Canonical

Running AI Applications on RZ/V2L with Ubuntu images

Prepared by: Hon Ming Hui

Reviewed by: Devrim Ayyildiz

Prepared on: 25 Feb 2026

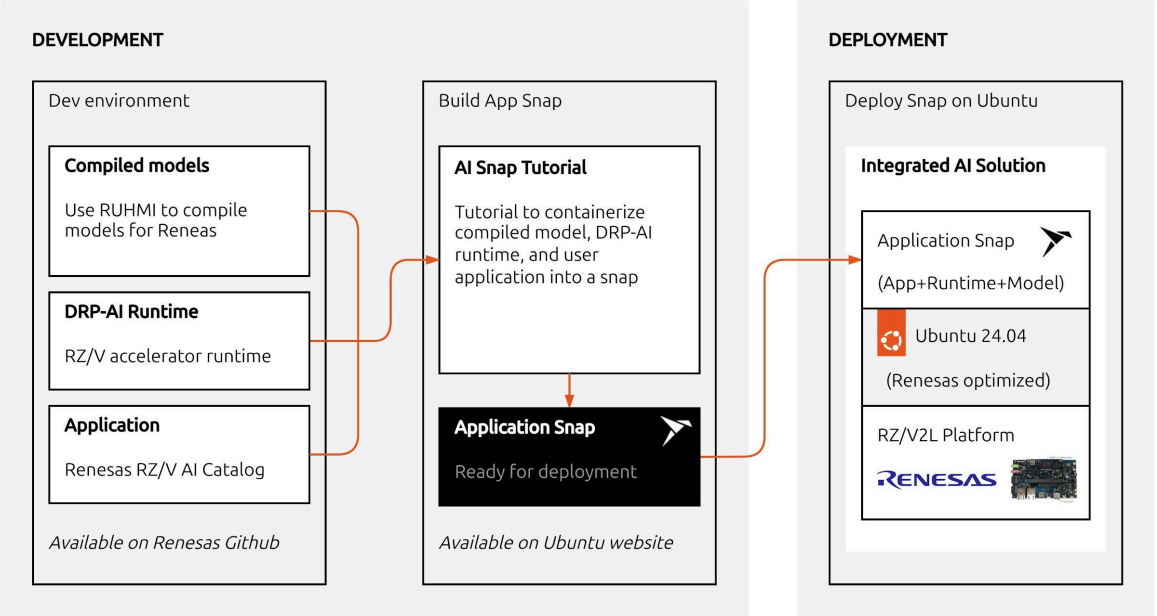
Reviewed on: 9 Jun 2026

Version: 1.2

© Canonical Ltd. All rights reserved.
Confidential and proprietary, do not share without permission.

1. Purpose

This document describes how to build Renesas AI applications using Snapcraft, install the snap on the RZ/V2L board, and run the applications seamlessly.



2. Prerequisites

To get started, you must first set up the RZ/V2L board with the appropriate Ubuntu image. For setup instructions, refer to the Quick Start Guide available at https://ubuntu.com/download/renesas-iot.

In this tutorial we have provisioned the RZ/V2L with an Ubuntu Server 24.04 image.

The demo requires a camera (we used the Omnivision OV5645), an HDMI monitor connected to the RZ/V2L, and an Ethernet cable connected to the RZ/V2L to provide internet access.

3. General Information

The AI application snap is based on the AI applications presented in Renesas RZ/V AI Applications repository: https://github.com/renesas-rz/rzv_ai_sdk.

The snap source used in this guide is in: https://github.com/canonical/rzv_ai_sdk_snap.

For demonstration purposes, we provide two sample snaps:

- One smaller snap that contains a single application with one model.
- One bigger snap that contains three applications, one of them having three different models for different use cases.

This way we demonstrate that we can have different usage scenarios with snaps, either with a snap containing only one application for one specific purpose or a bigger snap with various applications for several purposes.

The smaller snap is called **rzv-ai-sdk-q08** and is named after the Q08_object_counter example in Renesas RZ/V AI Applications repository. To make this snap smaller in size, we have only added the model corresponding to vehicle counting.

The larger snap is called **rzv-ai-sdk-collections** since it is a collection of 3 applications in the same repository: object-tracker, object-counter, and object-detection. Object-counter application takes different arguments: vehicle, coco or animal and depending on the argument, it uses a different model trained for that particular purpose.

The snaps are already published in the Snap Store, therefore we will simply install the snaps directly from the store to the RZ/V2L via network connection. In the last section we will also show how to build snaps locally.

4. Running the AI Snap Demo

Step 1: Install Required Snaps

On the RZ/V2L board, install Ubuntu Frame, which serves as a reliable and secure display server for embedded Linux devices.

```
Shell
$ sudo snap install ubuntu-frame --channel=24/stable
```

Ubuntu Frame automatically installs another snap, mesa-2404. In this demo, we will be using Renesas GPU snap instead. Therefore at this step, we remove the mesa-24 snap:

```
Shell
$ sudo snap remove mesa-2404
```

Next we install the rz-gpu-snap-core24 snap to enable the GPU capabilities (Mali drivers) on the board.

```
Shell
$ sudo snap install --devmode rz-gpu-snap-core24 --channel=beta
```

Then, install the AI SDK collection snap:

```
Shell
$ sudo snap install --devmode rzv-ai-sdk-collection --channel=beta
```

and/or the smaller q08 snap:

```
Shell
$ sudo snap install --devmode rzv-ai-sdk-q08 --channel=beta
```

Step 2: Connect Interfaces

Create a connection between the Ubuntu Frame, the newly installed AI snap(s), and the GPU snap so they can access the necessary hardware capabilities:

```
Shell
$ sudo snap connect ubuntu-frame:gpu-2404 rz-gpu-snap-core24
$ sudo snap connect rzv-ai-sdk-collection:gpu-2404 rz-gpu-snap-core24
$ sudo snap connect rzv-ai-sdk-q08:gpu-2404 rz-gpu-snap-core24
```

Step 3: Run Ubuntu Frame

Run Ubuntu Frame on the device. You will notice the Ubuntu Frame interface displayed on your HDMI-connected monitor.

```
Shell
$ ubuntu-frame &
```

Step 4: Execute the Applications

You can now run the desired AI applications. The command arguments are identical to those defined in the upstream AI applications provided in the `rzv_ai_sdk`.

For the collection snap:

```
Shell
$ rzv-ai-sdk-collection.object-tracker MIPI
$ rzv-ai-sdk-collection.object-counter <COCO|animal|vehicle> MIPI
$ rzv-ai-sdk-collection.object-detection
```

For the q08 snap:

```
Shell
$ rzv-ai-sdk-q08.object-counter vehicle MIPI
```

5. Building Your Own AI Snaps

You can create your own snaps to deploy other applications from the Renesas catalog or custom models. The primary configuration file used to build a snap is `snapcraft.yaml`.

For the AI application snap, refer to: https://github.com/canonical/rzv_ai_sdk_snap.

An AI application on RZ/V2L comprises three principal components:

- **Compiled model:** This is tailored for the RZ/V2L platform. Renesas provides the RUHMI framework for optimization, which typically requires compilation on an x86 host machine running Ubuntu 22.04.
- **TVM Runtime library:** This pre-built library handles communication with the DRP-AI driver. It is recommended to consistently use a unified version of the TVM components.
- **User application:** This integrates all the components to form the final application.

Step 1: Configure snapcraft.yaml

Below are the key code snippets necessary to build your AI applications inside a snap environment.

1. Fetch the TVM Runtime: The `tvm-runtime` component retrieves the runtime libraries and stages the requisite header files needed for application compilation.

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

2. Retrieve the App and Dependencies: The `rz-ai-sdk-src` component retrieves the application source code. Using the `stage-packages` section, you easily incorporate all the necessary Ubuntu archive dependencies required to compile the application.

```
None
  rz-ai-sdk-src:
    after: [tvm-runtime]
    plugin: nil
    source: https://github.com/renesas-rz/rzv_ai_sdk.git
    source-type: git
    source-tag: v6.20
    stage-packages:
      - libmmngr1:$CRAFT_ARCH_BUILD_FOR
      - libmmngrbuf1:$CRAFT_ARCH_BUILD_FOR
      - libopencv-video406t64:$CRAFT_ARCH_BUILD_FOR
      - libopencv-highgui406t64:$CRAFT_ARCH_BUILD_FOR
      - libopencv-imgproc406t64:$CRAFT_ARCH_BUILD_FOR
      - libopencv-imgcodecs406t64:$CRAFT_ARCH_BUILD_FOR
      - libopencv-videoio406t64:$CRAFT_ARCH_BUILD_FOR
      - libopencv-contrib406t64:$CRAFT_ARCH_BUILD_FOR
      - libtesseract5:$CRAFT_ARCH_BUILD_FOR
      - v4l-utils:$CRAFT_ARCH_BUILD_FOR
      - gstreamer1.0-plugins-good:$CRAFT_ARCH_BUILD_FOR
```

> **Note:** Graphics-related libraries are excluded from `stage-packages` because the snap utilizes the GPU interface to take advantage of the Mali drivers.

3. Define the GPU Plug: To access the Mali driver capabilities provided by `rz-gpu-snap-core24`, you must define the `gpu-2404` plug.

```
None
plugs:
```

```
  gpu-2404:
    interface: content
    target: $SNAP/gpu-2404
    default-provider: mesa-2404
```

4. Pull the Compiled Model: If your application requires specific models that are not pre-packaged, you define a part to fetch them. Users can optimize and compile their own models and subsequently replace these files as necessary.

```
None
  object-detection-model:
    after: [r01-object-detection]
    source:
https://github.com/renesas-rz/rzv_ai_sdk/releases/download/v7.00/R01_obj
ect_detection_deploy_tvm_v2l-v261.so
    source-type: file
    plugin: dump
    organize:
      R01_object_detection_deploy_tvm_v2l-v261.so:
usr/r01/bin/yolov3_onnx/deploy.so
```

5. Compile with CMake: The Snapcraft CMake plugin handles the application compilation when provided with the proper build configurations.

```
None
  q01-footfall-counter:
    after: [rz-ai-sdk-src]
    plugin: cmake
    source:
${CRAFT_PROJECT_DIR}/../parts/rz-ai-sdk-src/src/Q01_footfall_counter/src
    source-type: local
    cmake-parameters:
      - -DCMAKE_INSTALL_PREFIX=/usr/q01
      - -DCMAKE_C_COMPILER=aarch64-linux-gnu-gcc
      - -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++
    build-environment:
      - PRODUCT: V2L
      - TVM_HOME: $CRAFT_STAGE/usr/local/tvm/tvm
    override-build: |
      mkdir -p $CRAFT_PART_INSTALL/usr/q01/bin/
```

```
      cp -a
${CRAFT_PROJECT_DIR}/../parts/rz-ai-sdk-src/src/Q01_footfall_counter/exe
_v2l/* $CRAFT_PART_INSTALL/usr/q01/bin/
      craftctl default
```

Step 2: Build the Snap

Once the `snapcraft.yaml` is fully configured, it is built with a single command:

```
Shell
$ snapcraft pack
```

Therefore building the AI SDK snap only requires the following commands to be executed:

```
Shell
$ git clone https://github.com/canonical/rzv_ai_sdk_snap.git
$ cd rzv_ai_sdk_snap
$ snapcraft pack
```

This builds a .snap file locally that is ready to be deployed to your RZ/V2L device. Refer to README of the repository for additional information.

General information about crafting snaps and environment setup can be found at https://documentation.ubuntu.com/snapcraft/stable/tutorials/craft-a-snap/

Step 3: Publish the Snap

To streamline distribution, you can upload your completed snap to the Snap Store. This allows users to discover and install your AI application using a single command. More information on registering and publishing snaps can be found here: https://documentation.ubuntu.com/snapcraft/latest/how-to/publishing/