Canonical

Running AI Application on RZ/V2L with Ubuntu images

Prepared by: Hon Ming Hui

Reviewed by: Asa Mirzaieva

Prepared on: Mar 24, 2026

Version: 1.3

1. Prerequisites

To set up the RZ/V2L with the Ubuntu image, please refer to the Quick Start Guide available at https://ubuntu.com/download/renesas-iot.

2. Basics of AI applications on RZ/V2L

2.1.

2.2. Major components

The AI application comprises three principal components:

1. The compiled model tailored for the RZ/V2L platform.
2. The runtime library responsible for interaction with the DRP-AI driver.
3. The user application.

2.3. Model compilation

Renesas provides the Robust Unified Heterogeneous Model Integration (RUHMI) framework for the optimization and deployment of AI models. The project repository is available on GitHub: GitHub - renesas-rz/rzv_drp-ai_tvm.

Model compilation is typically executed on a powerful host machine rather than the target device (RZ/V2L). Renesas offers the necessary setup, utilizing Ubuntu 22.04 as the base operating system. Follow these steps to compile a model for RZ/V2L. Further details on the setup can be found at https://github.com/renesas-rz/rzv_drp-ai_tvm/tree/main/setup (important note: an x86 architecture machine is required).

2.3.1. Prerequisites

- OS : Ubuntu 22.04 on an amd64 machine
- Python: 3.10
- git, wget, unzip, docker

2.3.2. Prepare the Dockerfile

Download the Dockerfile from the official repository:

```shell
# Download Dockerfile
wget
https://raw.githubusercontent.com/renesas-rz/rzv_drp-ai_tvm/main/Dockerfile
-O Dockerfile
```

2.3.3. Download the SDK and Translator

Target versions:

- DRP-AI Translator V1.90 Installer: v1.90 (renesas.com account required)
- RZ/V2L AI Software Development Kit: v7.0.0 (renesas.com account required)

2.3.4. Prepare SDK Files and Translator files

```shell
export SDK_FILE="RTK0EF0160F07000SJ.zip"
# Extract toolchain script
mkdir -p sdk_temp
unzip -q ${SDK_FILE} -d sdk_temp
export TOOLCHAIN_SCRIPT=$(find sdk_temp -type f -name "*toolchain**.sh" -o
-name "*toolchain*.sh" | head -n 1)

cp "$TOOLCHAIN_SCRIPT" ./
TOOLCHAIN_SCRIPT="./$(basename $TOOLCHAIN_SCRIPT)"
chmod +x ${TOOLCHAIN_SCRIPT}
# Clean up temporary directory
rm -rf sdk_temp

unzip r20ut5035ej0190-drp-ai-translator.zip
```

2.3.5. Prepare Build Directory

```shell
# Create build directory
mkdir -p docker_build

export TRANSLATOR_FILE="DRP-AI_Translator-v1.90-Linux-x86_64-Install"

# Copy only required files to build directory
cp Dockerfile docker_build/
cp r20ut5035ej0190-drp-ai-translator/${TRANSLATOR_FILE} docker_build/
cp ${TOOLCHAIN_SCRIPT} docker_build/
```

2.3.6. Build Docker image and run the container

```shell
cd docker_build

# Set product type
export PRODUCT=V2L

# build the image - this will take a while
docker build -t \
drp-ai_tvm_image -f Dockerfile \
--build-arg PRODUCT=${PRODUCT} .

# run the container
docker run -it --name drp-ai_tvm_container drp-ai_tvm_image
```

Tip: If you encounter an error like "404 Not Found," try adding the `--no-cache` option with "`docker build`".

2.3.7. Compile ONNX models

We picked ResNet18, which is not the best model, but it's small and a good example.

```shell
# inside the newly created container,
# download onnx model from official ONNX model zoo
wget
https://github.com/onnx/models/raw/main/validated/vision/classification/resne
t/model/resnet18-v1-7.onnx


# compile the model
python3 compile_onnx_model.py \
    ./resnet18-v1-7.onnx \
    -o resnet18_onnx \
    -s 1,3,224,224 \
    -i data

# the output should look like this
root@cf59e4079094:/drp-ai_tvm/tutorials# find ./resnet18_onnx/
./resnet18_onnx/
./resnet18_onnx/deploy.json
./resnet18_onnx/deploy.so
./resnet18_onnx/deploy.params
./resnet18_onnx/preprocess
./resnet18_onnx/preprocess/pp_weight.dat
./resnet18_onnx/preprocess/drp_param.bin
./resnet18_onnx/preprocess/pp_addrmap_intm.txt
./resnet18_onnx/preprocess/aimac_desc.bin
./resnet18_onnx/preprocess/pp_drpcfg.mem
./resnet18_onnx/preprocess/drp_param_info.txt
./resnet18_onnx/preprocess/drp_desc.bin
```

2.4. TVM Runtime library

The TVM runtime, which communicates with the DRP AI driver, is provided as a pre-built library, available at https://github.com/renesas-rz/rzv_drp-ai_tvm/tree/main/obj/build_runtime.

For V2L, please use the V2M files.

As the TVM Runtime library evolves in tandem with the Model compiler, it is advisable for the final application to consistently utilize a specific, unified version of the TVM components.

2.5. User application

This application integrates all components to form the final user application. Within the Ubuntu environment, the recommended method for creating such an application is through snap.

There is a tutorial application that comes with the DRP AI repository https://github.com/renesas-rz/rzv_drp-ai_tvm/tree/main/apps. The corresponding example as snap is available at https://github.com/canonical/rzv_drp-ai_tvm_snap. The primary configuration file for a snap is the snapcraft.yaml. The following provides an examination of key code snippets from this example.

1. The `tvm-runtime` component is responsible for retrieving the runtime libraries. It will also stage all the requisite header files necessary for application compilation.

```
None
parts:
  tvm-runtime:
    plugin: nil
    source: https://github.com/renesas-rz/rzv_drp-ai_tvm.git
    source-type: git

# Refer to the full yaml file for the detail steps
```

2. The subsequent section addresses the compilation of the application utilizing CMake. The Snapcraft CMake plugin will execute this task, provided the appropriate CMAKE parameters are configured as detailed below.

```
None
 tutorial-app:
    after: [tvm-runtime]
    plugin: cmake
    source: ${CRAFT_PROJECT_DIR}/../parts/tvm-runtime/src/apps
    source-type: local
    cmake-parameters:
      - -DCMAKE_INSTALL_PREFIX=/usr/
      - -DCMAKE_C_COMPILER=aarch64-linux-gnu-gcc
      - -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++
    build-environment:
      - PRODUCT: V2L
      - TVM_ROOT: ${CRAFT_PROJECT_DIR}/../parts/tvm-runtime/src
```

3. The `stage-packages` section within the `tutorial-app` component is employed to incorporate all necessary dependencies for compiling the AI applications and subsequently installing them within the resulting snap. Utilizing existing packages from the Ubuntu archive within a snap environment is a relatively uncomplicated process.

```
None
    stage-packages:
      - libmmngr1:$CRAFT_ARCH_BUILD_FOR
      - libmmngrbuf1:$CRAFT_ARCH_BUILD_FOR
```

4. The snap includes the compiled model for demonstration purposes. The pre-compiled `ResNet18 ONNX` model is placed in a directory named `resnet18_onnx` while staging the part `tvm-runtime`.
5. The final step is to expose the application to the user

```
None
apps:
  tutorial-app:
    command: usr/bin/launch-tutorial.sh

    environment:
      LD_LIBRARY_PATH:
$LD_LIBRARY_PATH:$SNAP/usr/lib:$SNAP/usr/lib/$CRAFT_ARCH_TRIPLET_BUILD_FOR/la
pack:$SNAP/usr/lib/$CRAFT_ARCH_TRIPLET_BUILD_FOR/blas
    plugs: [camera, opengl, wayland]
```

The wrapper script `launch-turial.sh` calls the application `tutorial_app_v2ml` which finds the compiled model under `resnet18_onnx.`

3. Install and run the snap

`rzv-drp-ai-tvm-examples` is published in the Edge and Beta channels in the snap store. To install and run the application simply do:

```
Shell
$ sudo snap install --channel=beta rzv-drp-ai-tvm-examples --devmode
```

```
$ sudo rzv-drp-ai-tvm-examples.tutorial-app
```

If no argument is passed, the application will process the sample.bmp image embedded in the snap. It is also possible to pass another image file to the application such as:

```
Shell
$ sudo rzv-drp-ai-tvm-examples.tutorial-app ~/beagle.bmp
```

The input files should be placed in the rootfs where the snap can access it (such as /home/ubuntu).

A sample run of the application should look like:

```
Shell
$ sudo rzv-drp-ai-tvm-examples.tutorial-app ~/beagle.bmp
SNAP = /snap/rzv-drp-ai-tvm-examples/2
Processing: /home/ubuntu/beagle.bmp
[2026-03-23 14:56:54.846] [console] [info] MERA 1.0 (drp-tvm) Runtime
Runtime memory usage: start=0x80000000, end=0x82259620
Preruntime memory allocation: start=0x83000000
[INFO] Input  (H, W, C, unit byte_size)=(480, 640, 3, 1)
[INFO] Output (H, W, C, unit byte_size)=(224, 224, 3, 4)
PreProcessing Parameter List
  pre_in_shape_w =      640
  pre_in_shape_h =      480
  pre_in_addr    = 83000000
  pre_in_format  =     fffe(RGB)
  pre_out_format =     fffd(BGR)
  resize_alg     =        1
  resize_w       =      224
```

```
  resize_h       =      224
  cof_add        = -123.6875, -116.2500, -103.5000
  cof_mul        = 0.0171, 0.0175, 0.0174
  crop_tl_x      =    65535
  crop_tl_y      =    65535
  crop_w         =    65535
  crop_h         =    65535
[INFO] Changed pre_in_format: 0xfffd(BGR)
[INFO] Changed pre_out_format: 0xfffe(RGB)
[INFO] Input  (H, W, C, unit byte_size)=(480, 640, 3, 1)
[INFO] Output (H, W, C, unit byte_size)=(224, 224, 3, 4)
[TIME] UpdateParamData() Processing Time: 0.24 msec.
[TIME] UpdateWeightData() Processing Time: 0.00 msec.
[TIME] parammodified Processing Time: 0.04 msec.
[TIME] weightmodified Processing Time: 0.00 msec.
PreProcessing Parameter List
  pre_in_shape_w =      640
  pre_in_shape_h =      480
  pre_in_addr    = b0000000
  pre_in_format  =     fffd(BGR)
  pre_out_format =     fffe(RGB)
  resize_alg     =        1
  resize_w       =      224
  resize_h       =      224
  cof_add        = -123.6875, -116.2500, -103.5000
  cof_mul        = 0.0171, 0.0175, 0.0174
  crop_tl_x      =    65535
  crop_tl_y      =    65535
  crop_w         =    65535
```

```
  crop_h         =    65535
[TIME] PreRuntime DRP-AI processing time : 2.91 msec
[TIME] GetResult() Processing Time : 2.13 msec
[TIME] Pre Processing Time: 5.93 msec.
Running tvm runtime
[TIME] AI Processing Time: 30.74 msec.
Output data type : FP16.
Result ----------------------
  Top 1 [ 66.0%] : [beagle]
  Top 2 [ 13.0%] : [English foxhound]
  Top 3 [ 11.4%] : [Walker hound, Walker foxhound]
  Top 4 [  5.9%] : [basset, basset hound]
  Top 5 [  0.7%] : [bloodhound, sleuthhound]
```