mera Documentation

Release 2.3.2

mera

23/05/2025

CONTENTS

CHAPTER

ONE

MERA USER MANUAL

1.1 Introduction

1.1.1 Description

The EdgeCortix© MERA™ software stack provides a compiler and the necessary tools to convert machine learning models into C source code compatible with range of Renesas MCUs powered by Arm Ethos-U NPUs.

MERA software stack generates C source code while ensuring compatibility and tight integration the with Renesas e^2 studio.

It also ships with MERA Quantizer, a post-training static INT8 quantizer, allowing more demanding models to meet the memory and latency constraints typical of microcontrollers and Ethos-U accelerators.

1.1.2 Overall workflow

- **Import** MERA accept models from the most used ML frameworks as PyTorch, TensorFlow Lite and ONNX.
- **Compile** the compiler lowers the graph, applies operator fusion, inserts fast math libraries, and emits plain C99 source code that calls either CMSIS-NN (for Cortex-M CPUs) or the Ethos-U driver depending on the targets. This C code builds inside Renesas e^2 studio providing support for HAL, Ethos-U drivers, OSPI Flash drivers, CMSIS-NN/CMSIS DSP libraries and other stacks necessary to run machine learning models on the different supported targets.
- **Quantize** when targeting more resource limited devices, MERA Quantizer helps to convert weights and activations to INT8 precision. This tool performs layer-wise calibration on representative data, calculates quantization parameters and emits a quantized model that can be further compiled by the MERA compiler.
- **Build & deploy** Renesas e^2 studio is compatible with the code generated by MERA software stack. The same flow works for pure MCU targets or for MCU + NPU combinations.

1.1.3 Supported embedded platforms

Currently, MERA software stack provides support for Renesas embedded platforms:

- EK-RA8P1 board (device R7KA8P1KFLCAC)

1.1.4 Software components overview

MERA software stack

Provided as a Python PIP package for Ubuntu 22.04 LTS that can be installed on a Python 3.10 virtual environment.

TODO: explain components (compiler, quantizer, targets, platforms…etc)

ARM Vela compiler

When targeting a Renesas device powered by Arm Ethos-U NPU the MERA software stack will automatically leverage the Arm Vela compiler. Those parts of the machine learning model that can accelerated with the NPU will be processed with the Arm Vela compiler in order to obtain valid assembly for the Ethos-U NPU.

The version of ARM Vela compiler used to compile those subgraphs assigned to the Ethos-U55 target is 4.2.0. Note that when installing the MERA software stack PIP package it will automatically pull Arm Vela as a dependency.

1.2 MERA Installation - Ubuntu Linux

In order to install MERA 2.3.1 MCU on supported environment you will need:

- A machine with Ubuntu 22.04 installation is recommended as this was the version used for testing
- A working installation of PyEnv or other Python virtual environment management system that provides Python version 3.10.x.

1.2.1 Installation steps

System dependencies necessary to create environments and run demos:

```
sudo apt update; sudo apt install build-essential cmake python3-venv python3-pip
```

Recommended: use the default Python installation

Because MERA software stack is compatible by default with the base system Python version provided by Ubuntu 22.04 we can create a virtual environment as follows:

```
python3 -m venv mera-env
source mera-env/bin/activate
pip install --upgrade pip && pip install decorator typing_extensions psutil attrs pybind11
```

Your prompt should now show that you are under a virtual environment *mera-env*:

```
(mera-env) user@compute:~$
```

Alternative: PyEnv installation

If PyEnv is preffered over the base system Python installation you can get started with:

```
# pyenv dependencies
sudo apt update; sudo apt install build-essential libssl-dev zlib1g-dev \
libbz2-dev libreadline-dev libsqlite3-dev curl git cmake \
libncursesw5-dev xz-utils tk-dev libxml2-dev libxmlsec1-dev libffi-dev liblzma-dev

# actual installation of PyEnv
curl https://pyenv.run | bash
```

The installation of PyEnv recommends to do some post-installation steps that involve to modify your .bashrc file in order to easily create virtual environments:

```
echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
echo '[[ -d $PYENV_ROOT/bin ]] && export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(pyenv init - bash)"' >> ~/.bashrc
```

To create a Python Virtual Environment for MERA named "mera-env" using PyEnv:

```
MENV=mera-env; pyenv install 3.10.15 && pyenv virtualenv 3.10.15 $MENV && pyenv activate $MENV && \
pip install --upgrade pip && \
pip install decorator typing_extensions psutil attrs pybind11
```

Your prompt should now show that you are under a virtual environment *mera-env*:

```
(mera-env) user@compute:~$
```

Install MERA

Finally install MERA on the virtual environment *mera-env*:

```
pip install ./mera-2.3.1+pkg.1503-cp310-cp310-manylinux_2_27_x86_64.whl
```

where the versions may vary depending on the MERA release used.

At this point MERA should be ready to use. You can confirm with the following example:

```
python -c "import mera;print(dir(mera))"
```

1.3 MERA Installation - Windows 11

MERA software stack is also provided as PIP package compatible with Windows 11.

The only requirement needed on Windows are C++ runtime libraries. Please download and install [this package](https://aka.ms/vs/17/release/vc_redist.x64.exe) in order to provide these libraries.

TODO

1.4 How to deploy models with MERA

The following code shows how to use the MERA deployment API to compile an already quantized TFLite model on a board with Ethos-U55 support:

```
import os
import mera

from mera import Platform, Target

def needs_ospi(file, size_mb):
    return os.path.getsize(file) / (1024 * 1024) > size_mb

def deploy_mcu(model_path, deploy_dir, with_ethos, with_ospi, with_ref_data):
    try:
        with mera.Deployer(deploy_dir, overwrite=True) as deployer:
            model = mera.ModelLoader(deployer).from_tflite(model_path)

            platform = Platform.MCU_ETHOS if with_ethos else Platform.MCU_CPU

            # ARM Vela options
            # only effective if Platform.MCU_ETHOS is selected, ignored otherwise
            vela_config = {}
            vela_config['enable_ospi'] = needs_ospi(model_path, 1.5)
            vela_config['sys_config'] = 'RA8P1'
            vela_config['memory_mode'] = 'Sram_Only'
            vela_config['accel_config'] = 'ethos-u55-256'
            vela_config['optimise'] = 'Performance'
            vela_config['verbose_all'] = False

            # MCU C code generation options
            mcu_config = {}
            mcu_config['suffix'] = ''
            mcu_config['weight_location'] = 'flash' # other option is: 'iram'

            # On other scripts we could use True and it will not generate
            # x86 related source code (python bindings for example)
            mcu_config['use_x86'] = False

            # generation of reference data from original model format
            # using the original runtime is only supported for TFLite models
            enable_ref_data = with_ref_data and model_path.suffix == '.tflite'
```

(continues on next page)

(continued from previous page)

```
            deployer.deploy(model,
                            mera_platform=platform,
                            target=Target.MCU,
                            vela_config=vela_config,
                            mcu_config=mcu_config,
                            enable_ref_data=enable_ref_data)

    except Exception as e:
        print(str(e))
```

This release provides some tested models, if the models provided are for example:

```
models_int8/
├── ad_medium_int8.tflite
├── kws_micronet_m.tflite
├── mobilenet_v2_1.0_224_INT8.tflite
├── person-det.tflite
├── rnnoise_INT8.tflite
├── vww4_128_128_INT8.tflite
├── wav2letter_int8.tflite
└── yolo-fastest_192_face_v4.tflite
```

then by running the provided script *scripts/mcu_deploy.py* we can compile the model for MCU only:

```
cd scripts/
python mcu_deploy.py --ref_data ../models_int8 deploy_qtzed
```

you will get the following results:

```
deploy_qtzed
├── ad_medium_int8_no_ospi
├── kws_micronet_m_no_ospi
├── mobilenet_v2_1.0_224_INT8_ospi
├── person-det_no_ospi
├── rnnoise_INT8_no_ospi
├── vww4_128_128_INT8_no_ospi
├── wav2letter_int8_ospi
└── yolo-fastest_192_face_v4_no_ospi
```

same but enabling Ethos-U support:

```
cd scripts/
python mcu_deploy.py --ethos --ref_data ../models_int8 deploy_qtzed_ethos
```

When Ethos-U support is enabled, each of the directories contain a deployment of the corresponding model for MCU + Ethos-U55 target:

```
├── person-det_no_ospi
│   ├── build
│   │   └── MCU
│   │       ├── compilation
│   │       │   ├── mera.plan
│   │       │   ├── src # compilation results: C source code and C++ testing support code
│   │       │   │   ├── CMakeLists.txt
│   │       │   │   ├── compare.cpp
│   │       │   │   ├── compute_sub_0000.c # CPU subgraph generated C source code
│   │       │   │   ├── compute_sub_0000.h
│   │       │   │   ├── ...
│   │       │   │   ├── ethosu_common.h
│   │       │   │   ├── hal_entry.c          # HAL entry example
│   │       │   │   ├── kernel_library_int.c # kernel library if CPU subgraphs are present
│   │       │   │   ├── ...
│   │       │   │   ├── model.c
│   │       │   │   ├── model.h
│   │       │   │   ├── model_io_data.c
│   │       │   │   ├── model_io_data.h
│   │       │   │   ├── python_bindings.cpp
│   │       │   │   ├── sub_0001_command_stream.c # Ethos-U55 subgraph generated C source code
│   │       │   │   ├── sub_0001_command_stream.h
│   │       │   │   ├── sub_0001_invoke.c
│   │       │   │   ├── sub_0001_invoke.h
```

(continues on next page)

(continued from previous page)

```
│   │       │   │   └── ...
│   │       │   ├── ...
│   │       ├── deploy_cfg.json
│   │       ├── ir_dumps
│   │       │   ├── person-det_can.dot
│   │       │   ├── ...
│   │       ├── person-det_after_canonicalization.dot
│   │       └── person-det_subgraphs.dot
│   ├── logs
│   ├── model
│   │   └── input_desc.json
│   └── project.mdp
```

Please reference to the provided pre-generated e2studio projects to see how the generated C code under *build/MCU/compilation/src* can be incorporated into a e2studio project.

1.5 Quantize and deploy models with MERA

If the starting point it is a Float32 precision model it is possible to use the MERA Quantizer to first quantize the model and finally deploy with MCU/Ethos-U55 support.

The following is a sample of how to quantize a model with the MERA Quantizer API:

```python
with mera.Deployer(str(deploy_dir), overwrite=True) as deployer:
    if model_path.suffix == '.tflite':
        mera_model = mera.ModelLoader(deployer).from_tflite(str(model_path))
    elif model_path.suffix == '.onnx':
        mera_model = mera.ModelLoader(deployer).from_onnx(str(model_path), shape_mapping=SYMBOLIC_DIMS)
    elif model_path.suffix == '.pte':
        mera_model = mera.ModelLoader(deployer).from_executorch(str(model_path))
    else:
        raise ValueError("Unsupported model format: " + model_path.suffix)

    cal_data = generate_input_data(mera_model, num_cal)
    qtzer = mera.Quantizer(deployer,mera_model,quantizer_config=quantizer_config,mera_platform=platform)
    qtz_path = Path(result_path)
    res_path = qtz_path / 'model.mera'
    qty = qtzer.calibrate(cal_data).quantize().evaluate_quality(cal_data[:1])
    Q = qty[0].out_summary()[0]
    if Q["psnr"] < 5:
        model.status = Status.ERROR_PSNR
        model.last_error = f'PSNR too low: {Q["psnr"]}'
        return {'inputs': None, 'outputs': None}

    qtzer.save_to(res_path)
    print(f'Successfully quantized model with quality: {Q["psnr"]} psnr, {Q["score"]} score')
    return res_path
```

Once quantized, we can deploy the quantized model by using using *.from_quantized_mera()* instead of *.from_tflite()*:

```python
with mera.Deployer(str(deploy_dir), overwrite=True) as deployer:

    mera_model = mera.ModelLoader(deployer).from_quantized_mera(model_path)

    platform = Platform.MCU_ETHOS if with_ethos else Platform.MCU_CPU

    # ARM Vela options
    # only effective if Platform.MCU_ETHOS is selected, ignored otherwise
    vela_config = {}
    vela_config['enable_ospi'] = needs_ospi(model_path, 1.5)
    vela_config['sys_config'] = 'RA8P1'
    vela_config['memory_mode'] = 'Sram_Only'
    vela_config['accel_config'] = 'ethos-u55-256'
    vela_config['optimise'] = 'Performance'
    vela_config['verbose_all'] = False

    # MCU C code generation options
    mcu_config = {}
    mcu_config['suffix'] = ''
    mcu_config['weight_location'] = 'flash' # other option is: 'iram'
```

(continues on next page)

(continued from previous page)

```
    # On other scripts we could use False and it will not generate
    # x86 related source code (python bindings for example),
    mcu_config['use_x86'] = True

    # generation of reference data from original model format
    # using the original runtime is only supported for TFLite models
    enable_ref_data = with_ref_data and model_path.suffix == '.tflite'

    deployer.deploy(mera_model,
                    mera_platform=platform,
                    target=Target.MCU,
                    vela_config=vela_config,
                    mcu_config=mcu_config,
                    enable_ref_data=enable_ref_data)
```

1.6 Full quantization demo

We provide a script that given a model (either with .tflite, .onnx or .pte extension) will:

- calibrate and quantize the model with random data
- deploy the model for the MCU C Codegen target and generate C source code
- compile the auto-generated Python bindings for the C source code
- execute the C source code through the Python bindings
- compare the MERA Interpreter results and the C source code results

To run this script:

```
cd scripts/

# deploy for MCU only
python mcu_quantize.py ../models_fp32 deploy_mcu

# deploy for MCU+Ethos-U55
python mcu_quantize.py -e ../models_fp32_ethos deploy_ethos
```

1.7 Guide to the generated C source code

After processing a model with the MERA compiler you will find several files on your deployment directory. This include some deploying artifacts generated during compilation that are worth to be kept around for debugging purposes.

The most important output that MERA generates is found under the directory **<deployment_directory>build/MCU/compilation/src**. This directory contains the model converted into a set of C99 source code files.

1.7.1 Runtime API - MPU only deployment

When a model is converted into source code with MERA compiler without Ethos-U support, all the operators in the model being deployed will be prepared to be run on CPU/MCU only. In this case, the generated code will refer to a single subgraph **compute_sub_0000<suffix>**, by default, when no suffix is provided, the name of the header that need to included on your application entry point is **compute_sub_0000.h**.

This header provides the declaration of a C function that if called it will run the model with the provided inputs and write the results on the provided output buffers:

```
enum BufferSize_sub_0000 {
  kBufferSize_sub_0000 = <intermediate_buffers_size>
};

void compute_sub_0000(
```

(continues on next page)

(continued from previous page)

```
  // buffer for intermediate results
  uint8_t* main_storage, // should provide at least <intermediate_buffers_size> bytes of storage

  // inputs
  const int8_t <input_name>[xxx], // 1,224,224,3

  // outputs
  int8_t <output_name>[xxx]  // 1,1000
);
```

It provides to the user the possibility of providing a buffer to hold intermediate outputs of the model. And this size if provided in compilation time as the value **kBufferSize_sub_0000** so the user can use this size to allocate the buffer on the stack, the heap or a custom data section.

1.7.2 Runtime API - MPU + Ethos-U deployment

If Ethos-U support is enabled during conversion into source code with MERA compiler then an arbitrary amount of subgraphs for either CPU or Ethos-U will be generated. Each of these subgraphs will correspond to generated C functions to run the corresponding section of the model on CPU or Ethos. Each function call will get its inputs from previous outputs of other subgraphs and write its outputs on buffers that are designated to became again inputs to other functions and so on. To make easier for the user to invoke these models where CPU and NPU are involved, the generated code will automate this process and provide a single function that will orchestrate the calls to the different computation units named **void RunModel(bool clean_outputs)** and helpers to access to each of the input and output areas at **model level** not per subgraph level. The runtime API header when Ethos-U is enabled can be found on a file named **model.h** under the same directory **<deployment_directory>/build/MCU/compilation/src**.

For example, after enabling Ethos-U support for a model with two inputs and three outputs MERA provides the next runtime API:

```
// invoke the whole model
void RunModel(bool clean_outputs);

  // Model input pointers
float* GetModelInputPtr_input0();
float* GetModelInputPtr_input1();

  // Model output pointers
float* GetModelOutputPtr_out0();
float* GetModelOutputPtr_out1();
float* GetModelOutputPtr_out2();
```

The function **GetModelInputPtr_input0** provides access to the buffer where the user can write the first input of the model and **GetModelInputPtr_input1** gives us the pointer where the user can copy the second input.

To run the model and all the CPU or NPU units needed to be invoked to do inference with the deployed model, the user should invoke to the **RunModel()** function. The parameter **clean_outputs** should be used only for debugging purposes because it will set to zero all the output buffers used by an NPU unit before invoking it. Recommended value for the parameter **clean_outputs** is **false**, as it will not incur into extra time expend on clearing these buffers.