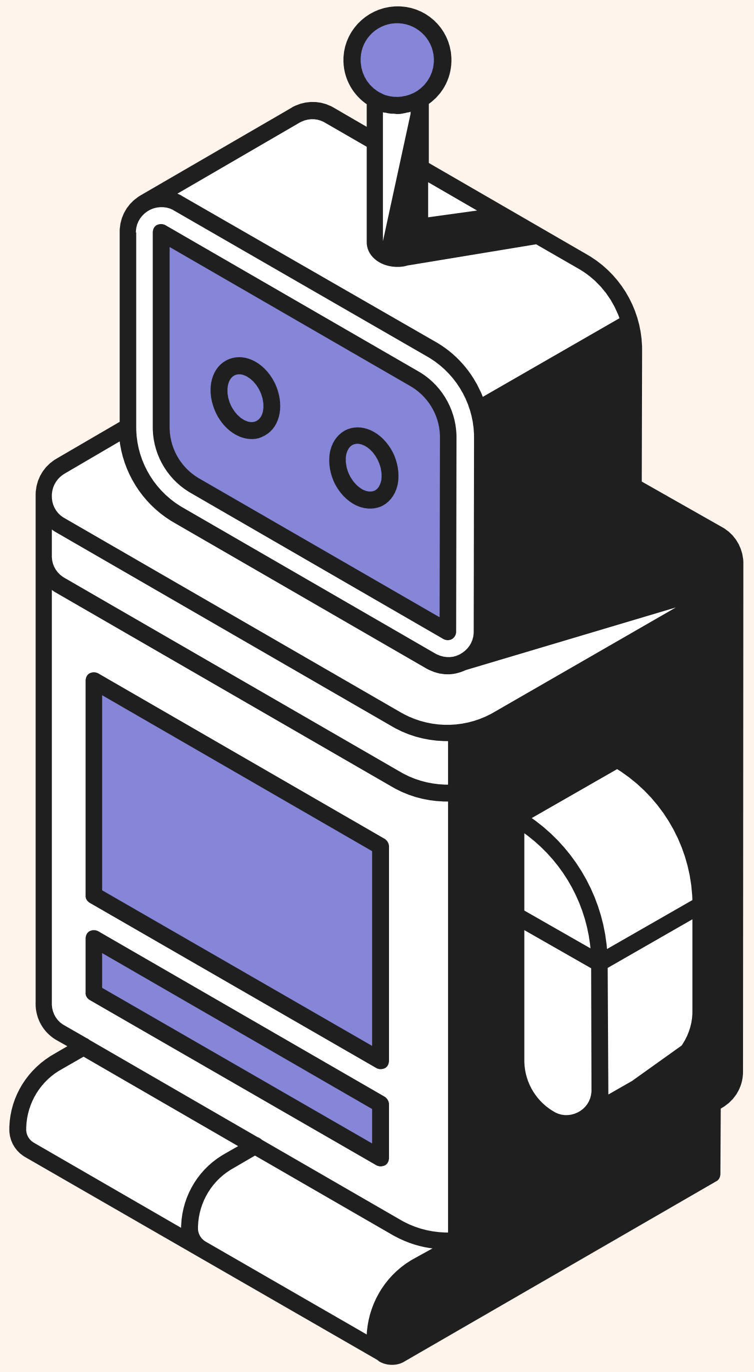
Real Time Systems and Industrial Applications Project

ANALYSIS OF AI WORKLOADS

ON *NVIDIA JETSON NANO*

Authors: Antonio Boccarossa, Francesco Brunello
Institution: Università degli studi di Napoli Federico II



WHY EDGE AI?

Artificial Intelligence is transforming many industries, but the traditional model of relying solely on **cloud-based processing** is facing limitations, particularly in applications **demanding real-time performance**.
This has led to the emergence of ***Edge AI*** – **a paradigm where AI algorithms are executed directly on devices** at the edge of the network, rather than in a centralized cloud.

WHAT DOES ‘EDGE' MEAN?

It refers to devices like smartphones, embedded systems, industrial sensors, and, as in our case, platforms like the **Nvidia Jetson Nano**.

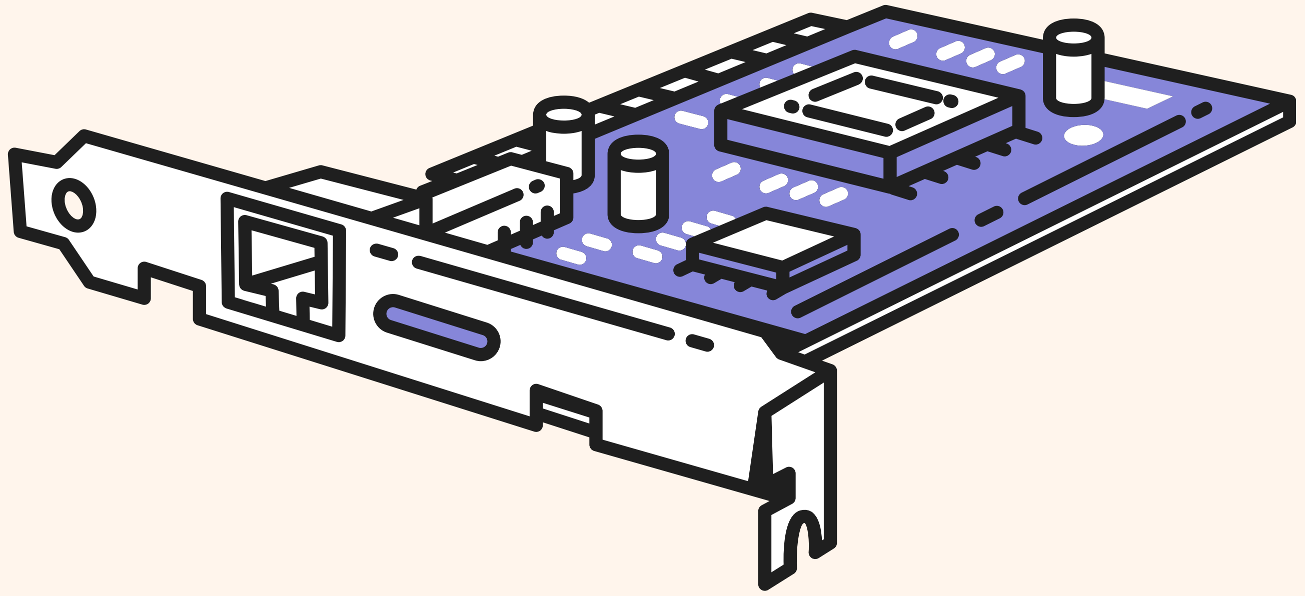
The benefits of Edge AI are significant. First and foremost is *reduced latency*.
By processing data locally, we eliminate the round-trip time to the cloud, enabling near-instantaneous responses. This **is critical for applications like autonomous driving, where milliseconds matter**.

BENEFITS OF EDGE COMPUTING

Beyond latency, Edge AI offers increased *privacy* – sensitive data doesn't need to be transmitted.
It also improves *reliability* – **operation isn't dependent on a constant cloud connection.** And it **reduces *bandwidth*** *costs* by processing data locally and only sending relevant insights to the cloud.

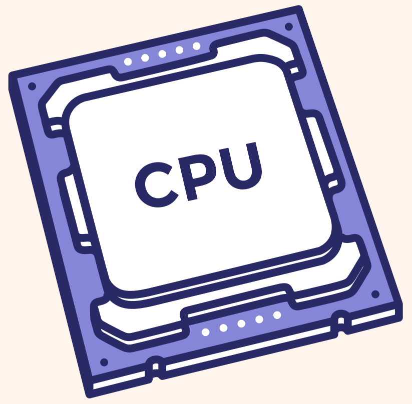
PROJECT OVERVIEW

The main purpose of the project is to analyze the inference time performances of **some well-known models** on **Nvidia Jetson Nano**. The models considered are **Inception v1, Inception v4, v2, SSD MobileNet v1, SSD MobileNet v2**. We'll analyze the inference times in a **"golden-run"** environment as baseline, and repeating those tests in presence of stressors injected with **stress-ng** tool and some other tests.



NVIDIA JESTON NANO

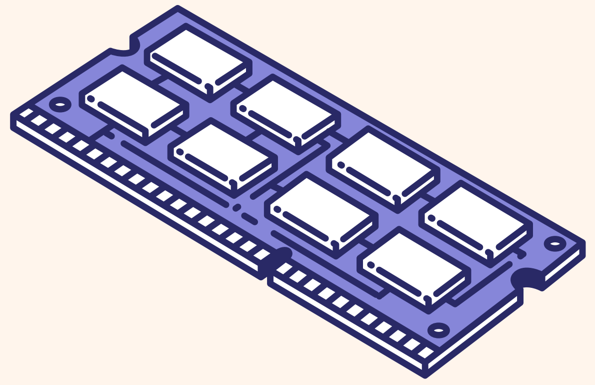
HARDWARE COMPONENTS



PROCESSING UNITS

Quad-core ARM Cortex-A57 MPCore processor

NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores



MEMORY (RAM & STORAGE)

4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s

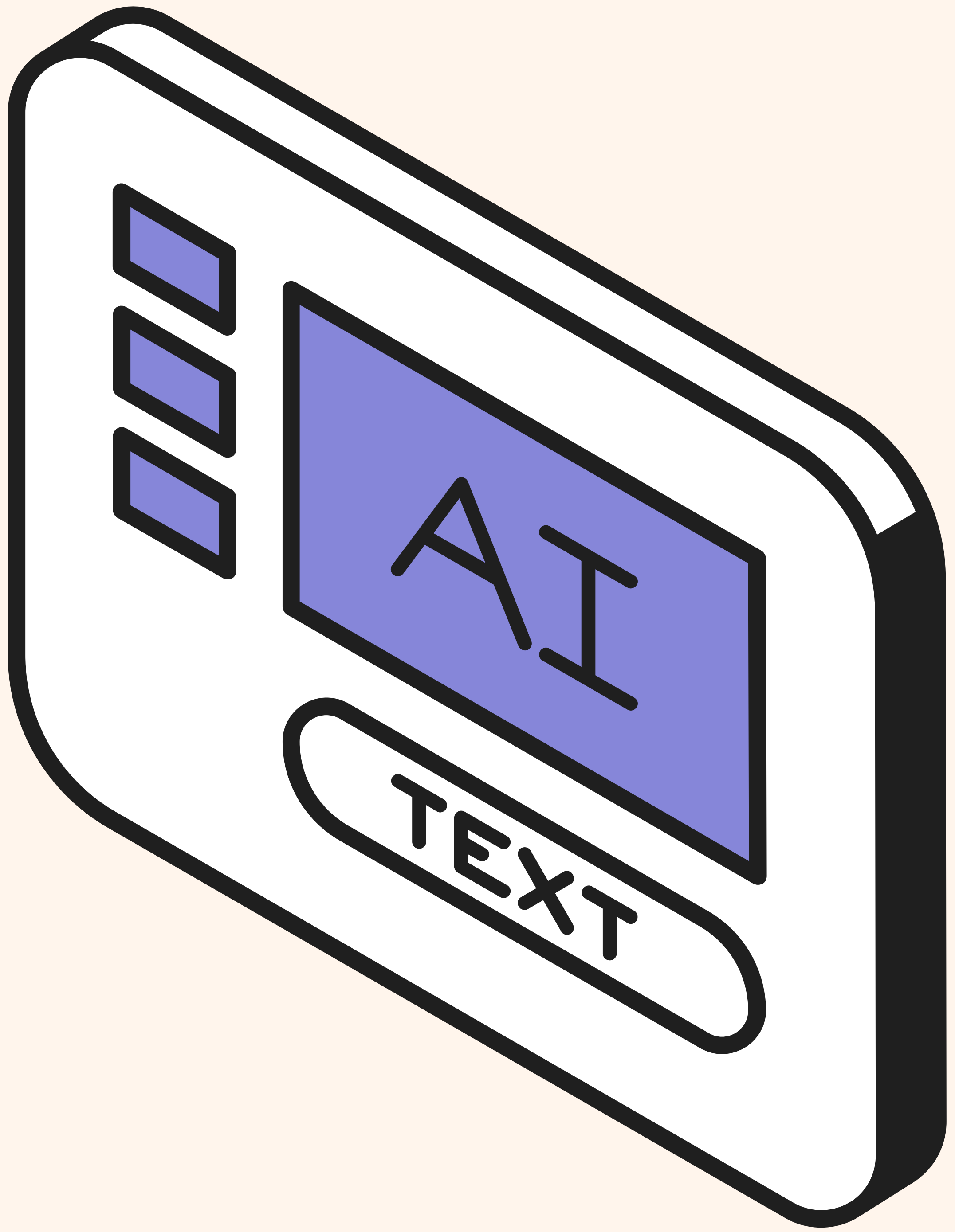
16 GB eMMC 5.1

MODELS ANALYZED

According to the suite of models provided by the "JetPack" SDK, these are the models which we used to make inference:

- **Inception-V1/V4**: image classification,
- **SSD-MobileNet-V1/V2**: object detection
- **MobileNet-V1/V2**: image detection

P.S.: MobileNet-V1 and MobileNet-V2 are not included in the tests due to incompatibility problems (packets missing due to old versions of Python and Libraries).



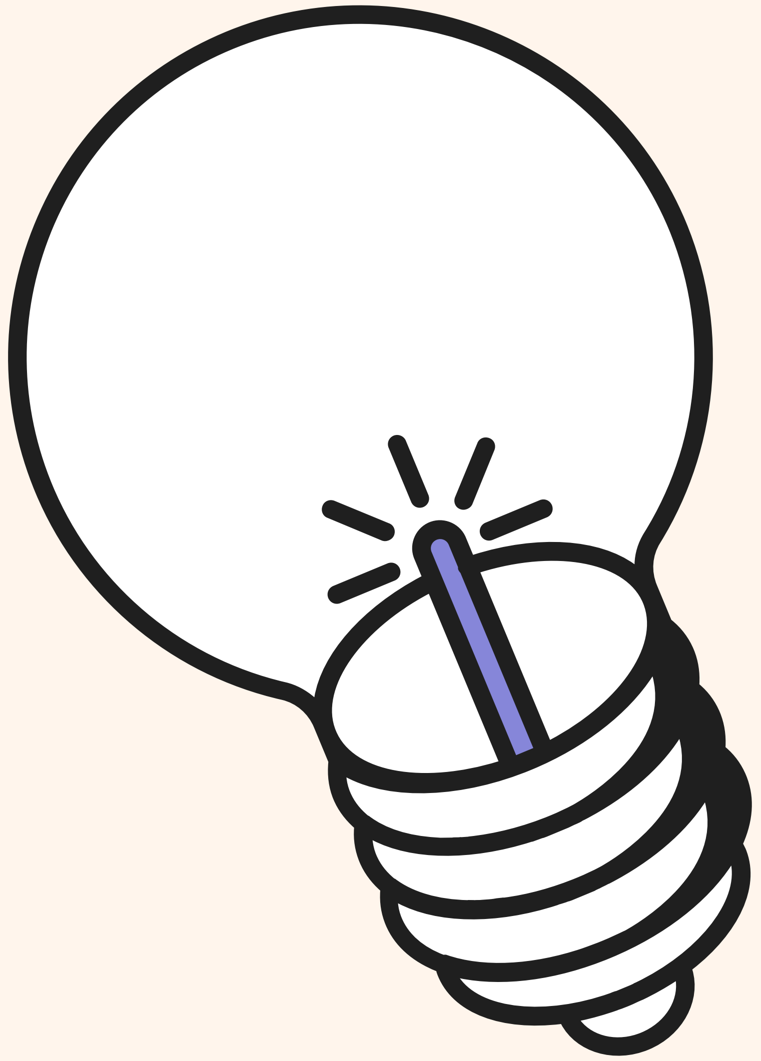
JETSON NANO

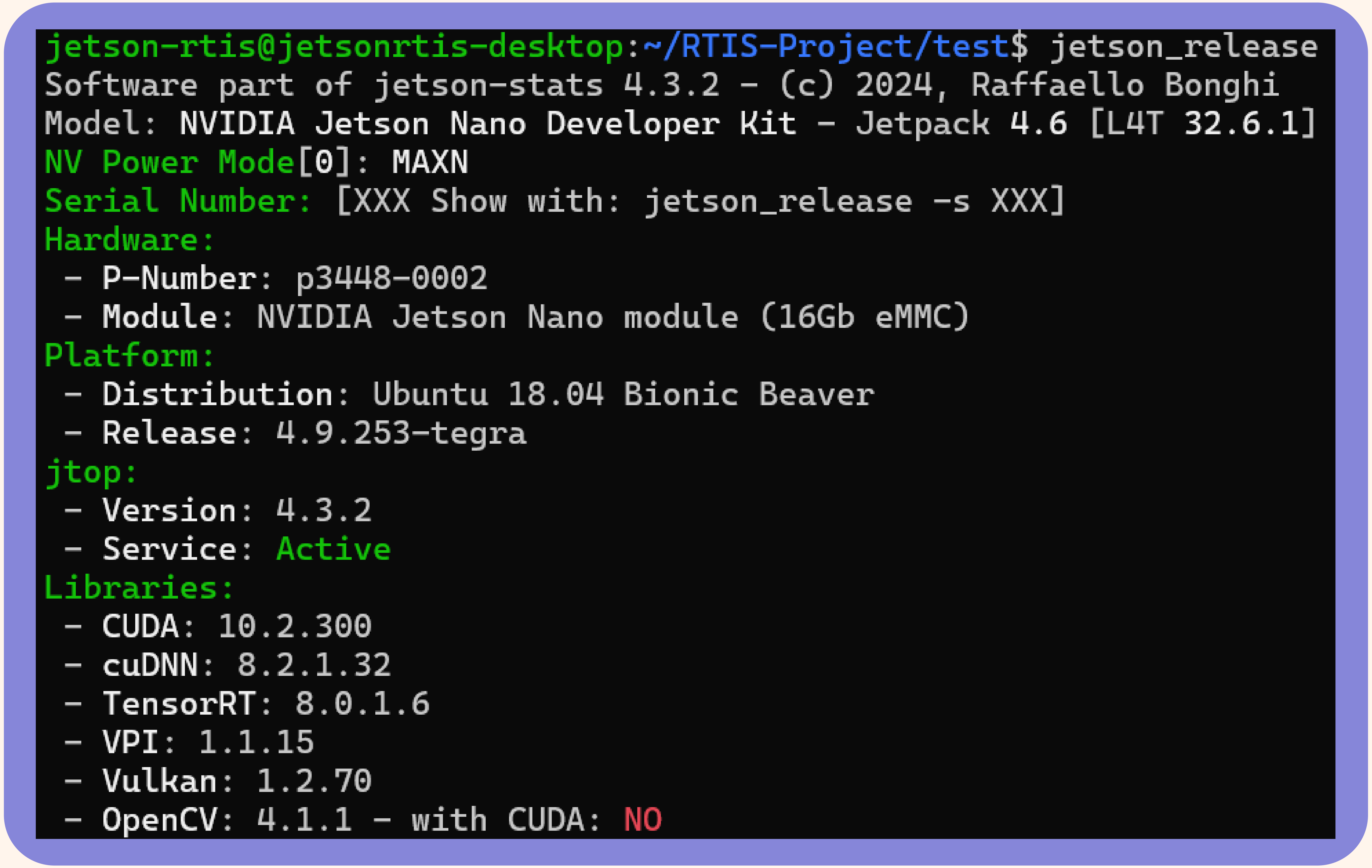
SETUP

In order to make inference tests with these models, we used a open-source project devloped by an Nvidia Developer (Jetson Inference on GitHub).

In a first appraoch we tried to use the *containerized* verison of the project, but due some incompatibilities we preferred to build the project from source (as shown on the repository).

After this, we added to the "**LD_LIBRARY_PATH**", which is the path were the **shared libraries** were compiled *(/usr/local/lib)*, in order to make our script work.





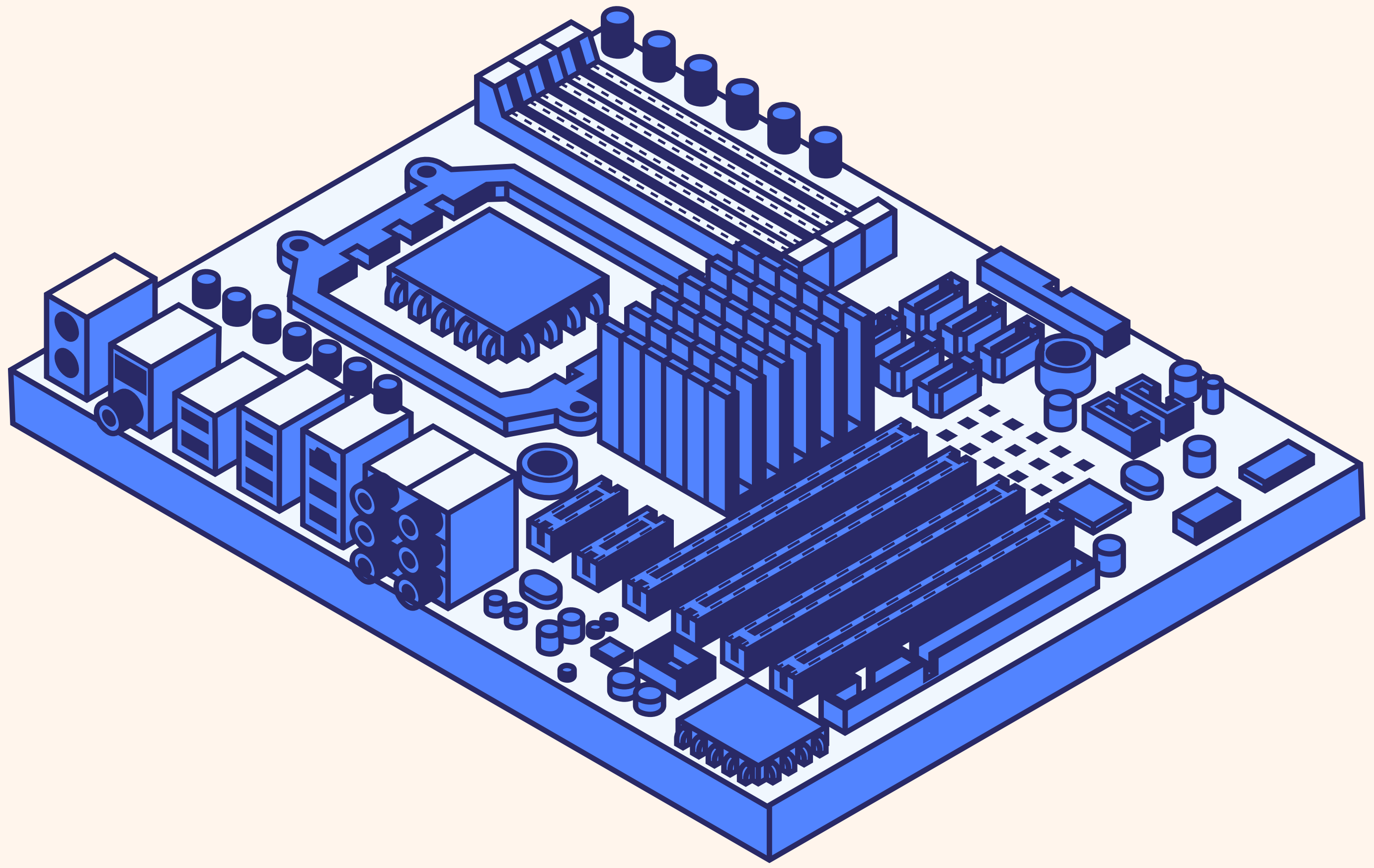
```
jetson-rtis@jetsonrtis-desktop:~/RTIS-Project/test$ jetson_release
Software part of jetson-stats 4.3.2 - (c) 2024, Raffaello Bonghi
Model: NVIDIA Jetson Nano Developer Kit - Jetpack 4.6 [L4T 32.6.1]
NV Power Mode[0]: MAXN
Serial Number: [XXX Show with: jetson_release -s XXX]
Hardware:
 - P-Number: p3448-0002
 - Module: NVIDIA Jetson Nano module (16Gb eMMC)
Platform:
 - Distribution: Ubuntu 18.04 Bionic Beaver
 - Release: 4.9.253-tegra
jtop:
 - Version: 4.3.2
 - Service: Active
Libraries:
 - CUDA: 10.2.300
 - cuDNN: 8.2.1.32
 - TensorRT: 8.0.1.6
 - VPI: 1.1.15
 - Vulkan: 1.2.70
 - OpenCV: 4.1.1 - with CUDA: NO
```

WHAT IS L4T ?

Linux for Tegra (L4T) is a custom Linux distribution developed by NVIDIA specifically for its *Jetson* platforms.

It is based on a **customized** Linux kernel with **NVIDIA-specific** drivers and patches for **ARM64** architecture.

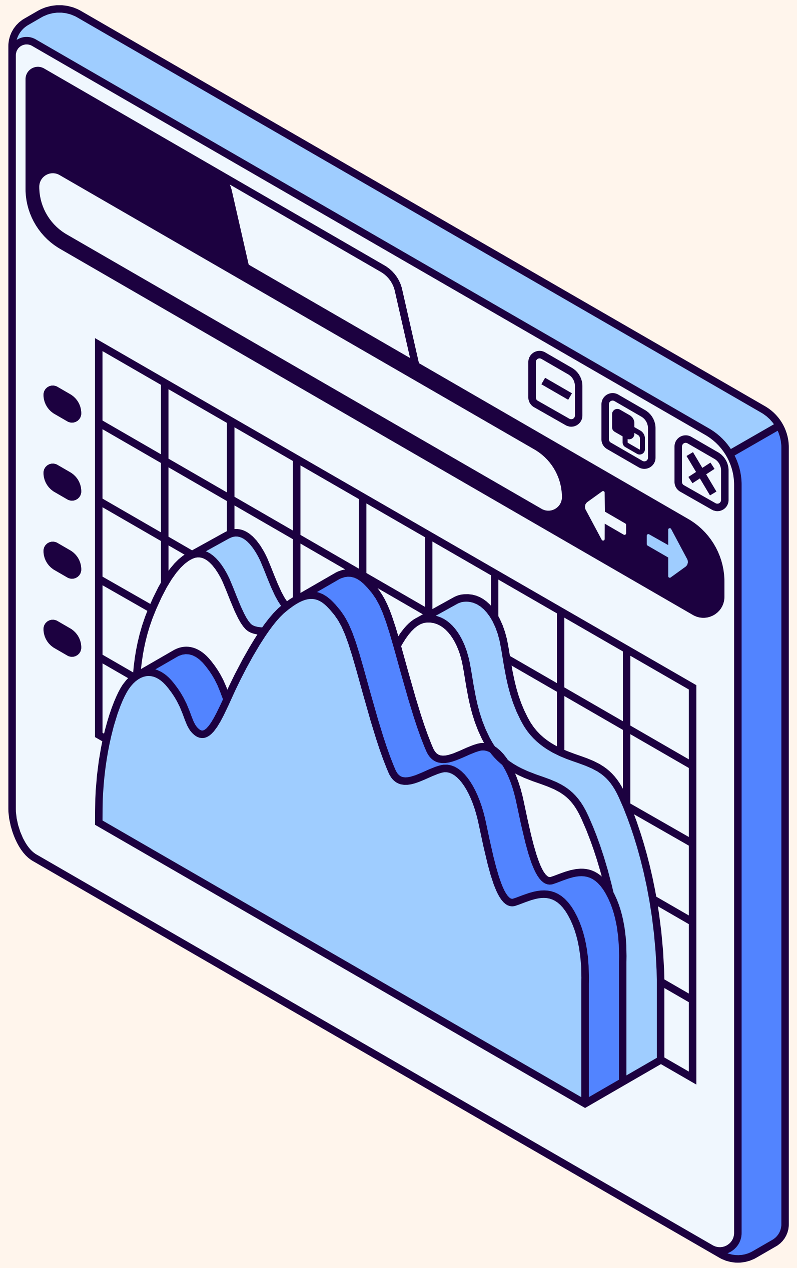
- Supports **CUDA, TensorRT, cuDNN,** and other AI frameworks.
- Includes NVIDIA GPU drivers and V4L2 camera support (for object detection purposes).
- Optimized for low power and real-time embedded applications.

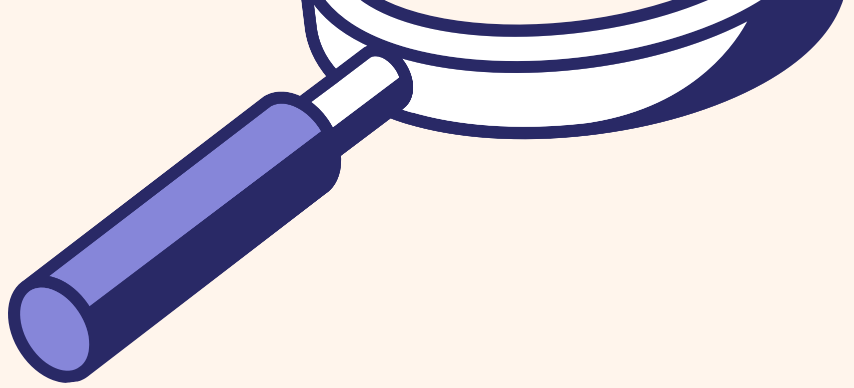


TESTS

OVERWIEW

- A single script makes inference by using the ***specified model on a 108 images dataset*** (with varying dimensions, major than 400x400). At the end, ***the script will append the average of all the inference times in a dedicated file*** (based on the specific test and model).
- The previously mentioned script is executed 30 times, in order to have for each specific test, a sample size of 30.

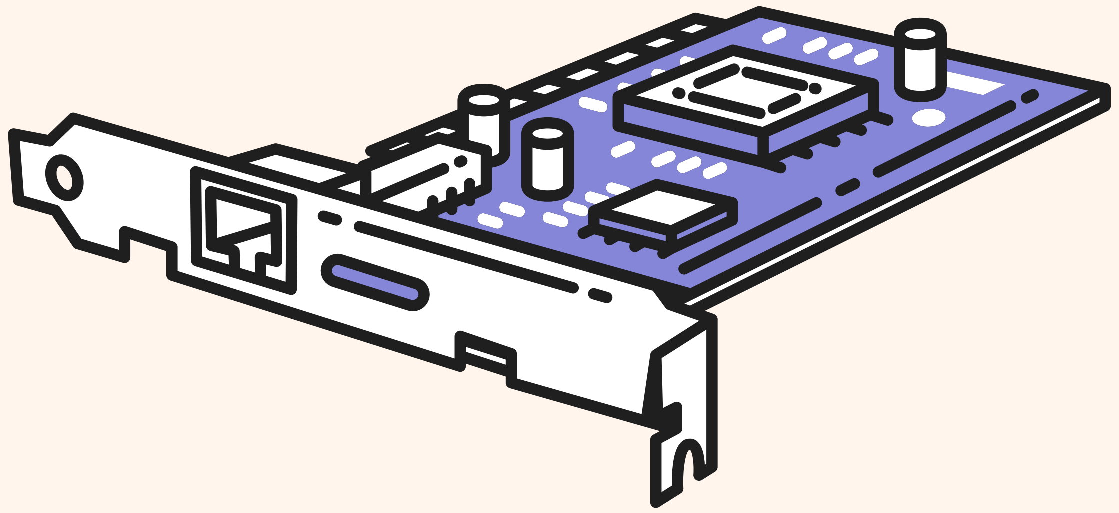




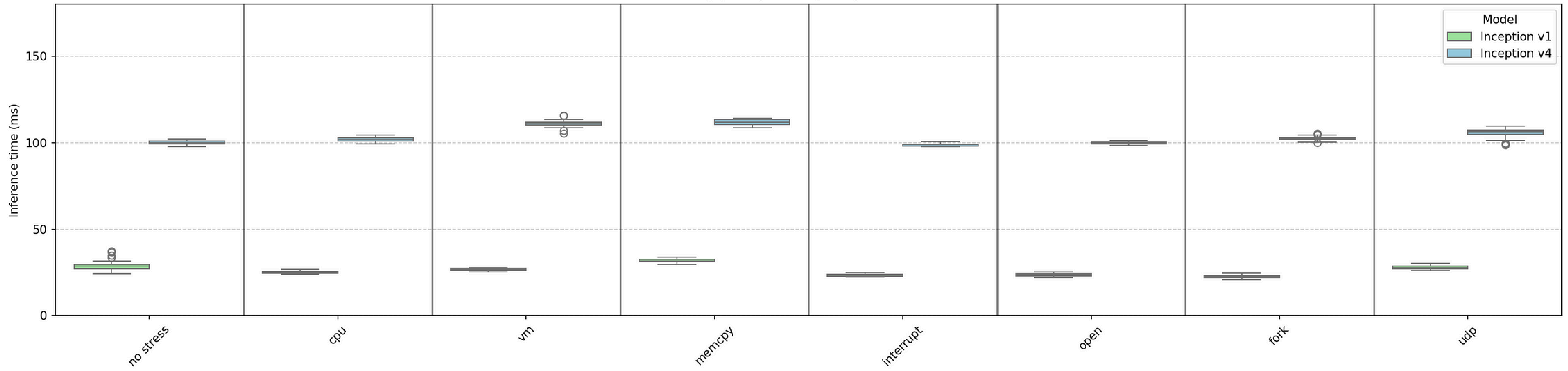
STRESS TEST
WITH STRESS-NG

In order to evaluate the performance of the board under load, we applied several stressors to our Jetson Nano:

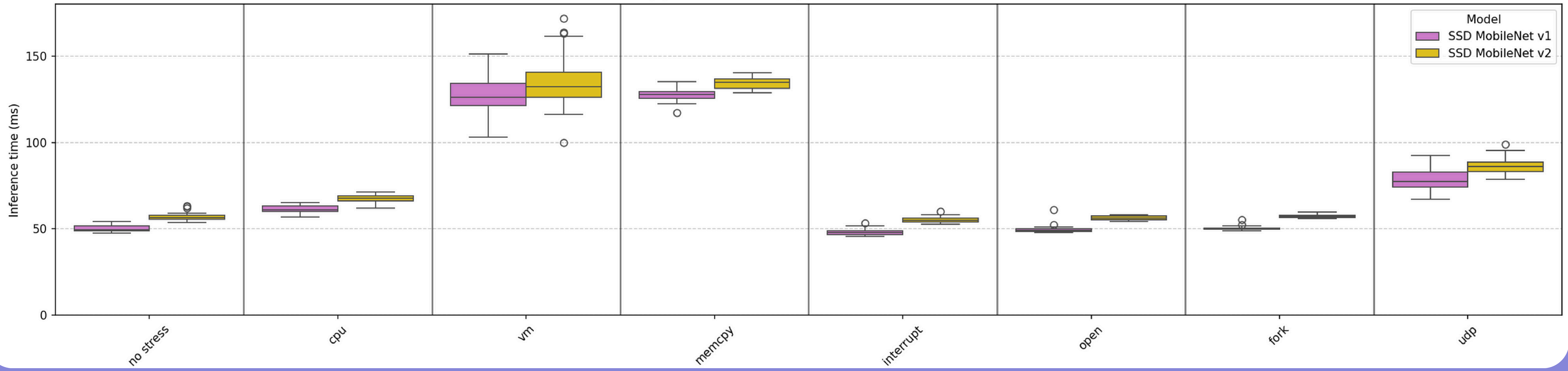
- *cpu – **stress-ng --cpu 4***
- *virtual memory – **stress-ng --vm 4 --vm-bytes 2.5G***
- *memcpy – **stress-ng --memcpy 8***
- *interrupt – **stress-ng --clock 4 --aio 4 --aio-requests 30***
- *open – **stress-ng --open 4***
- *fork – **stress-ng --fork 4***
- *udp – **stress-ng --udp 4***



(a) Inception v1, Inception v4



(b) SSD MobileNet v1, SSD MobileNet v2 tests

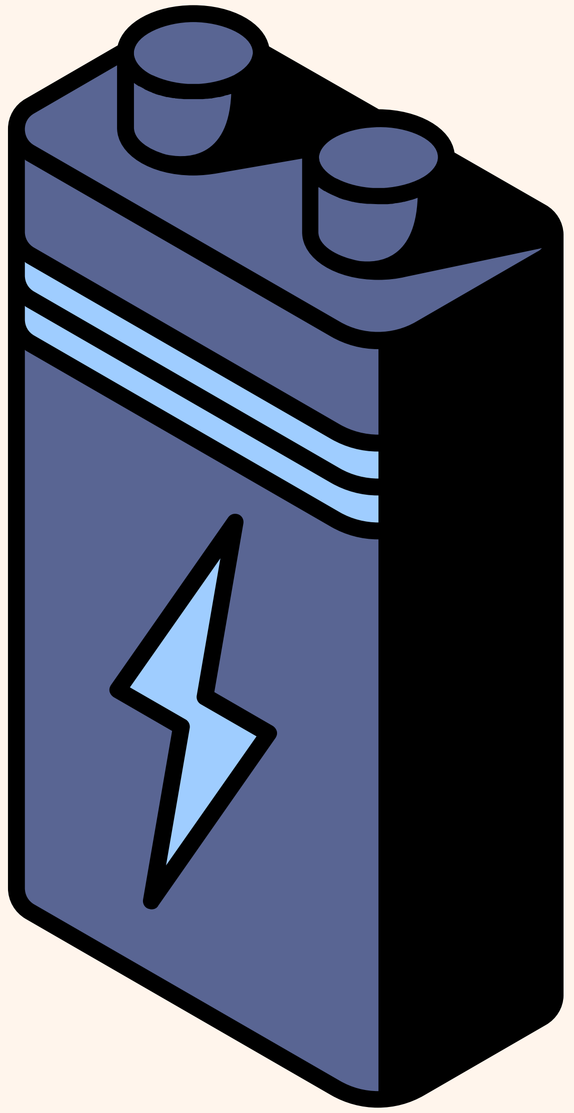


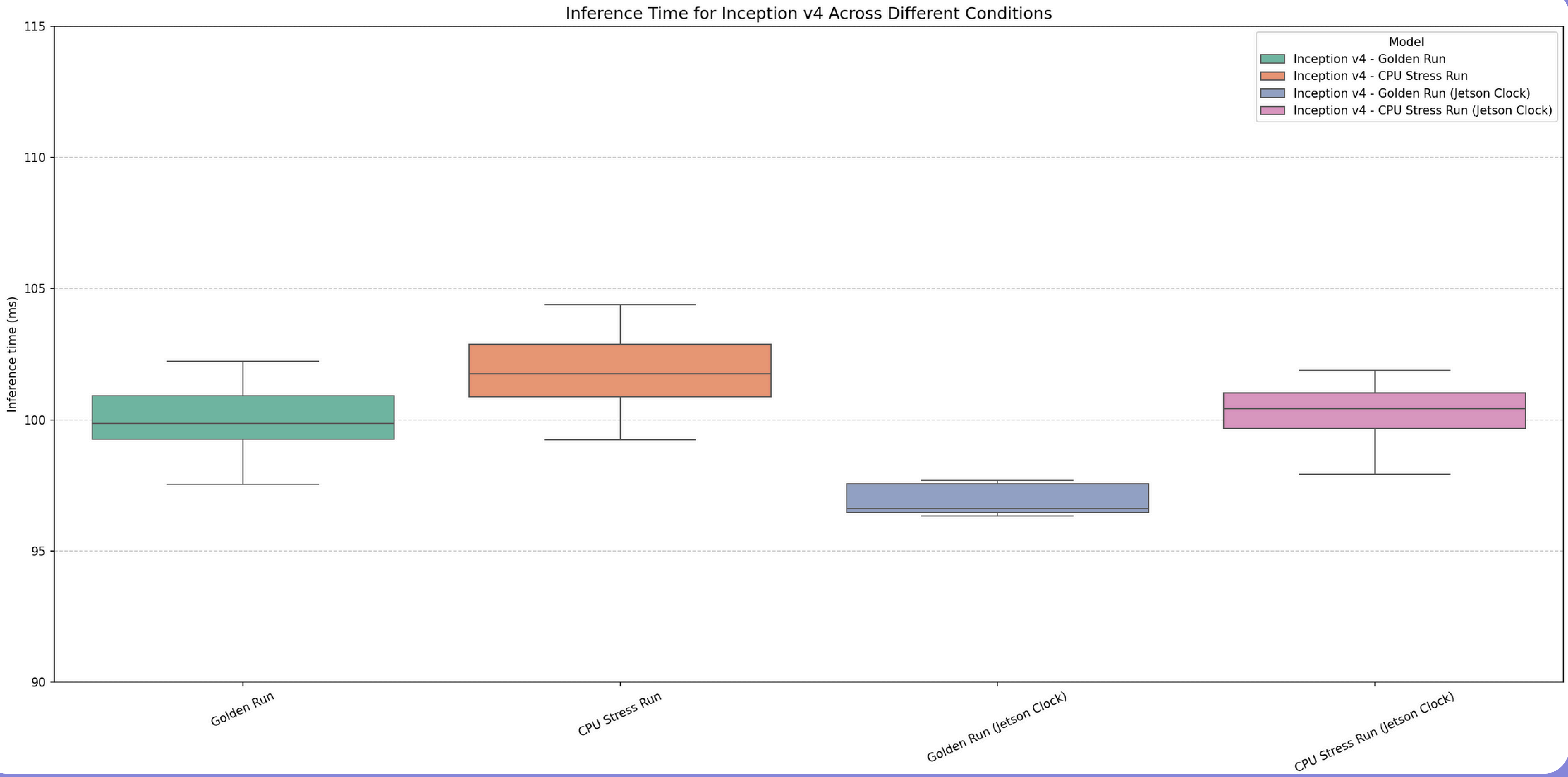
STRESS TEST
WITH JETSON_CLOCKS

In order to improve the overall performance while stressing the system, it’s useful to enable **jetson_clocks**:

- jetson_clocks is a tool provided by all **NVIDIA Jetson** to maximize performance.

When enabling **jetson_clocks**, the clock frequency is set to the maximum value (*1,479 GHz*) for each core: this results in a significant improvement for the inference times.

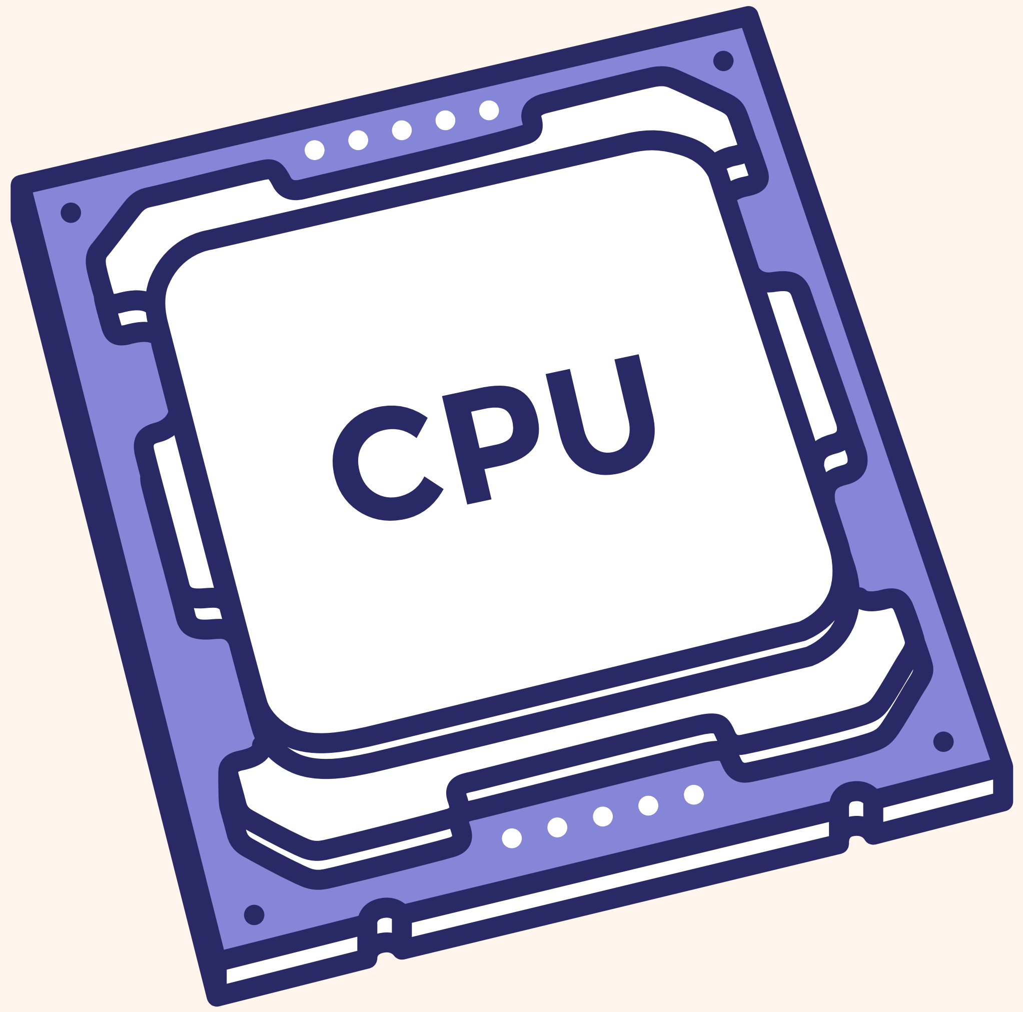


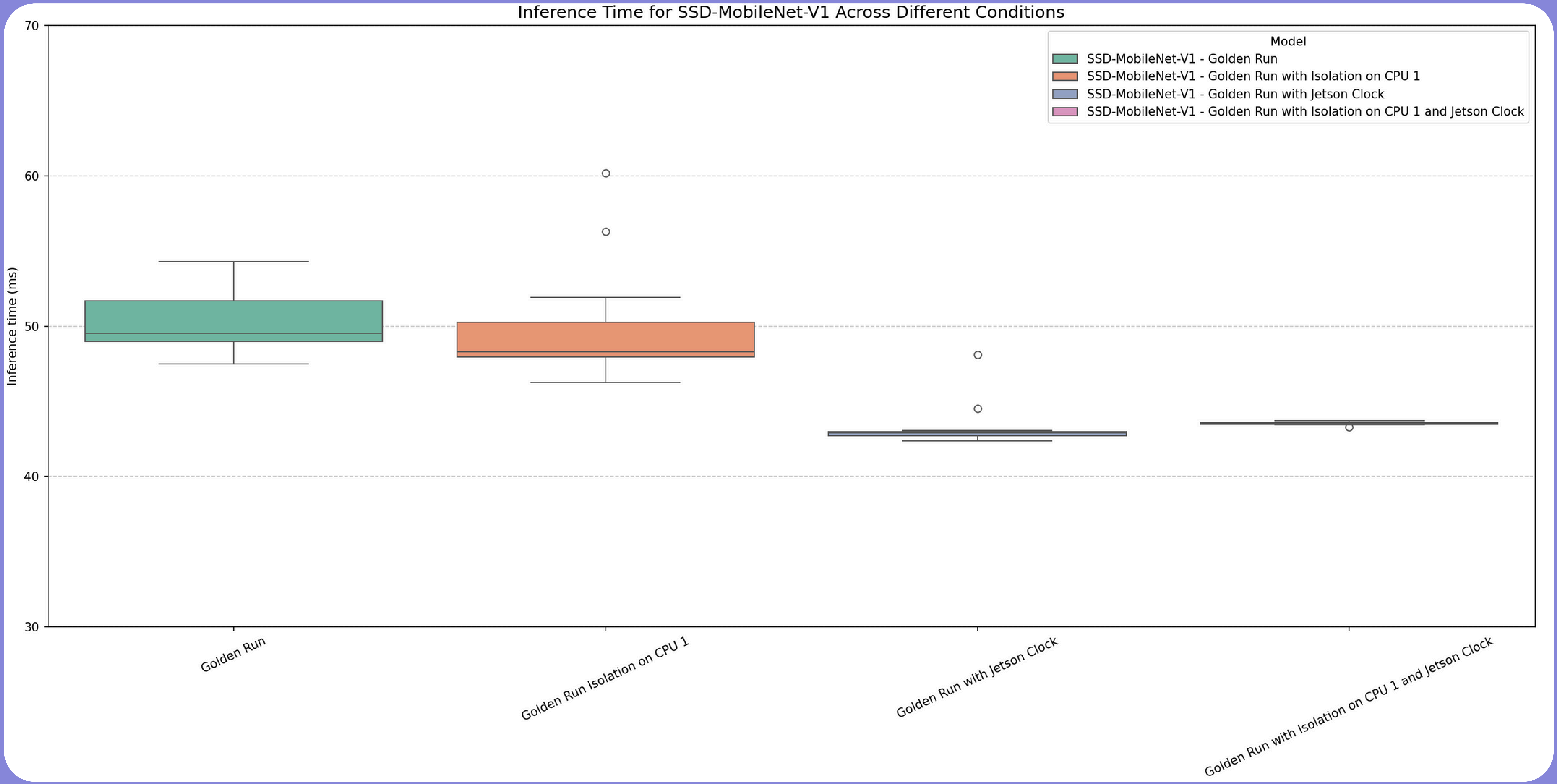

Inference Time for Inception v4 Across Different Conditions
Inference time (ms)
115
110
105
100
95
90
Model
Inception v4 - Golden Run
Inception v4 - CPU Stress Run
Inception v4 - Golden Run (Jetson Clock)
Inception v4 - CPU Stress Run (Jetson Clock)
Golden Run
CPU Stress Run
Golden Run (Jetson Clock)
CPU Stress Run (Jetson Clock)

STRESS TEST
WITH CPU ISOLATION

To further investigate about the inference times of our models, we needed to isolate those tasks on a specific CPU core: this was done by editing the configuration file of the kernel with a specification about how many cores we wanted to isolate from the kernel scheduler:

- Following the **/boot/extlinux/** path, there's a file named **extlinux.conf**
- Within this file, we needed to add *isolcpus=1* and *rcu_nocbs=1* to the **APPEND** line



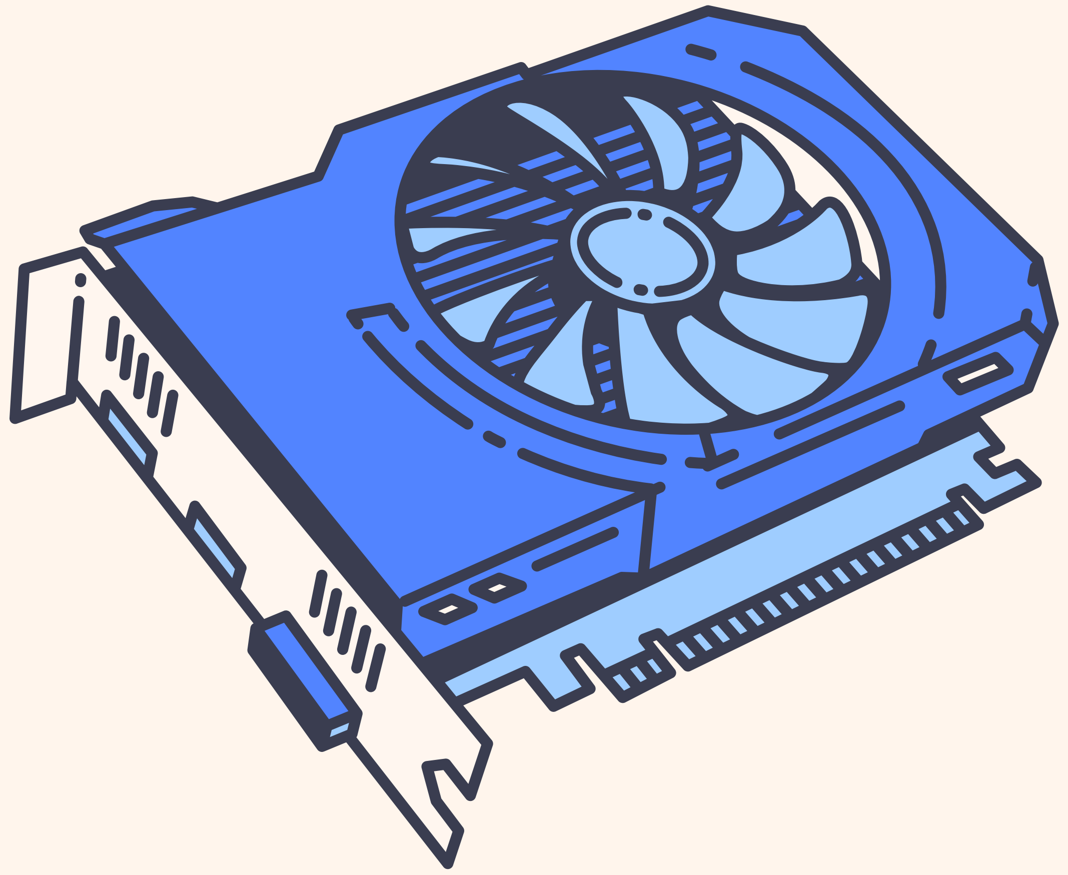

Inference Time for SSD-MobileNet-V1 Across Different Conditions
Model
SSD-MobileNet-V1 - Golden Run
SSD-MobileNet-V1 - Golden Run with Isolation on CPU 1
SSD-MobileNet-V1 - Golden Run with Jetson Clock
SSD-MobileNet-V1 - Golden Run with Isolation on CPU 1 and Jetson Clock
Inference time (ms)
70
60
50
40
30
Golden Run
Golden Run Isolation on CPU 1
Golden Run with Jetson Clock
Golden Run with Isolation on CPU 1 and Jetson Clock

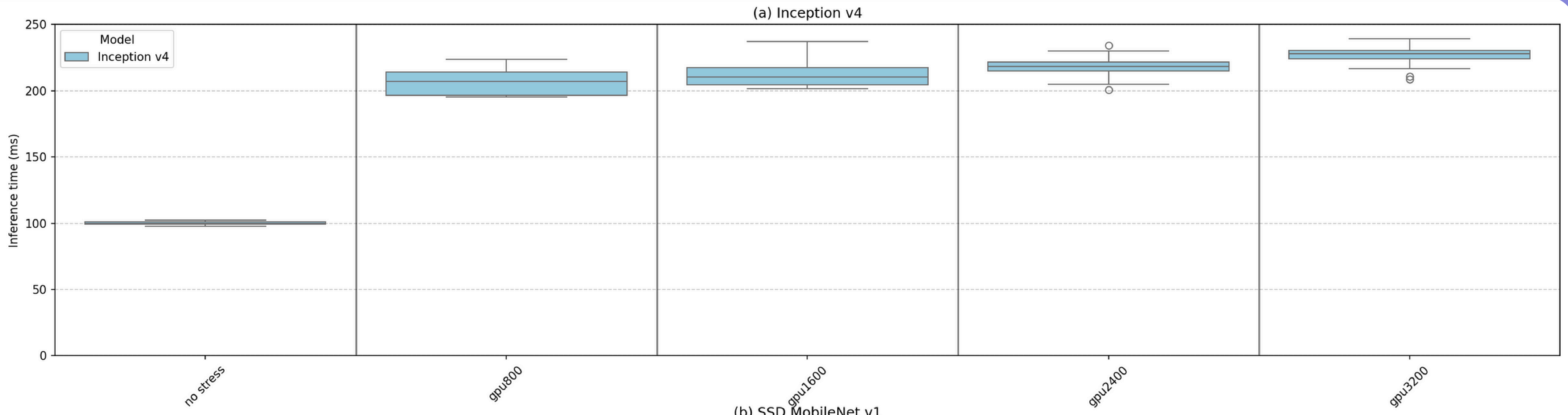
STRESS TEST
WITH MATRIXMUL

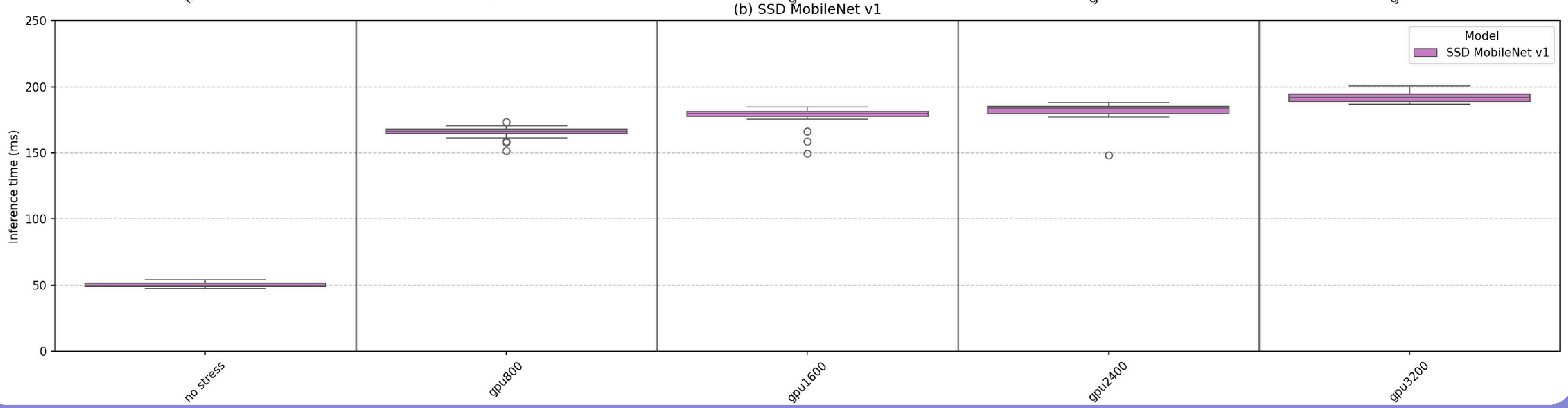
For this kind of test, we wanted to do stress the NVIDIA GPU within our component.
The tests executed are made with four different matrix dimensions:

- ***800x800 * 800x800*** (29.5Mb of GPU memory occupied)
- ***1600x1600 * 1600x1600*** (51.4Mb of GPU memory occupied)
- ***2400x2400 * 2400x2400*** (88Mb of GPU memory occupied)
- ***3200x3200 * 3200x3200*** (139Mb of GPU memory occupied)

Our goal was to stress the inference tasks while performing matrix multiplications. With higher values than the previously presented, **matrixMul** had some execution problems due to matrix dimensions.




(a) Inception v4
Model
Inception v4
Inference time (ms)
250
200
150
100
50
0
no stress
gpu800
gpu1600
gpu2400
gpu3200


(b) SSD MobileNet v1
Model
SSD MobileNet v1
Inference time (ms)
250
200
150
100
50
0
no stress
gpu800
gpu1600
gpu2400
gpu3200

PROBLEMS ENCOUNTERED

In order to execute these tests, we encountered some problems on the way:

1. Due to missing packets, as **Torch for Python 3.6.9, the tests about MobileNet-V1 and MobileNet-V2** were not possible:
   a. Trying to install the packages manually from the *.whl files*, but seems there is not a torch version for arm architectures that are CUDA 10.2 enabled,
   b. Trying to look for a newer version of the JetPack SDK respect to the pre-installed version 4.6, we found that the last version supported by the Jetson Nano is 4.6.3, that brings some little security patches to the kernel and not some brand new version of python, CUDA, cuDNN and TensorRT.
2. Due to **MatrixMul operations** crashes, caused by failing operation, with matrix dimensions greater or equal than 4000 and lower or equal than 50. (For example **1000x4000 * 4000x1000** and **50x50**)

THANK YOU