continues on next page)

(continued from previous page)

```
        # blank_image.save("watermark.png")
        buffer = blank_image.tobytes()
        overlay_images.append(sail.BMImage(handle, buffer, watermark_h, watermark_
↪w, sail.Format.FORMAT_ARGB_PACKED, sail.ImgDtype.DATA_TYPE_EXT_
↪1N_BYTE))

    # after inference and other procedure
    # the position of the watermark map obtained by simulation
    overlay_info = []
    for i in range(len(text)):
        x = int(i*org_image.width()/len(text))
        y = int(org_image.height()/2)
        overlay_info.append([x, y, watermark_w, watermark_h])

    ret = bmcv.bmcv_overlay(org_image, overlay_info, overlay_images)
    ret = bmcv.imwrite("overlayed.jpg", org_image)
```

5.14.46 faiss_indexflatL2

Calculate squared L2 distance between query vectors and database vectors, output the top topK L2sqr-values and the corresponding indices.

Interface1:

```
faiss_indexflatL2(self, query_vecs: numpy.ndarray, query_vecs_L2norm: numpy.ndarray,F
↪database_vecs: numpy.ndarray, database_vecs_L2norm: numpy.ndarray,
                  vec_dims: int, query_vecs_nums: int, database_vecs_nums: int, topK: int) ->F
↪tuple[numpy.ndarray, numpy.ndarray]:

faiss_indexflatL2(self, query_vecs: numpy.ndarray, query_vecs_L2norm: numpy.ndarray,F
↪database_vecs: Tensor, database_vecs_L2norm: Tensor,
                  vec_dims: int, query_vecs_nums: int, database_vecs_nums: int, topK: int) ->F
↪tuple[numpy.ndarray, numpy.ndarray]:
```

Parameters:

- **query_vecs: numpy.ndarray** The query vectors, the supported data type is only numpy.float32.
- **query_vecs_L2norm: numpy.ndarray** Calculates the sum of the square values of the elements in each row of the query vector, numpy.float32.
- **database_vecs: numpy.ndarray or Tensor** The database vectors, the supported data type is only numpy.float32 or sail.Dtype.BM_FLOAT32.
- **database_vecs_L2norm: numpy.ndarray or Tensor** Calculates the sum of the square values of the elements in each row of the database vector, numpy.float32 or sail.Dtype.BM_FLOAT32.
- **vec_dims: int** The dimension of the query vectors and database vectors.

- **query_vecs_nums: int** The numbers of the query vectors.
- **database_vecs_nums: int** The numbers of the database vectors.
- **topK: int** Get top topK values.

Returns:

- **result: tuple[numpy.ndarray, numpy.ndarray]** Returns the square value of the top topK best-matched L2 distance and its corresponding index.

Sample1:

```
import numpy as np
import sophon.sail as sail
import time

if __name__ == '__main__':
    # 1. database_vecs
    db_vecs = np.array([
        [-2.0, 0.0, 4.0],
        [-5.0, 3.0, -1.0],
        [1.0, 2.0, 4.0],
        [0.0, 5.0, -3.0],
        [2.0, 1.0, -4.0],
    ], dtype=np.float32)

    # 2. query_vecs
    query_vecs = np.array([
        [1.0, 2.0, 3.0]
    ], dtype=np.float16)

    # 3. test bmcv.faiss_indexflatL2
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    bmcv = sail.Bmcv(handle)

    db_vecs_square = np.square(db_vecs)
    db_vecs_l2norm = np.sum(db_vecs_square, axis=1)
    print("db_vecs_l2norm:", db_vecs_l2norm)

    query_vecs_square = np.square(query_vecs)
    query_vecs_l2norm = np.sum(query_vecs_square, axis=1)
    print("query_vecs_l2norm:", query_vecs_l2norm)

    start = time.time()
    similarity_L2, index_L2 = bmcv.faiss_indexflatL2(query_vecs, query_vecs_l2norm, db_
↪vecs, db_vecs_l2norm, 3, 1, 5, 3)
    end = time.time()
    execution_time_milliseconds_L2 = (end - start) * 1000
    print(f"Execution time: {execution_time_milliseconds_L2:.6f} milliseconds")
    print("similarity_L2:", similarity_L2)
    print("index_L2:", index_L2)
```

Sample2:

```
import numpy as np
import sophon.sail as sail
import time

if __name__ == '__main__':
    # 1. database_vecs
    db_vecs = np.array([
        [-2.0, 0.0, 4.0],
        [-5.0, 3.0, -1.0],
        [1.0, 2.0, 4.0],
        [0.0, 5.0, -3.0],
        [2.0, 1.0, -4.0],
    ], dtype=np.float32)

    # 2. query_vecs
    query_vecs = np.array([
        [1.0, 2.0, 3.0]
    ], dtype=np.float16)

    # 3. test bmcv.faiss_indexflatL2
    tpu_id = 0
    handle = sail.Handle(tpu_id)
    bmcv = sail.Bmcv(handle)

    db_vecs_square = np.square(db_vecs)
    db_vecs_l2norm = np.sum(db_vecs_square, axis=1)
    print("db_vecs_l2norm:", db_vecs_l2norm)

    query_vecs_square = np.square(query_vecs)
    query_vecs_l2norm = np.sum(query_vecs_square, axis=1)
    print("query_vecs_l2norm:", query_vecs_l2norm)

    # 4. database_vecs and db_vecs_l2norm to sail::Tensor
    db_tensor = sail.Tensor(handle, db_vecs, False)
    db_tensor_l2norm = sail.Tensor(handle, db_vecs_l2norm, False)

    start = time.time()
    similarity_L2, index_L2 = bmcv.faiss_indexflatL2(query_vecs, query_vecs_l2norm, db_
↪tensor, db_tensor_l2norm, 3, 1, 5, 3)
    end = time.time()
    execution_time_milliseconds_L2 = (end - start) * 1000
    print(f"Execution time: {execution_time_milliseconds_L2:.6f} milliseconds")
    print("similarity_L2:", similarity_L2)
    print("index_L2:", index_L2)
```

5.14.47 faiss_indexflatIP

Calculate inner product distance between query vectors and database vectors, output the top K IP-values and the corresponding indices.

Interface:

```
faiss_indexflatIP(self, query_vecs: numpy.ndarray, database_vecs: numpy.ndarray,
                  vec_dims: int, query_vecs_nums: int, database_vecs_nums: int, topK: int) ->F
↪tuple[numpy.ndarray, numpy.ndarray]:

faiss_indexflatIP(self, query_vecs: numpy.ndarray, database_vecs: Tensor,
                  vec_dims: int, query_vecs_nums: int, database_vecs_nums: int, topK: int) ->F
↪tuple[numpy.ndarray, numpy.ndarray]:
```

Parameters:

- **query_vecs: numpy.ndarray** The query vectors, the supported data type is only numpy.float32.
- **database_vecs: numpy.ndarray or Tensor** The database vectors, the supported data type is only numpy.float32 or sail.Dtype.BM_FLOAT32.
- **vec_dims: int** The dimension of the query vectors and database vectors.
- **query_vecs_nums: int** The numbers of the query vectors.
- **database_vecs_nums: int** The numbers of the database vectors.
- **topK: int** Get top topK values.

Returns:

- **result: tuple[numpy.ndarray, numpy.ndarray]** Returns the top topK best-matched inner product distance values and their corresponding indexes.

Sample:

```
import numpy as np
import sophon.sail as sail
import time

if __name__ == '__main__':
    # 1. database_vecs
    db_vecs = np.array([
        [-2.0, 0.0, 4.0],
        [-5.0, 3.0, -1.0],
        [1.0, 2.0, 4.0],
        [0.0, 5.0, -3.0],
        [2.0, 1.0, -4.0],
    ], dtype=np.float32)

    # 2. query_vecs
    query_vecs = np.array([
```

(continues on next page)

(continued from previous page)

```
    [1.0, 2.0, 3.0]
], dtype=np.float16)

# 3. test bmcv.faiss_indexflatIP
tpu_id = 0
handle = sail.Handle(tpu_id)
bmcv = sail.Bmcv(handle)

db_tensor = sail.Tensor(handle, db_vecs, False)
start = time.time()
similarity_IP, index_IP = bmcv.faiss_indexflatIP(query_vecs, db_tensor, 3, 1, 5, 3)
# similarity_IP, index_IP = bmcv.faiss_indexflatIP(query_vecs, db_vecs, 3, 1, 5, 3)
end = time.time()
execution_time_milliseconds_L2 = (end - start) * 1000
print(f"Execution time: {execution_time_milliseconds_L2:.6f} milliseconds")
print("similarity_IP:", similarity_IP)
print("index_IP:", index_IP)
```

5.14.48 faiss_indexPQ_encode

Perform PQ quantization encoding on the input vector and output the encoded vector.

Interface:

```
faiss_indexPQ_encode(self, input_vecs: Tensor, centroids_vecs: Tensor,
            encode_vecs_num: int, vec_dims: int, slice_num: int, centroids_num: int, IP_
↪metric: int) -> Tensor:

faiss_indexPQ_encode(self, input_vecs: numpy.ndarray, centroids_vecs: numpy.ndarray,
            encode_vecs_num: int, vec_dims: int, slice_num: int, centroids_num: int, IP_
↪metric: int) -> numpy.ndarray:

faiss_indexPQ_encode(self, input_vecs: numpy.ndarray, centroids_vecs: Tensor, encoded_
↪vecs: Tensor
            encode_vecs_num: int, vec_dims: int, slice_num: int, centroids_num: int, IP_
↪metric: int) -> int:
```

Parameters:

- **input_vecs: numpy.ndarray or Tensor** Input vector to be encoded, data types are supported only numpy.float32 or sail.Dtype.BM_FLOAT32, encode_vecs_num * vec_dims.
- **centroids_vecs: numpy.ndarray or Tensor** centroids vector, data types are supported only numpy.float32 or sail.Dtype.BM_FLOAT32, slice_num * centroids_num * (vec_dims / slice_num).
- **encode_vecs_num: int** The number of vectors to be encoded.
- **vec_dims: int** Dimension of the vector to be encoded.

- **slice_num: int** The number of slices of the original vector dimensions, for example the original vector dimension is 512, slice_num = 8, and each subvector dimension is 64.
- **centroids_num: int** The number of centroids.
- **IP_metric: int** 0 indicates that the distance is calculated using L2, and 1 indicates that the distance is calculated using IP.

Returns:

- **result: numpy.ndarray or Tensor** Output the encoded vector, numpy.ndarray (uint8) or Tensor (BM_UINT8).

Sample:

```
import faiss
import numpy as np
import sophon.sail as sail

encode_vecs_num = 3
vec_dims = 64
db_vecs_num = 10000
slice_num = 8
centroids_num = 256
nbits = 8

np.random.seed(666)
data = np.random.rand(db_vecs_num, vec_dims).astype('float32')
pq = faiss.ProductQuantizer(vec_dims, slice_num, nbits)
pq.train(data)

# get centroids
centroids = faiss.vector_float_to_array(pq.centroids)
print('centroids.shape: ', centroids.shape)
np.save('pq_centroids_random.npy', centroids)

# faiss PQ encode
input_vector = np.random.rand(encode_vecs_num, vec_dims).astype('float32')
faiss_PQ_encode = pq.compute_codes(input_vector)
print('faiss_PQ_encode:\n', faiss_PQ_encode)
print('faiss_PQ_encode shape:', faiss_PQ_encode.shape)

# test sail bmcv.faiss_indexPQ_encode
handle = sail.Handle(0)
bmcv = sail.Bmcv(handle)

centroids_vecs = np.load("pq_centroids_random.npy").astype(np.float32)
print("centroids_vecs shape:", centroids_vecs.shape)
print("centroids_vecs dtype:", type(centroids_vecs))

input_tensor = sail.Tensor(handle, input_vector, False)
centroids_tensor = sail.Tensor(handle, centroids_vecs, False)
```

(continues on next page)

(continued from previous page)

```
encode_tensor = bmcv.faiss_indexPQ_encode(input_tensor, centroids_tensor, encode_
↪vecs_num, vec_dims, slice_num, centroids_num, 0)
print('bmcv_faiss_indexPQ_encode:\n', encode_tensor.asnumpy())

encode = bmcv.faiss_indexPQ_encode(input_vector, centroids_vecs, encode_vecs_num,F
↪vec_dims, slice_num, centroids_num, 0)
print('bmcv_faiss_indexPQ_encode:\n', encode)
```

5.14.49 faiss_indexPQ_ADC

The distance table is calculated by the query vector and the clustering center (centroid) vector, and the encoded database vector searches the table to calculate the distance and sorts it, output the topK distances and the corresponding indices.

Interface:

```
faiss_indexPQ_ADC(self, nxquery_vecs: numpy.ndarray, centroids_vecs: numpy.ndarray,F
↪nycodes_vecs: numpy.ndarray,
                  vec_dims: int, slice_num: int, centroids_num: int, database_vecs_num: int,F
↪query_vecs_num: int, topK: int, IP_metric: int) -> tuple[numpy.ndarray, numpy.
↪ndarray]:

faiss_indexPQ_ADC(self, nxquery_vecs: numpy.ndarray, centroids_vecs: Tensor, nycodes_
↪vecs: Tensor,
                  vec_dims: int, slice_num: int, centroids_num: int, database_vecs_num: int,F
↪query_vecs_num: int, topK: int, IP_metric: int) -> tuple[numpy.ndarray, numpy.
↪ndarray]:
```

Parameters:

- **nxquery_vecs: numpy.ndarray** query vectors, data types are supported only numpy.float32, and the size is query_vecs_nums * vec_dims.
- **centroids_vecs: numpy.ndarray or Tensor** centroids vectors, data types are supported only numpy.float32 or sail.Dtype.BM_FLOAT32, and the size is slice_num * centroids_num * (vec_dims / slice_num).
- **nycodes_vecs: numpy.ndarray or Tensor** Encoded database vector, data types are supported only numpy.uint8 or sail.Dtype.BM_UINT8, and the size is database_vecs_nums * slice_num.
- **vec_dims: int** dimension of the query vector.
- **slice_num: int** The number of slices of the original vector dimensions, for example the original vector dimension is 512, slice_num = 8, and each subvector dimension is 64.
- **centroids_num: int** The number of centroids.
- **database_vecs_nums: int** The number of database vectors.

- **query_vecs_nums: int** The number of query vectors.
- **topK: int** Get top topK values.
- **IP_metric: int** 0 indicates that the distance is calculated using L2, and 1 indicates that the distance is calculated using IP.

Returns:

- **result: tuple[numpy.ndarray, numpy.ndarray]** Returns the top topK best-matched distance values and their corresponding indexes.

Sample:

```
import numpy as np
import faiss
import sophon.sail as sail

np.random.seed(1024)
vec_dims = 512
database_vecs_num = 10000
query_vecs_num = 1

nydb_vecs = np.random.rand(database_vecs_num, vec_dims).astype('float32')
nxquery_vecs = np.random.rand(query_vecs_num, vec_dims).astype('float32')

slice_num = 8
n_bits = 8
index = faiss.IndexPQ(vec_dims, slice_num, n_bits)
index.train(nydb_vecs)
index.add(nydb_vecs)

# save centroids
centroids_faiss = index.pq.centroids
centroids = faiss.vector_float_to_array(centroids_faiss)
print('centroids.shape: ', centroids.shape)
np.save('centroids_random.npy', centroids)

# Compute PQ codes for the database vectors and save
db_codes = index.pq.compute_codes(nydb_vecs)
print('db_codes.shape:', db_codes.shape)
np.save('db_codes.npy', db_codes)

# faiss
D, I = index.search(nxquery_vecs, k=5)
print("The index of faiss:\n", I)
print("The distance of faiss:\n", D)

# test faiss_indexPQ_ADC of sail
handle = sail.Handle(0)
bmcv = sail.Bmcv(handle)

# get centroids
```

(continues on next page)

(continued from previous page)

```
centroids_vecs = np.load("centroids_random.npy").astype(np.float32)
print("centroids_vecs shape:", centroids_vecs.shape)
print("centroids_vecs dtype:", type(centroids_vecs))
# get PQ codes for the database vectors
nycodes_vecs = np.load("db_codes.npy")
print("nycodes_vecs shape:", nycodes_vecs.shape)
print("nycodes_vecs dtype:", type(nycodes_vecs))

# to tensor
centroids_tensor = sail.Tensor(handle, centroids_vecs, False)
nycode_tensor = sail.Tensor(handle, nycodes_vecs, False)

# D_, I_ = bmcv.faiss_indexPQ_ADC(nxquery_vecs, centroids_tensor, nycode_tensor,F
↪vec_dims, slice_num, 256, database_vecs_num, query_vecs_num, 5, 0)
D_, I_ = bmcv.faiss_indexPQ_ADC(nxquery_vecs, centroids_vecs, nycodes_vecs, vec_
↪dims, slice_num, 256, database_vecs_num, query_vecs_num, 5, 0)
print("The index of sail:\n", I_)
print("The distance of sail:\n", D_)
```

5.14.50 faiss_indexPQ_SDC

The Symmetric Distance Computation (SDC) lookup table is used to speed up the distance calculation between PQ encodings, output the topK distances and the corresponding indices.

Interface:

```
faiss_indexPQ_SDC(self, nxcodes_vecs: numpy.ndarray, nycodes_vecs: numpy.ndarray,F
↪sdc_table: numpy.ndarray,
                  slice_num: int, centroids_num: int, database_vecs_num: int, query_vecs_
↪num: int, topK: int, IP_metric: int) -> tuple[numpy.ndarray, numpy.ndarray]:

faiss_indexPQ_SDC(self, nxcodes_vecs: numpy.ndarray, nycodes_vecs: Tensor, sdc_table:F
↪Tensor,
                  slice_num: int, centroids_num: int, database_vecs_num: int, query_vecs_
↪num: int, topK: int, IP_metric: int) -> tuple[numpy.ndarray, numpy.ndarray]:
```

Parameters:

- **nxcodes_vecs: numpy.ndarray** Encoded query vectors, data types are supported only numpy.uint8, and the size is query_vecs_nums * slice_num.
- **nycodes_vecs: numpy.ndarray or Tensor** Encoded database vector, data types are supported only numpy.uint8 or sail.Dtype.BM_UINT8, and the size is database_vecs_nums * slice_num.
- **sdc_table: numpy.ndarray or Tensor** The Symmetric Distance Computation (SDC) lookup table, data types are supported only numpy.float32 or sail.Dtype.BM_FLOAT32, and the size is slice_num * centroids_num * centroids_num.

- **slice_num: int** The number of slices of the original vector dimensions, for example the original vector dimension is 512, slice_num = 8, and each subvector dimension is 64.
- **centroids_num: int** The number of centroids.
- **database_vecs_nums: int** The number of database vectors.
- **query_vecs_nums: int** The number of query vectors.
- **topK: int** Get top topK values.
- **IP_metric: int** 0 indicates that the distance is calculated using L2, and 1 indicates that the distance is calculated using IP.

Returns:

- **result: tuple[numpy.ndarray, numpy.ndarray]** Returns the top topK best-matched distance values and their corresponding indexes.

Sample:

```
import numpy as np
import faiss
import sophon.sail as sail

np.random.seed(1024)
vec_dims = 512            # The dimension of the vectors.
database_vecs_num = 10000  # The number of the database vectors.
query_vecs_num = 1        # The number of the query vectors.

# 1. Random database and query
database = np.random.rand(database_vecs_num, vec_dims).astype('float32')
query = np.random.rand(query_vecs_num, vec_dims).astype('float32')

# 2. The parameters of PQ
slice_num = 8      # Divide the vector of the vec_dims dimension into 8 subvectors
n_bits = 8         # centroids = 256 = 2^n_bits

# 3. Create a PQ index
index = faiss.IndexPQ(vec_dims, slice_num, n_bits)
index.train(database)  # Train the PQ index
index.add(database)

# 4. Set to SDC (Symmetric Distance Calculation)
index.search_type = faiss.IndexPQ.ST_SDC

# 5. Calculate sdc_table and save it
index.pq.compute_sdc_table()

sdc_table = faiss.vector_float_to_array(index.pq.sdc_table)
print('sdc_table shape:', sdc_table.shape)
np.save('sdc_table.npy', sdc_table)
```

(continues on next page)

(continued from previous page)

```
# 6. Query operations using faiss
topK = 5
D, I = index.search(query, topK)
print("The index of faiss:\n", I)
print("The distance of faiss:\n", D)

# Calculate and save the PQ encoding of the database vector (for sail test)
db_codes = index.pq.compute_codes(database)
print('db_codes.shape:', db_codes.shape)
np.save('db_codes.npy', db_codes)

# Calculate and save the PQ encoding of the query vectors (for sail test)
query_code = index.pq.compute_codes(query)
print('query_code.shape:', query_code.shape)
np.save('query_code.npy', query_code)
print("------------------------------------------------------------
↪------------------------------")

# test faiss_indexPQ_SDC
handle = sail.Handle(0)
bmcv = sail.Bmcv(handle)

# 1. get PQ codes of the database vectors
nycodes_vecs = np.load("db_codes.npy")
print("nycodes_vecs shape:", nycodes_vecs.shape)
print("nycodes_vecs dtype:", nycodes_vecs.dtype)

# 2. get PQ codes of the query vectors
nxcodes_vecs = np.load('query_code.npy')
print("nxcodes_vecs shape:", nxcodes_vecs.shape)
print("nxcodes_vecs dtype:", nxcodes_vecs.dtype)

# 3. get the SDC table
SDC_tables = np.load('sdc_table.npy')

D_, I_ = bmcv.faiss_indexPQ_SDC(nxcodes_vecs, nycodes_vecs, SDC_tables, slice_num,
↪ 256, database_vecs_num, query_vecs_num, topK, 0)
print("The index of sail:\n", I_)
print("The distance of sail:\n", D_)
```

5.15 sail.Blend

Picture stitching. Multiple pictures can be stitched together. Only BM1688 and CV186AH SoC mode support this interface.

5.15.1 __init__

Initialize Blend.

Interface:

```
def __init__(self, src_h: int, ovlap_attr: list[list[int]], bd_attr: list[list[int]], wgt_phy_
↪mem: list[list[string]], wgt_mode: sail.blend_wgt_mode))
```

Parameters:

- src_h: int

Input images. The height of the input images must be consistent.

- ovlap_attr: list[list[int]]

The left and right boundaries of the images overlap area.

- bd_attr: list[list[int]]

The left and right black border properties of the input images. This option is not currently supported, so just leave it blank.

- wgt_phy_mem: list[list[string]]

Weights file for input images. Reference[Get weight files of images], get image weight files。

- wgt_mode: sail.blend_wgt_mode

The stitching mode of the input image. Currently the following two stitching modes are supported, usually sail.blend_wgt_mode.BM_STITCH_WGT_YUV_SHARE is selected.

- sail.blend_wgt_mode.BM_STITCH_WGT_YUV_SHARE: YUV share alpha and beta(1 alpha + 1 beta)
- sail.blend_wgt_mode.BM_STITCH_WGT_UV_SHARE: UV share alpha and beta(2 alpha + 2 beta)

5.15.2 process

Perform image stitching.

Interface:

```
def process(input: list[BMImage], output: BMImage) -> int
```

Returns

- ret: int

Returns whether the image stitching is successful, 0 means success, other values mean failure.

Parameters:

- input: list[BMImage]

Input BMImage to be stitched.

- output: BMImage

Output stitched BMImage.

Interface:

```
def process(input: list[BMImage]) -> BMImage
```

Returns

- output: BMImage

Return stitched BMImage.

Parameters:

- input: list[BMImage]

Input BMImage to be stitched.

Sample:

```
import sophon.sail  as sail
if __name__ == '__main__':
    handle = sail.Handle(0)
    decoder = sail.Decoder("./left.jpg", False, 0)
    image_left = sail.BMImage()
    ret = decoder.read(handle, image_left)

    decoder = sail.Decoder("./right.jpg", False, 0)
    image_right = sail.BMImage()
    ret = decoder.read(handle, image_right)

    blend_obj = sail.Blend(2240, [[2112],[2239]], [], [["data/wgt/c01_alpha_444p_m2__0_
↪2240x128.bin","data/wgt/c01_beta_444p_m2__0_2240x128.bin"]], sail.blend_wgt_
↪mode.BM_STITCH_WGT_YUV_SHARE)
```

(continues on next page)

(continued from previous page)

```
    img = blend_obj.process([image_left,image_right])

    sail.Bmcv(handle).imwrite("result.jpg",img)
```

5.16 sail.MultiDecoder

Multichannel decoding interface, supporting simultaneous decoding of multichannel video.

5.16.1 __init__

Interface:

```
def __init__(self,
             queue_size: int = 10,
             tpu_id: int = 0,
             discard_mode: int = 0)
```

Parameters:

- queue_size: int

For each video, the length of the decoded cached image queue.

- tpu_id: int

Tpu id that used, which defaults to 0.

- discard_mode: int

Data discard method when the cache reaches the maximum value. 0 indicates that data is not put into the cache; 1 indicates that the image of the queue header is poped , and then push into the decoded image. The default value is 0.

5.16.2 set_read_timeout

Set the timeout period for reading images. This takes effect on the read and read_ interfaces.If the image is not obtained after the timeout, the result will be returned.

Interface:

```
def set_read_timeout(self, timeout: int) -> None
```

Parameters:

- timeout: int

Timeout period, in seconds.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    multiDecoder.set_read_timeout(100)
```

5.16.3 add_channel

Add a channel

Interface:

```
def add_channel(self,
                file_path: str,
                frame_skip_num: int = 0) -> int
```

Parameters:

- file_path: str

The path or link to the video.

- frame_skip_num: int

Number of active frame loss in decoded cache. The default value is 0, which means no active frame loss.

Returns

Returns the unique channel number corresponding to the video. The type is an integer.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
```

5.16.4 del_channel

Delete a video channel that has been added.

Interface:

```
def del_channel(self, channel_idx: int) -> int
```

Parameters:

- channel_idx: int

The channel number of the video to be deleted.

Returns

Return 0 on success and other values on failure.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []

    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
    ret = multiDecoder.del_channel(0)
    if(ret!=0):
        print("delete channel error!")
```

5.16.5 clear_queue

Clears the image cache for the specified channel.

Interface:

```
def clear_queue(self, channel_idx: int) -> int
```

Parameters:

- channel_idx: int

The channel number of the video to be deleted.

Returns:

Return 0 on success and other values on failure.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
    ret = multiDecoder.clear_queue(0)
    if(ret!=0):
        print(" Clear failure!")
```

5.16.6 read

Gets an image from the specified video channel.

Interface1:

```
def read(self,
        channel_idx: int,
        image: BMImage,
        read_mode: int = 0) -> int
```

Parameters1:

- channel_idx: int

The specified video channel number.

- image: BMImage

The decoded image.

- read_mode: int

Mode of obtaining images. 0 indicates that one image is read directly from the cache without waiting, and will be returned whether it is read or not. Others representations wait until the image is retrieved or timeout, and then return.

Returns1:

Return 0 on success and other values on failure.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    frame_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        frame_list.append([])
    count = 0
    while True:
        count += 1
        for idx in channel_list:
            bmimg = sail.BMImage()
            ret = multiDecoder.read(idx,bmimg,1)
            frame_list[idx].append(bmimg)
        if count == 20:
            break
```

Interface2:

```
def read(self, channel_idx: int) -> BMImage
```

Parameters2:

- channel_idx: int

The specified video channel number.

Returns2:

Returns the decoded image of type BMImage.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    frame_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
```

(continues on next page)

(continued from previous page)

```
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        frame_list.append([])
    count = 0
    while True:
        count += 1
        for idx in channel_list:
            bmimg = multiDecoder.read(idx)
            frame_list[idx].append(bmimg)
        if count == 20:
            break
```

5.16.7 read_

Gets an image from the specified video channel, usually used with BMImageArray.

Interface1:

```
def read_(self,
        channel_idx: int,
        image: bm_image,
        read_mode: int=0) -> int
```

Parameters1:

- channel_idx: int

The specified video channel number.

- image: bm_image

The decoded image.

- read_mode: int

Mode of obtaining images. 0 indicates that one image is read directly from the cache without waiting, and will be returned whether it is read or not. Others representations wait until the image is retrieved or timeout, and then return.

Returns1:

Return 0 on success and other values on failure.

Sample:

```
import sophon.sail as sail
if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
```

(continues on next page)

(continued from previous page)

```
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    frame_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        frame_list.append([])
    count = 0
    while True:
        count += 1
        for idx in channel_list:
            img = sail.BMImage()
            bmimg = img.data()
            ret = multiDecoder.read_(idx,bmimg,1)
            frame_list[idx].append(bmimg)
        if count == 20:
            break
```

Interface2:

```
def read_(self, channel_idx: int) -> bm_image:
```

Parameters2:

- channel_idx: int

The specified video channel number.

Returns2:

Returns the decoded image of type bm_image.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    frame_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
```

(continues on next page)

(continued from previous page)

```
        frame_list.append([])
    count = 0
    while True:
        count += 1
        for idx in channel_list:
            bmimg = multiDecoder.read_(idx)
            frame_list[idx].append(bmimg)
        if count == 20:
            break
```

5.16.8 reconnect

Reconnect the video of the corresponding channel.

Interface:

```
def reconnect(self, channel_idx: int) -> int
```

Parameters:

- channel_idx: int

The channel index of the input image.

Returns

Return 0 on success and other values on failure.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
    ret = multiDecoder.reconnect(0)
    print(ret)
```

5.16.9 get_frame_shape

Get the image shape of the corresponding channel.

Interface:

```
def get_frame_shape(self, channel_idx: int) -> list[int]
```

Parameters:

- channel_idx: int

The channel index of the input image.

Returns

Returns a list:[1, number of channels, image height, image width].

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        print(multiDecoder.get_frame_shape(idx))
```

5.16.10 set_local_flag

Set whether the video is a local video. If it is not called, the video is represented as a network video stream.

Interface:

```
def set_local_flag(self, flag: bool) -> None:
```

Parameters:

- flag: bool

Standard bit, if True, fixed decoding of 25 frames per second per video channel

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    multiDecoder.set_local_flag(True)
```

5.16.11 get_channel_fps

Get the video fps of the specified channel

Interface

```
def get_channel_fps(self, channel_idx: int) -> float:
```

Parameters

- channel_idx: int

The specified channel index

Returns

Returns the video fps of the specified channel

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        print(multiDecoder.get_channel_fps(idx))
```


5.16.12 get_drop_num

Obtain the number of dropped frames.

Interface:

```
def get_drop_num(self, channel_idx: int) -> int
```

Parameters:

- channel_idx: int

The channel index of the input image.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
    dev_id=0
    discard_mode=0
    multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
    channel_list = []
    file_path = "your_video_path"
    for i in range(4):
        idx = multiDecoder.add_channel(file_path)
        if(idx<0):
            exit(-1)
        channel_list.append(idx)
        print(multiDecoder.get_drop_num(idx))
```

5.16.13 reset_drop_num

Set the number of dropped frames to zeros.

Interface:

```
def reset_drop_num(self,  channel_idx: int) -> None:
```

Parameters:

- channel_idx: int

The channel index of the input image.

Sample:

```
import sophon.sail as sail

if __name__ == '__main__':
    queue_size=16
```

(continues on next page)

(continued from previous page)

```
dev_id=0
discard_mode=0
multiDecoder = sail.MultiDecoder(queue_size,dev_id,discard_mode)
channel_list = []
file_path = "your_video_path"
for i in range(4):
    idx = multiDecoder.add_channel(file_path)
    if(idx<0):
        exit(-1)
    channel_list.append(idx)
    multiDecoder.reset_drop_num(idx)
```

5.17 SAIL Python high performance interface

5.17.1 sail.TensorPTRWithName

Tensor with name.

get_name

Get Tensor name

Interface:

```
def get_name(self) -> str
```

Returns:

Return Tensor name

get_data

Get Tensor

Interface:

```
def get_data(self) -> Tensor
```

Returns:

Return Tensor

5.17.2 sail.ImagePreProcess

General preprocessing interface, internal use of thread pool way.

__init__

Interface:

```
def __init__(self,
             batch_size: int,
             resize_mode:sail_resize_type,
             tpu_id: int=0,
             queue_in_size: int=20,
             queue_out_size: int=20,
             use_mat_output: bool = False)
```

Parameters:

- batch_size: int

The batch size of output

- resize_mode: sail_resize_type

Methods of internal scaling.

- tpu_id: int

The Tensor Computing Processor id that used, which defaults to 0

- queue_in_size: int

The maximum length of the input image queue cache, which defaults to 20.

- queue_out_size: int

The maximum length of Tensor queue cache of preprocess result , which is 20 by default.

- use_mat_output: bool

Whether to use OpenCV Mat as the output of the picture. The default value is False.

SetResizeImageAtrr

Sets the properties of the image scaling.

Interface:

```
def SetResizeImageAtrr(self,
             output_width: int,
             output_height: int,
             bgr2rgb: bool,
             dtype: ImgDtype) -> None
```

Parameters:

- output_width: int

The image width after scaling

- output_height: int

The image height after scaling

- bgr2rgb: bool

Whether to convert an image from BGR to GRB.

- dtype: ImgDtype

The data type after image scaling, the current version only supports BM_FLOAT32,BM_INT8,BM_UINT8. It can be set according to the input data type of the model.

SetPaddingAtrr

Sets the Padding properties, only works when the resize_mode is among BM_PADDING_VPP_NEAREST, BM_PADDING_TPU_NEAREST, BM_PADDING_TPU_LINEAR, BM_PADDING_TPU_BICUBIC.

Interface:

```
def SetPaddingAtrr(self,
          padding_b: int=114,
          padding_g: int=114,
          padding_r: int=114,
          align: int=0) -> None
```

Parameters: * padding_b: int

The padding pixel value of b channel, which defaults to 114.

- padding_g: int

The padding pixel value of g channel, which defaults to 114.

- padding_r: int

The padding pixel value of r channel, which defaults to 114.

- align: int

Image fill position, 0 indicates fill from the top left corner, 1 indicates center fill, default is 0.

SetConvertAtrr

Sets the properties of the linear transformation.

Interface:

```
def SetConvertAtrr(self, alpha_beta) -> int
```

Parameters:

- alpha_beta: (a0, b0), (a1, b1), (a2, b2)。

  a0 is the coefficient of linear transformation for the 0th channel;

  b0 is the offset of linear transformation for the 0th channel;

  a1 is the coefficient of linear transformation for the 1th channel;

  b1 is the offset of linear transformation for the 1th channel;

  a2 is the coefficient of linear transformation for the 2th channel;

  b2 is the offset of linear transformation for the 2th channel;

Returns:

Return 0 on success and other values on failure.

PushImage

push image data

Interface:

```
def PushImage(self,
    channel_idx: int,
    image_idx: int,
    image: BMImage) -> int
```

Parameters:

- channel_idx: int

The channel index of the input image

- image_idx: int

The image index of the input image

- image: BMImage

The input image

Returns:

Return 0 on success and other values on failure.

GetBatchData

Get process result.

Interface:

```
def GetBatchData(self)
   -> tuple[Tensor, list[BMImage],list[int],list[int],list[list[int]]]
```

Returns: tuple[data, images, channels, image_idxs, padding_attrs]

- data: Tensor

The inference result

- images: list[BMImage]

Original image queue

- channels: list[int]

The result corresponds to the channel sequence of the original picture.

- image_idxs: list[int]

The result corresponds to the index sequence of the original picture.

- padding_attrs: list[list[int]]

The attribute list of the filling image. The starting point coordinate x, starting point coordinate y, the width after scaling, and the height after scaling.

set_print_flag

Set the flag bit for printing logs. Logs are not printed when this interface is not used.

Interface:

```
def set_print_flag(self, flag: bool) -> None:
```

Returns:

- flag: bool

Flag bit for printing. False means no printing, True indicates printing.

5.17.3 sail.EngineImagePreProcess

Image inference interface with preprocessing function, internal use of thread pool way, higher efficiency while using Python.

__init__

Interface:

```
def __init__(self,
             bmodel_path: str,
             tpu_id: int,
             use_mat_output: bool = False,
             core_list:list = [])
```

Parameters:

- bmodel_path: str

The path of the input model.

- tpu_id: int

The Tensor Computing Processor id that used.

- use_mat_output: bool

Whether to use OpenCV' s Mat as the output of the picture. The default value is False, which means not used.

- use_mat_output: bool

When using the processor and BModel that support multi-core inference, you can choose multiple cores to use for inference. By default, N cores starting from core0 are used for inference, and N is determined by the current bmodel. For processors and bmodel models that only support single-core inference, only a single core used for inference is supported, and the input list length of the parameter must be 1, if the length of the incoming list is greater than 1, the inference will be automatically on the core0. Default is empty. If this parameter is not specified, the inference is performed by N cores starting from core 0.

InitImagePreProcess

Initialize the image preprocessing module.

Interface:

```
def InitImagePreProcess(self,
             resize_mode: sail_resize_type,
             bgr2rgb: bool = False,
             queue_in_size: int = 20,
             queue_out_size: int = 20) -> int
```

Parameters:

- resize_mode: sail_resize_type

Methods of internal scaling.

- bgr2rgb: bool

Whether to convert an image from BGR to RGB.

- queue_in_size: int

The maximum length of the input image queue cache, which defaults to 20. Must not be less than the batch_size of bmodel, if not, the value will be set to batch_size.

- queue_out_size: int

The maximum length of Tensor queue cache of preprocess result, which is 20 by default. Must not be less than the batch_size of bmodel, if not, the value will be set to batch_size.

Returns:

Return 0 on success and other values on failure.

SetPaddingAtrr

Sets the Padding properties, only works when the resize_mode is among BM_PADDING_VPP_NEAREST, BM_PADDING_TPU_NEAREST, BM_PADDING_TPU_LINEAR, BM_PADDING_TPU_BICUBIC.

Interface:

```
def SetPaddingAtrr(self,
          padding_b:int=114,
          padding_g:int=114,
          padding_r:int=114,
          align:int=0) -> int
```

Parameters: * padding_b: int

The padding pixel value of b channel, which defaults to 114.

- padding_g: int

The padding pixel value of g channel, which defaults to 114.

- padding_r: int

The padding pixel value of r channel, which defaults to 114.

- align: int

Image fill position, 0 indicates fill from the top left corner, 1 indicates center fill, default is 0.

Returns:

Return 0 on success and other values on failure.

SetConvertAtrr

Sets the properties of the linear transformation.

Interface:

```
def SetConvertAtrr(self, alpha_beta) -> int:
```

Parameters:

- alpha_beta: (a0, b0), (a1, b1), (a2, b2)。

  a0 is the coefficient of linear transformation for the 0th channel;

  b0 is the offset of linear transformation for the 0th channel;

  a1 is the coefficient of linear transformation for the 1th channel;

  b1 is the offset of linear transformation for the 1th channel;

  a2 is the coefficient of linear transformation for the 2th channel;

  b2 is the offset of linear transformation for the 2th channel;

Returns:

Return 0 on success and other values on failure.

PushImage

push image data

Interface:

```
def PushImage(self,
              channel_idx: int,
              image_idx: int,
              image: BMImage) -> int
```

Parameters: * channel_idx: int

The channel index of the input image

- image_idx: int

The image index of the input image

- image: BMImage

The input image

Returns:

Return 0 on success and other values on failure.

GetBatchData_Npy

Get a batch of inference results. When using this interface, use_mat_output must be False because the result type is BMImage.

Interface:

```
def GetBatchData_Npy(self)
-> tuple[[dict[str, ndarray], list[BMImage],list[int],list[int],list[list[int]]]]
```

Returns:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: dict[str, ndarray]

The inference result

- ost_images: list[BMImage]

Original image queue

- channels: list[int]

The result corresponds to the channel sequence of the original picture.

- image_idxs: list[int]

The result corresponds to the index sequence of the original picture.

- padding_attrs: list[list[int]]

The attribute list of the filling image. The starting point coordinate x, starting point coordinate y, the width after scaling, and the height after scaling.

GetBatchData_Npy2

Get a batch of inference results. When using this interface, use_mat_output must be True because the result type is numpy.ndarray[numpy.uint8].

Interface:

```
def GetBatchData_Npy2(self)
   -> tuple[dict[str, ndarray], list[numpy.ndarray[numpy.uint8]],list[int],list[int],list[list[int]]]
```

Returns:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: dict[str, ndarray]

The inference result

- ost_images: list[numpy.ndarray[numpy.uint8]]

Original image queue

- channels: list[int]

The result corresponds to the channel sequence of the original picture.

- image_idxs: list[int]

The result corresponds to the index sequence of the original picture.

- padding_attrs: list[list[int]]

The attribute list of the filling image. The starting point coordinate x, starting point coordinate y, the width after scaling, and the height after scaling.

GetBatchData

Get a batch of inference results. When using this interface, use_mat_output must be False because the result type is BMImage. It is worth noting that output tensors must be manually released.

Interface:

```
def GetBatchData(self,
        need_d2s: bool = True)
        -> tuple[list[TensorPTRWithName], list[BMImage],list[int],list[int],list[list[int]]]
```

Parameters:

- need_d2s: bool

Whether to move data to the system memory. The default value is True, which means Yes.

Returns:

tuple[output_array, ost_images, channels, image_idxs, padding_attrs]

- output_array: list[TensorPTRWithName]

The inference result

- ost_images: list[BMImage]

Original image queue

- channels: list[int]

The result corresponds to the channel sequence of the original picture.

- image_idxs: list[int]

The result corresponds to the index sequence of the original picture.

- padding_attrs: list[list[int]]

The attribute list of the filling image. The starting point coordinate x, starting point coordinate y, the width after scaling, and the height after scaling.

get_graph_name

Get the name of model computation graph

Interface:

```
def get_graph_name(self) -> str
```

Returns:

Return the first name of model computation graph

get_input_width

Get the width of model input.

Interface:

```
def get_input_width(self) -> int
```

Returns:

Return the width of model input.

get_input_height

Get the height of model input.

Interface:

```
def get_input_height(self) -> int
```

Returns:

Return the height of model input.

get_output_names

Get tensor names of model output.

Interface:

```
def get_output_names(self) -> list[str]
```

Returns:

Return all tensor names of model output.

get_output_shape

Get the shape of the specified output Tensor

Interface:

```
def get_output_shape(self, tensor_name: str) -> list[int]
```

Parameters:

- tensor_name: str

The name of output tensor

Returns:

Return the shape of the specified output Tensor

5.17.4 Yolov5 post-processing acceleration interfaces

Post-processing acceleration interfaces for single-output and three-output YOLOv5 models.

sail.algo_yolov5_post_1output

For post-processing interfaces with a single output YOLOv5 model, internally implemented using thread pools.

__init__

Interface:

```
def __init__(
        self,
        shape: list[int],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20,
        input_use_multiclass_nms: bool=True,
        agnostic: bool=False)
```

Parameters:

- shape: list[int]

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- input_use_multiclass_nms: bool

Each detection box has multiple categories.

- agnostic: bool

Algorithm without class-specific NMS

push_npy

Input numpy. Only input with a batchsize of 1 is supported, or data is split before input and then sent to the interface.

Interface:

```
def push_npy(self,
        channel_idx: int,
        image_idx: int,
        data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],
        dete_threshold: float,
        nms_threshold: float,
        ost_w: int,
        ost_h: int,
        padding_left: int,
        padding_top: int,
        padding_width: int,
        padding_height: int) -> int
```

Parameters:

- channel_idx: int

The channel number of the input image.

- image_idx: int

The sequence number of the input image.

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]]

The input data.

- dete_threshold: float

Detection threshold sequence.

- nms_threshold: float

nms threshold

- ost_w: int

The width of original image.

- ost_h: int

The height of original image.

- padding_left: int

The starting point coordinate x of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_top: int

The starting point coordinate y of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_width: int

Fill the width of the image,

The width of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_height: int

The height of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

Returns:

Return 0 if successful, otherwise failed.

push_data

Input data. The value of batchsize other than 1 is supported.

Interface1:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: TensorPTRWithName,
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

Parameters1:

- channel_idx: int

The channel number of the input image.

- image_idx: int

The sequence number of the input image.

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],

The input data.

- dete_threshold: float

Detection threshold sequence.

- nms_threshold: float

nms threshold.

- ost_w: int

The width of original image.

- ost_h: int

The height of original image.

- padding_attrs: list[list[int]]

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

Interface2:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: TensorPTRWithName,
    dete_threshold: list[list[float]],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

Parameters2:

- channel_idx: int

The channel number of the input image.

- image_idx: int

The sequence number of the input image.

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],

The input data.

- dete_threshold: float

Detection threshold sequence.

- nms_threshold: float

nms threshold.

- ost_w: int

The width of original image.

- ost_h: int

The height of original image.

- padding_attrs: list[list[int]]

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
def get_result_npy(self)
      -> tuple[tuple[int, int, int, int, int, float],int, int]
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

sail.algo_yolov5_post_3output

For post-processing interfaces with a triple output YOLOv5 model, internally implemented using thread pools.

__init__

Interface:

```
def __init__(
        self,
        shape: list[list[int]],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20,
        input_use_multiclass_nms: bool=True,
        agnostic: bool=False)
```

Parameters:

- shape: list[list[int]]

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- input_use_multiclass_nms: bool

Each detection box has multiple categories.

- agnostic: bool

Algorithm without class-specific NMS

push_data

Support input with arbitrary batchsize.

Interface1:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: list[TensorPTRWithName],
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

Parameters1:

- channel_idx: list[int]

The channel number of the input image.

- image_idx: list[int]

The sequence number of the input image.

- input_data: list[TensorPTRWithName],

The input data, including three outputs.

- dete_threshold: list[float]

Detection threshold sequence.

- nms_threshold: list[float]

nms threshold.

- ost_w: list[int]

The width of original image.

- ost_h: list[int]

The height of original image.

- padding_attrs: list[list[int]]

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

Interface2:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: list[TensorPTRWithName],
    dete_threshold: list[list[float]],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

Parameters2:

- channel_idx: list[int]

The channel number of the input image.

- image_idx: list[int]

The sequence number of the input image.

- input_data: list[TensorPTRWithName],

The input data, including three outputs.

- dete_threshold: list[list[float]]

Detection threshold sequence.

- nms_threshold: list[float]

nms threshold.

- ost_w: list[int]

The width of original image.

- ost_h: list[int]

The height of original image.

- padding_attrs: list[list[int]]

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
def get_result_npy(self)
        -> tuple[tuple[int, int, int, int, int, float],int, int]
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

reset_anchors

Reset anchors.

Interface:

```
def reset_anchors(self, anchors_new: list[list[list[int]]]) -> int
```

Parameters:

- anchors_new: list[list[list[int]]]

new anchors.

Returns:

Return 0 if successful, otherwise failed.

sail.algo_yolov5_post_cpu_opt

The post-processing is accelerated for the 3-output or 1-output yolov5 model.

__init__

Interface:

```
def __init__(
        self,
        shapes: list[list[int]],
        network_w: int = 640,
        network_h: int = 640)
```

Parameters:

- shapes: list[list[int]]

Input. Shape of input data.

- network_w: int

Input. Input width of the model, default is 640.

- network_h: int

Input. Input height of the model, default is 640.

process

Processing interface.

Interface 1:

```
def process(self,
    input_data: list[TensorPTRWithName],
    ost_w: list[int],
    ost_h: list[int],
    dete_threshold: list[float],
    nms_threshold: list[float],
    input_keep_aspect_ratio: bool,
    input_use_multiclass_nms: bool)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

Parameters 1:

- input_data: list[TensorPTRWithName]

Input. Input data with three outputs or one output.

- ost_w: list[int]

Input. Width of original images.

- ost_h: list[int]

Input. Height of original images.

- dete_threshold: list[float]

Input. Detection threshold.

- nms_threshold: list[float]

Input. NMS threshold

- input_keep_aspect_ratio: bool

Input. Whether to keep aspect ratio in images.

- input_use_multiclass_nms: bool

Input. Whether to use multiclass nms.

Interface 2:

```
def process(self,
    input_data: dict[str, Tensor],
    ost_w: list[int],
    ost_h: list[int],
    dete_threshold: list[float],
    nms_threshold: list[float],
    input_keep_aspect_ratio: bool,
    input_use_multiclass_nms: bool)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

Parameters 2:

- input_data: dict[str, Tensor]

Input. Input data with three outputs or one output.

- ost_w: list[int]

Input. Width of original images.

- ost_h: list[int]

Input. Height of original images.

- dete_threshold: list[float]

Input. Detection threshold.

- nms_threshold: list[float]

Input. NMS threshold

- input_keep_aspect_ratio: bool

Input. Whether to keep aspect ratio for boxes.

- input_use_multiclass_nms: bool

Input. Whether to multiclass nms.

Returns:

list[list[tuple[left, top, right, bottom, class_id, score]]]

- left: int

The leftmost x-coordinate of the detection result.

- top: int

The topmost y-coordinate of the detection result.

- right: int

The rightmost x-coordinate of the detection result.

- bottom: int

The bottommost y-coordinate of the detection result.

- class_id: int

The class label of the detection result.

- score: float

The score of the detection result.

Interface 3:

```
def process(self,
    input_data: list[TensorPTRWithName],
    ost_w: list[int],
    ost_h: list[int],
    dete_threshold: list[list[float]],
    nms_threshold: list[float],
    input_keep_aspect_ratio: bool,
    input_use_multiclass_nms: bool)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

Parameters 3:

- input_data: list[TensorPTRWithName]

Input. Input data with three outputs or one output.

- ost_w: list[int]

Input. Width of original images.

- ost_h: list[int]

Input. Height of original images.

- dete_threshold: list[list[float]]

Input. Detection threshold.

- nms_threshold: list[float]

Input. NMS threshold

- input_keep_aspect_ratio: bool

Input. Whether to keep aspect ratio in images.

- input_use_multiclass_nms: bool

Input. Whether to use multiclass nms.

Interface 4:

```
def process(self,
    input_data: dict[str, Tensor],
    ost_w: list[int],
    ost_h: list[int],
    dete_threshold: list[list[float]],
    nms_threshold: list[float],
    input_keep_aspect_ratio: bool,
    input_use_multiclass_nms: bool)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

Parameters 4:

- input_data: dict[str, Tensor]

Input. Input data with three outputs or one output.

- ost_w: list[int]

Input. Width of original images.

- ost_h: list[int]

Input. Height of original images.

- dete_threshold: list[list[float]]

Input. Detection threshold.

- nms_threshold: list[float]

Input. NMS threshold

- input_keep_aspect_ratio: bool

Input. Whether to keep aspect ratio for boxes.

- input_use_multiclass_nms: bool

Input. Whether to multiclass nms.

Returns:

list[list[tuple[left, top, right, bottom, class_id, score]]]

- left: int

The leftmost x-coordinate of the detection result.

- top: int

The topmost y-coordinate of the detection result.

- right: int

The rightmost x-coordinate of the detection result.

- bottom: int

The bottommost y-coordinate of the detection result.

- class_id: int

The class label of the detection result.

- score: float

The score of the detection result.

reset_anchors

Update the size of the anchor.

Interface:

```
def reset_anchors(self, anchors_new: list[list[list[int]]]) -> int
```

Parameters:

- anchors_new: list[list[list[int]]]

List of anchor sizes to be updated.

Returns:

A return value of 0 indicates success, while other values indicate failure.

sail.algo_yolov5_post_cpu_opt_async

For accelerated post-processing interfaces in cpu, internally implemented using thread pools.

__init__

Interface:

```
def __init__(
        self,
        shape: list[list[int]],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20,
        use_multiclass_nms: bool=True)
```

Parameters:

- shape: list[list[int]]

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- use_multiclass_nms: bool

Whether to use multi-class NMS, which defaults to true.

push_data

Support input with arbitrary batchsize.

Interface1:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: list[TensorPTRWithName],
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
```

(continues on next page)

(continued from previous page)

```
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

Parameters1:

- channel_idx: list[int]

The channel number of the input image.

- image_idx: list[int]

The sequence number of the input image.

- input_data: list[TensorPTRWithName],

The input data, including three outputs.

- dete_threshold: list[float]

Detection threshold sequence.

- nms_threshold: list[float]

nms threshold.

- ost_w: list[int]

The width of original image.

- ost_h: list[int]

The height of original image.

- padding_attrs: list[list[int]]

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

Interface2:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: list[TensorPTRWithName],
    dete_threshold: list[list[float]],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

Parameters2:

- channel_idx: list[int]

The channel number of the input image.

- image_idx: list[int]

The sequence number of the input image.

- input_data: list[TensorPTRWithName],

The input data, including three outputs.

- dete_threshold: list[list[float]]

Detection threshold sequence.

- nms_threshold: list[float]

nms threshold.

- ost_w: list[int]

The width of original image.

- ost_h: list[int]

The height of original image.

- padding_attrs: list[list[int]]

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
def get_result_npy(self)
      -> tuple[tuple[int, int, int, int, int, float],int, int]
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

reset_anchors

Reset anchors.

Interface:

```
def reset_anchors(self, anchors_new: list[list[list[int]]]) -> int
```

Parameters:

- anchors_new: list[list[list[int]]]

new anchors.

Returns:

Return 0 if successful, otherwise failed.

sail.tpu_kernel_api_yolov5_detect_out

The post-processing is accelerated using the Tensor Computing Processor Kernel for the 3-output yolov5 model, which currently only supports BM1684x, and the version of libsophon must be no less than 0.4.6 (v23.03.01).

__init__

Interface:

```
def __init__(
        self,
        device_id: int,
        shape: list[list[int]],
        network_w: int = 640,
        network_h: int = 640,
        module_file: str="/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_
↪kernel_module.so")
```

Parameters:

- device_id: int

Input. Device ID used.

- shape: list[list[int]]

Input. Shape of input data.

- network_w: int

Input. Input width of the model, default is 640.

- network_h: int

Input. Input height of the model, default is 640.

- module_file: str

Input. File path of the kernel module, default is "/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so" .

process

Processing interface.

Interface 1:

```
def process(self,
    input_data: list[TensorPTRWithName],
    dete_threshold: float,
    nms_threshold: float,
    release_input: bool = False)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

Parameters 1:

- input_data: list[TensorPTRWithName]

Input. Input data with three outputs.

· dete_threshold: float

Input. Detection threshold.

· nms_threshold: float

Input. NMS threshold

· release_input: bool

Input. Release input memory, default is False.

Interface 2:

```
def process(self,
    input_data: dict[str, Tensor],
    dete_threshold: float,
    nms_threshold: float,
    release_input: bool = False)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

Parameters 2:

· input_data: dict[str, Tensor]

Input. Input data with three outputs.

· dete_threshold: float

Input. Detection threshold.

· nms_threshold: float

Input. NMS threshold.

· release_input: bool

Input parameters. Release input memory, default is False.

Returns:

list[list[tuple[left, top, right, bottom, class_id, score]]]

· left: int

The leftmost x-coordinate of the detection result.

· top: int

The topmost y-coordinate of the detection result.

· right: int

The rightmost x-coordinate of the detection result.

· bottom: int

The bottommost y-coordinate of the detection result.

· class_id: int

The class label of the detection result.

- score: float

The score of the detection result.

reset_anchors

Update the size of the anchor.

Interface:

```
def reset_anchors(self, anchors_new: list[list[list[int]]]) -> int
```

Parameters:

- anchors_new: list[list[list[int]]]

List of anchor sizes to be updated.

Returns:

A return value of 0 indicates success, while other values indicate failure.

sail.tpu_kernel_api_yolov5_out_without_decode

The post-processing is accelerated using the Tensor Computing Processor Kernel for the 1-output yolov5 model, which currently only supports BM1684x, and the version of libsophon must be no less than 0.4.6 (v23.03.01).

__init__

Interface:

```
def __init__(
        self,
        device_id: int,
        shape: list[int],
        network_w: int = 640,
        network_h: int = 640,
        module_file: str="/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_
↪kernel_module.so")
```

Parameters:

- device_id: int

Input. Device ID used.

- shape: list[int]

Input. Shape of input data.

- network_w: int

Input. Input width of the model, default is 640.

- network_h: int

Input. Input height of the model, default is 640.

- module_file: str

Input. File path of the kernel module, default is "/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so" .

process

Processing interface.

Interface 1:

```
def process(self,
    input_data: TensorPTRWithName,
    dete_threshold: float,
    nms_threshold: float)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

Parameters 1:

- input_data: TensorPTRWithName

Input. Input data with one output.

- dete_threshold: float

Input. Detection threshold.

- nms_threshold: float

Input. NMS threshold

Interface 2:

```
def process(self,
    input_data: dict[str, Tensor],
    dete_threshold: float,
    nms_threshold: float)
        -> list[list[tuple[int, int, int, int, int, float]]]
```

Parameters 2:

- input_data: dict[str, Tensor]

Input. Input data with one output.

- dete_threshold: float

Input. Detection threshold.

- nms_threshold: float

Input. NMS threshold.

Returns:

list[list[tuple[left, top, right, bottom, class_id, score]]]

- left: int

The leftmost x-coordinate of the detection result.

- top: int

The topmost y-coordinate of the detection result.

- right: int

The rightmost x-coordinate of the detection result.

- bottom: int

The bottommost y-coordinate of the detection result.

- class_id: int

The class label of the detection result.

- score: float

The score of the detection result.

5.17.5 Yolox post-processing acceleration interfaces

sail.algo_yolox_post

For post-processing interfaces with yolox model, internally implemented using thread pools.

__init__

Interface:

```
def __init__(
        self,
        shape: list[int],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20)
```

Parameters:

- shape: list[int]

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

push_npy

Input numpy. Only input with a batchsize of 1 is supported, or data is split before input and then sent to the interface.

Interface:

```
def push_npy(self,
        channel_idx: int,
        image_idx: int,
        data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],
        dete_threshold: float,
        nms_threshold: float,
        ost_w: int,
        ost_h: int,
        padding_left: int,
        padding_top: int,
        padding_width: int,
        padding_height: int) -> int
```

Parameters:

- channel_idx: int

The channel number of the input image.

- image_idx: int

The sequence number of the input image.

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]]

The input data.

- dete_threshold: float

Detection threshold sequence.

- nms_threshold: float

nms threshold

- ost_w: int

The width of original image.

- ost_h: int

The height of original image.

- padding_left: int

The starting point coordinate x of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_top: int

The starting point coordinate y of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_width: int

Fill the width of the image,

The width of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_height: int

The height of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

Returns:

Return 0 if successful, otherwise failed.

push_data

Input data. The value of batchsize other than 1 is supported.

Interface:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: TensorPTRWithName,
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

Parameters:

- channel_idx: int

The channel number of the input image.

- image_idx: int

The sequence number of the input image.

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],

The input data.

- dete_threshold: float

Detection threshold sequence.

- nms_threshold: float

nms threshold.

- ost_w: int

The width of original image.

- ost_h: int

The height of original image.

- padding_attrs: list[list[int]]

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
def get_result_npy(self)
        -> tuple[tuple[int, int, int, int, int, float],int, int]
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

5.17.6 Yolov8 post-processing acceleration interfaces

sail.algo_yolov8_post_1output_async

For post-processing interfaces with a single output YOLOv8 model, internally implemented using thread pools.

__init__

Interface:

```
def __init__(
        self,
        shape: list[int],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20,
        input_use_multiclass_nms: bool=True,
        agnostic: bool=False)
```

Parameters:

- shape: list[int]

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- input_use_multiclass_nms: bool

Each detection box has multiple categories.

- agnostic: bool

Algorithm without class-specific NMS

push_npy

Input numpy. Only input with a batchsize of 1 is supported, or data is split before input and then sent to the interface.

Interface:

```
def push_npy(self,
        channel_idx: int,
        image_idx: int,
        data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],
        dete_threshold: float,
        nms_threshold: float,
        ost_w: int,
        ost_h: int,
        padding_left: int,
        padding_top: int,
        padding_width: int,
        padding_height: int) -> int
```

Parameters:

- channel_idx: int

The channel number of the input image.

- image_idx: int

The sequence number of the input image.

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]]

The input data.

- dete_threshold: float

Detection threshold sequence.

- nms_threshold: float

nms threshold

- ost_w: int

The width of original image.

- ost_h: int

The height of original image.

- padding_left: int

The starting point coordinate x of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_top: int

The starting point coordinate y of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_width: int

Fill the width of the image,

The width of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_height: int

The height of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

Returns:

Return 0 if successful, otherwise failed.

push_data

Input data. The value of batchsize other than 1 is supported.

Interface:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: TensorPTRWithName,
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

Parameters:

- channel_idx: int

The channel number of the input image.

- image_idx: int

The sequence number of the input image.

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],

The input data.

- dete_threshold: float

Detection threshold sequence.

- nms_threshold: float

nms threshold.

- ost_w: int

The width of original image.

- ost_h: int

The height of original image.

- padding_attrs: list[list[int]]

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
def get_result_npy(self)
      -> tuple[tuple[int, int, int, int, int, float],int, int]
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

sail.algo_yolov8_post_cpu_opt_1output_async

For post-processing interfaces with a single output YOLOv8 model, internally implemented using thread pools.

__init__

Interface:

```
def __init__(
        self,
        shape: list[int],
        network_w:int = 640,
        network_h:int = 640,
        max_queue_size: int=20,
        input_use_multiclass_nms: bool=True,
        agnostic: bool=False)
```

Parameters:

- shape: list[int]

The shape of the input data.

- network_w: int

The input width of the model, which defaults to 640.

- network_h: int

The input height of the model, which defaults to 640.

- max_queue_size: int

Maximum length of cached data.

- input_use_multiclass_nms: bool

Each detection box has multiple categories.

- agnostic: bool

Algorithm without class-specific NMS

push_npy

Input numpy. Only input with a batchsize of 1 is supported, or data is split before input and then sent to the interface.

Interface:

```
def push_npy(self,
        channel_idx: int,
        image_idx: int,
        data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],
        dete_threshold: float,
        nms_threshold: float,
        ost_w: int,
        ost_h: int,
        padding_left: int,
        padding_top: int,
        padding_width: int,
        padding_height: int) -> int
```

Parameters:

- channel_idx: int

The channel number of the input image.

- image_idx: int

The sequence number of the input image.

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]]

The input data.

- dete_threshold: float

Detection threshold sequence.

- nms_threshold: float

nms threshold

- ost_w: int

The width of original image.

- ost_h: int

The height of original image.

- padding_left: int

The starting point coordinate x of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_top: int

The starting point coordinate y of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_width: int

Fill the width of the image,

The width of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

- padding_height: int

The height of the fill image. Parameters can be obtained through the interface of general preprocessing or the inference interface with preprocessing, or can be calculated by yourselves.

Returns:

Return 0 if successful, otherwise failed.

push_data

Input data. The value of batchsize other than 1 is supported.

Interface:

```
def push_data(self,
    channel_idx: list[int],
    image_idx: list[int],
    input_data: TensorPTRWithName,
    dete_threshold: list[float],
    nms_threshold: list[float],
    ost_w: list[int],
    ost_h: list[int],
    padding_attrs: list[list[int]]) -> int
```

Parameters:

- channel_idx: int

The channel number of the input image.

- image_idx: int

The sequence number of the input image.

- data: numpy.ndarray[Any, numpy.dtype[numpy.float_]],

The input data.

- dete_threshold: float

Detection threshold sequence.

- nms_threshold: float

nms threshold.

- ost_w: int

The width of original image.

- ost_h: int

The height of original image.

- padding_attrs: list[list[int]]

The attribute list of the fill image, starting point coordinate x, starting point coordinate y, width after scaling, height after scaling.

Returns:

Return 0 if successful, otherwise failed.

get_result_npy

Get the final detection result.

Interface:

```
def get_result_npy(self)
      -> tuple[tuple[int, int, int, int, int, float],int, int]
```

Returns: tuple[tuple[left, top, right, bottom, class_id, score],channel_idx, image_idx]

- left: int

The left x coordinate of the detection result.

- top: int

The top y coordinate of the detection result.

- right: int

The right x coordinate of the detection result.

- bottom: int

The bottom y coordinate of the detection result.

- class_id: int

Category number of detection result.

- score: float

Score of detection result.

- channel_idx: int

The channel index of original image.

- image_idx: int

The image index of original image.

5.17.7 sort

sort_tracker_controller

SORT algorithm to track target

__init__

Interface:

```
def __init__(max_iou_distance:float = 0.7, max_age:int = 30,n_init:int = 3)
```

Parameters:

- max_iou_distance: float

Input. Maximum Intersection over Union (IoU) distance threshold used in the tracker.

- max_age: int

Input. The maximum number of frames that a tracked object can exist in the tracker.

- n_init: int

Input. The threshold for the number of initialization frames in the tracker.

process

Processing interface.

Interface:

```
def process(detected_objects:list[tuple[int, int, int, int, int, float]]) -> list[tuple[int, int, int,F
↪int, int, float, int]]
```

Parameters:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

Input. Detected objects.

Returns:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

Output。Tracked objects.

sort_tracker_controller_async

SORT algorithm to track target

__init__

Interface:

```
def __init__(max_iou_distance:float = 0.7,
        max_age:int = 30,
        n_init:int = 3,
        input_queue_size:int = 10,
        result_queue_size:int = 10)
```

Parameters:

- max_iou_distance: float

Input. Maximum Intersection over Union (IoU) distance threshold used in the tracker.

- max_age: int

Input. The maximum number of frames that a tracked object can exist in the tracker.

- n_init: int

Input. The threshold for the number of initialization frames in the tracker.

- input_queue_size: int

Input. Buffer size of the input queue.

- result_queue_size: int

Input. Buffer size of the result queue.

push_data

Interface:

```
def push_data(detected_objects:list[tuple[int, int, int, int, int, float]]) -> int
```

Parameters:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

Input. Detected objects.

Returns:

- int

Returns 0 on success and others on failure.

Returns:

- int

Returns 0 on success and others on failure.

get_result_npy

Interface:

```
def get_result_npy() -> tracked_objects:list[list[int, int, int, int, int, float, int]]
```

Returns:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

Output。Tracked objects.

5.17.8 deepsort

deepsort_tracker_controller

For the DeepSORT algorithm, tracking of targets is achieved by processing the detection results and extracting features.

__init__

Interface:

```
def __init__(max_cosine_distance:float,
        nn_budget:int,
        k_feature_dim:int,
        max_iou_distance:float = 0.7,
        max_age:int = 30,
        n_init:int = 3)
```

Parameters:

- max_cosine_distance: float

Input. Maximum threshold for cosine distance used in similarity calculation.

- nn_budget: int

Input. Maximum number for nearest neighbor search.

- k_feature_dim: int

Input. The feature dimension of the detected objects.

- max_iou_distance: float

Input. Maximum Intersection over Union (IoU) distance threshold used in the tracker.

- max_age: int

Input. The maximum number of frames that a tracked object can exist in the tracker.

- n_init: int

Input. The threshold for the number of initialization frames in the tracker.

process

Processing interface.

Interface 1:

```
def process(detected_objects:list[tuple[int, int, int, int, int, float]],
        feature:sail.Tensor)
        -> list[tuple[int, int, int, int, int, float, int]]
```

Parameters:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

Input. Detected objects.

- feature: sail.Tensor

Input. The features of the detected objects.

Returns:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

Output。Tracked objects.

Interface 2:

```
def process(detected_objects:list[tuple[int, int, int, int, int, float]],
        feature:numpy.array)
        -> list[tuple[int, int, int, int, int, float, int]]
```

Parameters:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

Input. Detected objects.

- feature: numpy.array

Input. The features of the detected objects.

Returns:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

Output。Tracked objects.

deepsort_tracker_controller

For the DeepSORT algorithm, tracking of targets is achieved by processing the detection results and extracting features.

__init__

Interface:

```
def __init__(max_cosine_distance:float,
        nn_budget:int,
        k_feature_dim:int,
        max_iou_distance:float = 0.7,
        max_age:int = 30,
        n_init:int = 3,
        queue_size:int = 10)
```

Parameters:

- max_cosine_distance: float

Input. Maximum threshold for cosine distance used in similarity calculation.

- nn_budget: int

Input. Maximum number for nearest neighbor search.

- k_feature_dim: int

Input. The feature dimension of the detected objects.

- max_iou_distance: float

Input. Maximum Intersection over Union (IoU) distance threshold used in the tracker.

- max_age: int

Input. The maximum number of frames that a tracked object can exist in the tracker.

- n_init: int

Input. The threshold for the number of initialization frames in the tracker.

- queue_size: int

Input. Buffer size of the result queue.

push_data

Interface 1:

```
def push_data(detected_objects:list[tuple[int, int, int, int, int, float]],
              feature:sail.Tensor) -> int
```

Parameters:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

Input. Detected objects.

- feature:sail.Tensor

Input. The features of the detected objects.

Returns:

- int

Returns 0 on success and others on failure.

Interface 2:

```
def push_data(detected_objects:list[tuple[int, int, int, int, int, float, int]],
              feature:list[numpy.array]) -> int
```

Parameters:

- detected_objects: list(tuple(left, top, right, bottom, class_id, score))

Input. Detected objects.

- feature: list[numpy.array]

Input. The features of the detected objects.

Returns:

- int

Returns 0 on success and others on failure.

get_result_npy

Interface:

```
def get_result_npy() -> tracked_objects:list[list[int, int, int, int, int, float, int]]
```

Returns:

- tracked_objects: list(tuple(left, top, right, bottom, class_id, score, track_id))

Output。Tracked objects.

5.17.9 bytetrack

bytetrack_tracker_controller

For the ByteTrack algorithm, tracking of targets is achieved by processing the detection results.

__init__

Interface:

```
def __init__(frame_rate:int = 30,
        track_buffer:int = 30)
```

Parameters:

· frame_rate: int

Input. Used to control the maximum number of frames allowed to disappear for tracked objects.

· track_buffer: int

Input. Used to control the maximum number of frames allowed to disappear for tracked objects.

process

Processing interface.

Interface:

```
def process(detected_objects:list[list[int, float, float, float, float, float, float, float]],
        tracked_objects:list[list[int, float, float, float, float, float, float, float, int]])
            -> int
```

Parameters:

· detected_objects: list[list[int, float, float, float, float, float, float, float]]

Input. Detected objects.

· tracked_objects: list[list[int, float, float, float, float, float, float, float, int]]

Output。Tracked objects.

Returns:

int

A return value of 0 indicates success, while other values indicate failure.

5.17.10 openpose

Use tpukernel to speed up openpose post-processing of part nms.

sail.tpu_kernel_api_openpose_part_nms

The part nms post-processing is accelerated using the Tensor Computing Processor Kernel, which currently only supports BM1684x, and the version of libsophon must be no less than 0.4.6 (v23.03.01).

__init__

Interface:

```
def __init__(
        self,
        device_id: int,
        network_c: int,
        module_file: str="/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_
↪kernel_module.so")
```

Parameters:

- device_id: int

Input. Device ID used.

- network_c: int

Input. Input channel of the model, corresponding to the number of keypoint channels.

- module_file: str

Input. File path of the kernel module, default is "/opt/sophon/libsophon-current/lib/tpu_module/libbm1684x_kernel_module.so" .

process

Processing interface.

Interface 1:

```
def process(self,
    input_data: TensorPTRWithName,
    shape: list[int],
    threshold: list[float],
    max_peak_num: list[int])
        -> tuple[list[list[int]], list[list[float]], list[list[int]]]
```

Parameters 1:

- input_data: TensorPTRWithName

Input. Input data.

- shape: list[int]

Input. Input data width and height.

- threshold: list[float]

Input. Detection threshold.

- max_peak_num: list[int]

Input. The max number of detected peaks.

Interface 2:

```
def process(self,
    input_data: dict[str, Tensor],
    shape: list[int],
    threshold: list[float],
    max_peak_num: list[int])
        -> tuple[list[list[int]], list[list[float]], list[list[int]]]
```

Parameters 2:

- input_data: dict[str, Tensor]

Input. Input data.

- shape: list[int]

Input. Input data width and height.

- threshold: list[float]

Input. Detection threshold.

- max_peak_num: list[int]

Input. The max number of detected peaks.

Returns:

tuple[list[list[int]], list[list[float]], list[list[int]]]

- 1st output: list[list[int]]

The number of the detected peaks in each channel.

- 2nd: list[list[float]]

The scores of all detected peaks.

- 3rd: list[list[int]]

The flatten coordinates of all detected peaks.

reset_network_c

Update the channel number of the input.

Interface:

```
def reset_network_c(self, network_c_new: int) -> int
```

Parameters:

- network_c_new: int

The number of channels to be updated.

Returns:

A return value of 0 indicates success, while other values indicate failure.

5.17.11 nms_rotated

sail.nms_rotated

nms with rotating boxes

sail.nms_rotated

get the retained index of rotating boxes according to the threshold

Interface:

```
def nms_rotated(boxes: numpy.ndarray[numpy.float32], scores: numpy.ndarray[numpy.
↪float32], threshold: float)-> list[int]:
```

Parameters:

- boxes: numpy.ndarray[numpy.float32]

all rotating boxes, shape is (N,5), box is represented by [x,y,w,h,theta]

- scores: numpy.ndarray[numpy.float32]

the confidence corresponding to the rotating boxes, shape is (N,)

- threshold: float

IOU threshold

Returns:

the retained index of rotating boxes according to the threshold

Sample:

```
import sophon.sail as sail
import numpy as np

boxes = np.load("boxes_data.npy")
scores = np.load("scores_data.npy")
threshold = 0.3

indices = sail.nms_rotated(boxes, scores, threshold)

# show all the boxes that remained
print(boxes[indices])
```

5.17.12 Yolov8_seg TPU post-processing acceleration interfaces

sail.algo_yolov8_seg_post_tpu_opt

For the YOLOv8 segmentation model, post-processing has been accelerated using TPU.

__init__

Interface:

```
def __init__(
        self,
        bmodel_file: str,
        dev_id: int,
        detection_shape: list[int],
        segmentation_shape: list[int],
        network_h: int,
        network_w: int)
```

Parameters:

- bmodel_file: str

Input. The TPU getmask bmodel path.

- dev_id: int

Input. device id.

- detection_shape: list[int]

Input. The shapes of detection head.

- segmentation_shape: list[int]

Input. The shapes of segmentation head, that is the shapes of Prototype Mask.

- network_h: int

Input. The input height of yolov8_seg network.

- network_w: int

Input. The input width of yolov8_seg network.

process

Processing interface.

Interface 1:

```
def process(self,
    detection_input: TensorPTRWithName,
    segmentation_input: TensorPTRWithName,
    ost_h: int,
    ost_w: int,
    dete_threshold: float,
    nms_threshold: float,
    input_keep_aspect_ratio: bool,
    input_use_multiclass_nms: bool)
        -> list[tuple[left, top, right, bottom, score, class_id, contour, mask]]
```

Parameters 1:

- detection_input: TensorPTRWithName

Input. The input data of detection head.

- segmentation_input: TensorPTRWithName

Input. The input data of segmentation head, that is Prototype Mask.

- ost_h: int

Input. Original image height.

- ost_w: int

Input. Original image width.

- dete_threshold: float

Input. Detection threshold.

- nms_threshold: float

Input. NMS threshold.

- input_keep_aspect_ratio: bool

Input. Whether to keep aspect ratio in images.

- input_use_multiclass_nms: bool

Input. Whether to use multiclass nms.

Interface 2:

```
def process(self,
    detection_input: dict[str, Tensor],
    segmentation_input: dict[str, Tensor],
    ost_h: int,
    ost_w: int,
    dete_threshold: float,
    nms_threshold: float,
    input_keep_aspect_ratio: bool,
    input_use_multiclass_nms: bool)
        -> list[tuple[left, top, right, bottom, score, class_id, contour, mask]]
```

Parameters 2:

· detection_input: dict[str, Tensor]

Input. The input data of detection head.

· segmentation_input: dict[str, Tensor]

Input. The input data of segmentation head, that is Prototype Mask.

· ost_h: int

Input. Original image height.

· ost_w: int

Input. Original image width.

· dete_threshold: float

Input. Detection threshold.

· nms_threshold: float

Input. NMS threshold.

· input_keep_aspect_ratio: bool

Input. Whether to keep aspect ratio in images.

· input_use_multiclass_nms: bool

Input. Whether to use multiclass nms.

Returns:

list[tuple[left, top, right, bottom, score, class_id, contour, mask]]

· left: int

The leftmost x-coordinate of the detection box.

· top: int

The topmost y-coordinate of the detection box.

· right: int

The rightmost x-coordinate of the detection box.

- bottom: int

The bottommost y-coordinate of the detection box.

- class_id: int

The class ID of the object within the detection box..

- score: float

The score of the object within the detection box..

- contour: list[float]

The contour of the object within the detection box..

- mask: numpy.ndarray

The segmentation mask of the object within the detection box.

CHAPTER 6

Appendix

6.1 Get Python3 for cross-compilation on an X86 host

1. Install the download tool dfss

```
pip3 install dfss --upgrade
```

2. Use dfss to download the Python3 required for compilation according to the version

- Python3.5

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.5.9.tar.gz
```

- Python3.6

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.6.5.tar.gz
```

- Python3.7

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.7.3.tar.gz
```

- Python3.8

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.8.2.tar.gz
```

- Python3.9

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.9.0.tar.gz
```

- Python3.10

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.10.0.tar.gz
```

- Python3.11

```
python3 -m dfss --url=open@sophgo.com:/toolchains/pythons/Python-3.11.0.tar.gz
```

6.2 Get weight files of images

1. Open the splicing parameter tuning tool in the Windows system, and the download link is as follows:

```
python3 -m dfss --url=open@sophgo.com:/sophon-stream/dwa_blend_encode/
↪stitchtool_circular_fisheye_v240103.7.zip
```

2. Stitch two pictures together to get a stitched picture The command description is as follows:

```
gen_gridinfo.exe mode result_path left_image_path right_image_path fusion_start_
↪position fusion_width
```

Take a picture with a width of 2240 as an example:

```
gen_gridinfo.exe 1 ./result_path ./L/left.png ./R/right.png 2144 32
```

Note: The last two values need to be divisible by 32, and their sum does not need to be equal to 2240. When merging, the right edge of the left image will directly take the right image. For example, in the example, 2240 – 2144 – 32 = 64, that is, the right edge of the left image with 64 pixels will directly take the right image.

3. Adjust stitching parameters according to the stitching diagram

```
gen_gridinfo.exe 0 ./L ./L/left.bmp x_shift y_shift theta_x theta_y theta_z
```

- x_shift: Positive numbers move the image to the right, and negative numbers move the image to the left.
- y_shift: Positive numbers move the image downward, and negative numbers move the image upward.
- theta_x: rotation around the horizontal axis
- theta_y: rotation around the vertical axis
- theta_z: rotation perpendicular to the paper

Note: These three parameters are generally not used and are set to 0 by default.

Generally, the right image is fixed and the left image is adjusted to align. The size of the X direction during stitching is adjusted with 32 as the minimum adjustment unit, and the best stitching image is selected as the final result.

```
gen_gridinfo.exe 1 ./result_path ./L/left.png ./R/right.png 2144 32
```

4. After the splicing is completed, the weight file is saved in result_path

