

Analysis of AI Workloads on Nvidia Jetson Nano

18.07.2025

SUBMITTED TO

Università degli studi di Napoli Federico II
Ingegneria Informatica

Real Time Systems and Industrial Applications course

Marcello Cinque, Andrea Marchetta

SUBMITTED BY

Antonio Boccarossa, Francesco Brunello

a.boccarossa@studenti.unina.it, f.brunello@studenti.unina.it

Indice

1. Introduction	1
2. Device exploration	1
3. Analysis of the models	2
3.1. Inception-V1	2
3.2. Inception-V4	2
3.3. SSD-MobileNet-V1 and SSD-MobileNet-V2	3
3.4. MobileNet-V1 and MobileNet-V2	3
4. Environment setting and tests	3
4.1. Stress Tests with <i>stress-ng</i>	3
4.1.1. Stressors	4
4.1.2. Inception-v1 and Inception-v4	4
4.1.3. SSD-MobileNet-v1 and SSD-MobileNet-v2	6
4.2. Golden run and CPU run tests with <i>jetson_clocks</i>	7
4.3. Golden run and CPU run tests with CPU isolation (<i>isol_cpu</i>)	7
4.3.1. Configuration	7
4.4. Stress tests with GPU stressors	8
4.4.1. MatrixMul	9
5. Results	10
5.1. Stress tests with <i>stress_ng</i>	10
5.2. Golden run and CPU run tests with <i>jetson_clocks</i>	10
5.3. Golden run and CPU run tests with CPU isolation (<i>isol_cpu</i> , <i>rcu_nocbs</i>)	11
5.4. Stress tests with GPU stressors	12

1. Introduction

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, MobileNet-V1, MobileNet-V2**. It will be analyzed the inference times in a **«golden-run»** environment as baseline, and repeating those tests in presence of stressors injected with **stress-ng** tool:

- *cpu*
- *virtual memory*
- *mem copy*
- *interrupt*
- *open*
- *fork*
- *udp*

These tests will be executed both with normal and maximum performance of the cores. The maximum performances are reached through the «jetson_clock» tools provided with the Jetson JetPack SDK v4.6 already installed on the machine.

2. Device exploration

The NVIDIA Jetson Nano is a small AI Computer designed to handle modern AI workloads and run multiple neural networks in parallel. In order to have enough space to store the images needed for classification/detection purposes, we had to flash the Jetson Nano module from an external SSD (<https://www.youtube.com/watch?v=53rRMrl1pWs>). **Note:** the JetPack SDK (which is a package offered by NVIDIA itself containing the Board Support Package, Linux installation, bootloader, and already pre-installed software stacks for various purposes) was already pre-installed when the first boot occurs. The following table describes the principal features of the Jetson Nano module:

Characteristics	J-10-1-H0 Specifics
GPU	NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores
CPU	Quad-core ARM Cortex-A57 MPCore processor
Memory	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Storage	16 GB eMMC 5.1
Video Encode	250MP/sec; 1x 4K @ 30 (HEVC); 2x 1080p @ 60 (HEVC) 4x 1080p @ 30 (HEVC); 4x 720p @ 60 (HEVC); 9x 720p @ 30 (HEVC)
Video Decode	500MP/sec; 1x 4K @ 60 (HEVC); 2x 4K @ 30 (HEVC) 4x 1080p @ 60 (HEVC); 8x 1080p @ 30 (HEVC); 9x 720p @ 60 (HEVC)
Camera	12 lanes (3x4 or 4x2) MIPI CSI-2 D-PHY 1.1 (1.5 Gb/s per pair)
Connectivity	M.2 Key M, 10/100/1000 BASE-T Ethernet
Display	HDMI 2.0 and eDP 1.4

Characteristics	J-10-1-H0 Specifics
I/O	1x SDIO / 2x SPI / 4x I2C / 2x I2S / GPIOs -> I2C, I2S
UPHY	1 x1/2/4 PCIE, 1x USB 3.0, 3x USB 2.0
Size	69.6 mm x 45 mm
Mechanical	260-pin edge connector

Now, in order to check if the libraries **cuDNN**, **CUDA**, **TensorRT** are provided within the SDK, the following command needs to be executed:

`jetson_release`

```
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
```

Figura 1: Output after executing `jetson_release`

3. Analysis of the models

This chapter talks about the set of models which are compliant with the Jetson Nano module:

3.1. Inception-V1

- Inception v1, also known as GoogLeNet, is a convolutional neural network introduced by Google in 2014 in the paper «Going Deeper with Convolutions» (<https://arxiv.org/pdf/1409.4842>). It was designed to improve both the depth and efficiency of deep networks by reducing the number of parameters while maintaining strong representational power.

3.2. Inception-V4

- Inception-V4 was introduced in combination with *Inception-ResNet* by the researchers at Google in 2016 (<https://arxiv.org/abs/1602.07261>). This model uses blocks called *Inception modules*, which combine convolutions of different sizes

in parallel in order to capture features at multiple scales and is deeper and more complex than the previous version *Inception-V3*.

3.3. SSD-MobileNet-V1 and SSD-MobileNet-V2

- **Single Shot Multibox Detector** is an algorithm which takes only one shot to detect many objects in the image using multibox. SSD uses an auxiliary network for feature extraction, called as *base-network*. These models, basically, are SSD with **MobileNet-V1** and **MobileNet-V2** as features extractors. (<https://iq.opengenus.org/ssd-mobilenet-v1-architecture/>)

3.4. MobileNet-V1 and MobileNet-V2

For what concerns those two models, there were some problems with the SDK version, especially with its Python release (3.6.9). Specifically, there was a problem with the installation of Python *transformers* module and its compilation dependencies, even with the latest versions of the SDK (because the Torch version is the same as the one present in the pre-installed JetPack SDK, in 4.6 flavour).

4. Environment setting and tests

In this section the setting of the environment will be described, in order to make do the previously specified models and their associated tests.

4.1. Stress Tests with *stress-ng*

In order to make inference tests with these models, an open-source project developed by an Nvidia Developer was used (<https://github.com/dusty-nv/jetson-inference/tree/master>). As a first approach, a *containerized* version of the project has been tested, but due to some incompatibilities, it was preferred to build the project from source.

After this, the `LD_LIBRARY_PATH` has been added: this is the path where the **shared libraries** were installed ("`/usr/local/lib`") in order to operate our scripts.

As previously mentioned, the tests have been performed either with stress and without stress and injected with the **stress-ng** tool.

These tests are structured in this way:

- 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 thirty times, in order to have for each specific test, a sample size of 30.

In the next subchapters the scripts used to perform those tests will be described.

4.1.1. Stressors

The tests are repeated identically for all the models, in such environments:

- **Golden Run (baseline):** without stressors,
- **Cpu Run:** with *cpu* stressors,
 - `stress-ng --cpu 4`
- **Vm Run:** with virtual *memory* stressors,
 - `stress-ng --vm 4 --vm-bytes 2.5G`
- **Mem Copy Run:** with *memcpy* stressors (for processor cache and central memory),
 - `stress-ng --memcpy 8`
- **Interrupt Run:** with *clock* and *aio* stressors in order to generate context switches and interrupts,
 - `stress-ng --clock 4 --aio 4 --aio-requests 30`
- **Open Run:** with *open* stressors, in order to make continuously open/close system call,
 - `stress-ng --open 4`
- **Fork Run:** with *fork* stressors, in order to create *new child process*,
 - `stress-ng --fork 4`
- **Udp Run:** with *udp* stressors, in order to stress the *network*.
 - `stress-ng --udp 4`

4.1.2. Inception-v1 and Inception-v4

```
#!/usr/bin/python3

import jetson.inference
import jetson.utils
import os
from os import listdir
from time import perf_counter
import argparse

# parse the command line
parser = argparse.ArgumentParser()
parser.add_argument("filename", type=str, help="filename of the image to process")
parser.add_argument("testtype", type=str, help="type of test executed")
parser.add_argument("--network", type=str, default="googlenet", help="model to use, can be: googlenet, resnet-18, ect.")
args = parser.parse_args()

net = jetson.inference.imageNet(args.network)
```

```
times = []

for image in os.listdir(args.filename):
    # load an image (into shared CPU/GPU memory)
    img = jetson.utils.loadImage(image)

    #classify images
    t1 = perf_counter()

    class_idx, confidence = net.Classify(img)

    t2 = perf_counter()

    times.append(t2-t1)

    # find the object description
    class_desc = net.GetClassDesc(class_idx)

    # print out the result
    print("image is recognized as '{:s}' (class {:d}) with {:.f}%\nconfidence".format(class_desc, class_idx, confidence * 100))

avg_time = sum(times)/len(times)

with open(args.network + "_result_" + args.testtype+".txt", "a") as file:
    file.write(str(avg_time)+"\n")
```

First of all, it's necessary to retrieve some parameters from the command line:

- **filename** is used to specify the path for the whole set of images (for the detection)
- **testtype** is needed to determine which kind of test will be executed
- **--network** specifies which kind of NN will be used for the detection

In order to store each inference time (for each image within the folder), on which an average will be calculated, it's necessary to define a collection of times.

For every element in the folder specified in the `filename` parameter, there are two tasks to do:

- *perform the classification*
- *calculate the times between the start and the end of the detection*

Once the model has finally detected all the images, an average of the inference times has been calculated.

Finally, in order to store our times persistently, all the averages have to be appended in a txt file named properly.

4.1.3. SSD-MobileNet-v1 and SSD-MobileNet-v2

The code shown below is straight similar to that previously presented, with some minor differences made in order to do object detection on the dataset.

```
#!/usr/bin/python3

import jetson.inference
import jetson.utils
import os
from os import listdir
from time import perf_counter, sleep
import argparse

# parse the command line
parser = argparse.ArgumentParser()
parser.add_argument("filename", type=str, help="path of the images to process")
parser.add_argument("testtype", type=str, help="type of the test executed (golden, cpu...)")
parser.add_argument("--network", type=str, default="ssd-mobilenet-v1", help="model to use, can be: googlenet, resnet-18, ect.")
args = parser.parse_args()

times= [] #list of elapsed inference time (for each image within the folder specified in the cli)

#load the recognition network
net = jetson.inference.detectNet(args.network)

for image in os.listdir(args.filename): #for every image in the directory specified

    # load an image (into shared CPU/GPU memory)
    img = jetson.utils.loadImage(image)

    # classify the image
    t1 = perf_counter()
    detections_list = net.Detect(img)
    t2 = perf_counter()
    times.append(t2-t1)

# find the object des
for i in range(len(detections_list)):
    print("The class detected within the box is {:s} with confidence
```



```
{:f} : ".format(net.GetClassLabel(detections_list[i].ClassID),
detections_list[i].Confidence*100))
avg_times = sum(times)/len(times) #process the average time of object
detection

print("Average Time of test: ", avg_times)

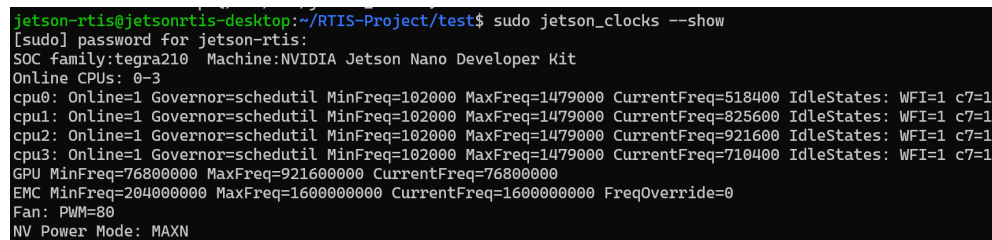
with open(args.network + "_result_" + args.testtype + ".txt", "a") as file:
    file.write(str(avg_times) + "\n")
```

4.2. Golden run and CPU run tests with jetson_clocks

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

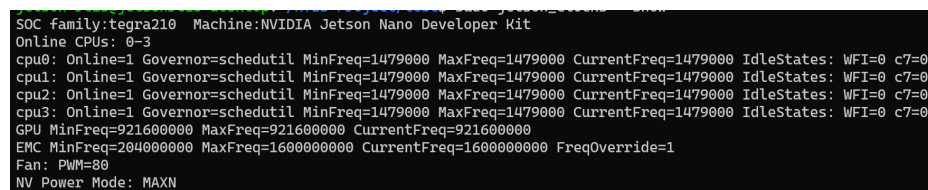
- **jetson_clocks** is a tool provided for all *NVIDIA Jetson* to maximize all performance.

The following screenshots show the difference (in terms of clock frequency) between when the tool is enabled and when it's not.



```
jetson-rtis@jetsonrtis-desktop:~/RTIS-Project/test$ sudo jetson_clocks --show
[sudo] password for jetson-rtis:
SOC family:tegra210 Machine:NVIDIA Jetson Nano Developer Kit
Online CPUs: 0-3
cpu0: Online=1 Governor=schedutil MinFreq=102000 MaxFreq=1479000 CurrentFreq=518400 IdleStates: WFI=1 c7=1
cpu1: Online=1 Governor=schedutil MinFreq=102000 MaxFreq=1479000 CurrentFreq=825600 IdleStates: WFI=1 c7=1
cpu2: Online=1 Governor=schedutil MinFreq=102000 MaxFreq=1479000 CurrentFreq=921600 IdleStates: WFI=1 c7=1
cpu3: Online=1 Governor=schedutil MinFreq=102000 MaxFreq=1479000 CurrentFreq=710400 IdleStates: WFI=1 c7=1
GPU MinFreq=76800000 MaxFreq=921600000 CurrentFreq=76800000
EMC MinFreq=204000000 MaxFreq=1600000000 CurrentFreq=1600000000 FreqOverride=0
Fan: PWM=80
NV Power Mode: MAXN
```

Figure 2: clock frequencies for each core without enabling jetson_clocks



```
SOC family:tegra210 Machine:NVIDIA Jetson Nano Developer Kit
Online CPUs: 0-3
cpu0: Online=1 Governor=schedutil MinFreq=1479000 MaxFreq=1479000 CurrentFreq=1479000 IdleStates: WFI=0 c7=0
cpu1: Online=1 Governor=schedutil MinFreq=1479000 MaxFreq=1479000 CurrentFreq=1479000 IdleStates: WFI=0 c7=0
cpu2: Online=1 Governor=schedutil MinFreq=1479000 MaxFreq=1479000 CurrentFreq=1479000 IdleStates: WFI=0 c7=0
cpu3: Online=1 Governor=schedutil MinFreq=1479000 MaxFreq=1479000 CurrentFreq=1479000 IdleStates: WFI=0 c7=0
GPU MinFreq=921600000 MaxFreq=921600000 CurrentFreq=921600000
EMC MinFreq=204000000 MaxFreq=1600000000 CurrentFreq=1600000000 FreqOverride=1
Fan: PWM=80
NV Power Mode: MAXN
```

Figure 3: clock frequencies for each core when enabling jetson_clocks

As can be noticed from those images, 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.

For this type of test, **Inception-V4** model was preferred because is the most complex one and, in addition, as shown by the previous tests, is the model with the highest inference times among all the models.

4.3. Golden run and CPU run tests with CPU isolation (*isol_cpu*)

4.3.1. Configuration

In order to enable the CPU core isolation properly, it's necessary to edit the **extlinux.conf** kernel configuration file (following the **/boot/extlinux/** path) by modifying the **primary** section as follows:

```

LABEL primary
MENU LABEL primary kernel
LINUX /boot/Image
INITRD /boot/initrd
APPEND ${cbootargs} quiet root=PARTUUID=6cc2cca0-ea5f-01db-5066-61b62ff5ed00
rw rootwait rootfstype=ext4 console=ttyS0,115200n8 console=tty0 fbcon=map:0
net.ifnames=0 sdhci_tegra.en_boot_part_access=1 isolcpus=1 rcu_nocbs=1

```

Let's focus on the last two parameters within the **APPEND** line: **isolcpus=1** specifies which cores will be isolated (it can be specified all the cores except for the 0 core, which is used for *house-keeping*: in that case, only the core 1 will be isolated), **rcu_nocbs=1** describes which cores will be offloaded by the latencies caused by RCU callbacks (which are invoked in softirq context). Subsequently, it's necessary to do **interrupt offloading** from those isolated cpus by modifying the *default_smp_affinity* text file: the default value is **f** (remember, it's an hexadecimal value and its value is 1111), which means that all the cpu cores will be used for the interrupt handling, but, although only core 1 have been isolated, that value needs to be overwritten with **D** (and its value is 1101).

Note that when trying to change the affinity manually, an **Input/Output** error will occurs: this happens because some interrupts can't be removed from a specific core, they are fixed.

For this kind of test, **SSD-MobileNet-V1** model has been chosen because it's quite fast, lightweight and figured as a low-latency model in previous tests.

4.4. Stress tests with GPU stressors

In this section, it's possible to note how GPU stressors impact on general inference performances of **SSD-MobileNet-V1** and **Inception-V4** models. The tests executed are made with three different matrix dimensions (tiny, small, medium and large):

- 50x50 * 50x50
- 800x800 * 800x800 (29.5Mb of GPU memory allocated)
- 1600x1600 * 1600x1600 (51.4Mb of GPU memory allocated)
- 2400x2400 * 2400x2400 (88Mb of GPU memory allocated)
- 3200x3200 * 3200x3200 (139Mb of GPU memory allocated)

Note: for what concerns «50x50 * 50x50» tests, something went wrong:

```

Error! Matrix[02479]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02480]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02481]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02482]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02483]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02484]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02485]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02486]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02487]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02488]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02489]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02490]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02491]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02492]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02493]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02494]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02495]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02496]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02497]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02498]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Error! Matrix[02499]=0.00000000, ref=0.50000000 error term is > 1.000000E-06
Result = FAIL

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Maxwell" with compute capability 5.3
MatrixA(50,50), MatrixB(50,50)

```

As shown in the picture, the computed values are all 0, while the expected reference values are 0.5. This leads to an error term greater than the acceptable threshold, as indicated in the output, causing a crash of the stressor.

A *while true* script (using the tool **matrixMul**) has been written in order to execute various matrix multiplications in sequence to generate stress.

```

while :; do /usr/local/cuda/samples/0_Simple/matrixMul/matrixMul -wA=800 -
hA=800 -wB=800 - hB=800; done

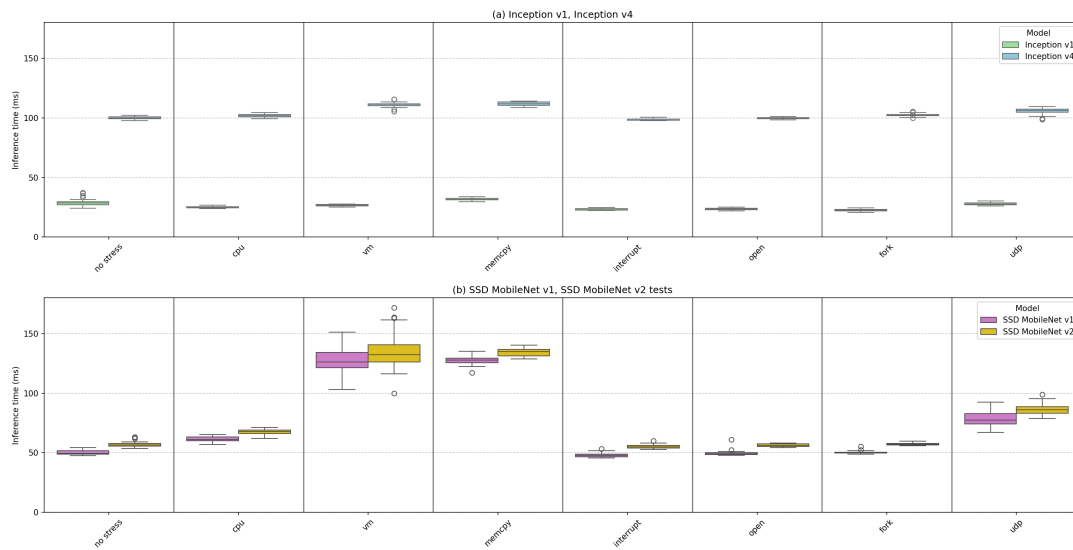
```

4.4.1. MatrixMul

This tool is already present in the SDK, located at the following path: `/usr/local/cuda/samples/0_Simple/matrixMul`. With this, we can perform matrix multiplications directly on the GPU. After compiling with **make** command (`matrixMul.cu`), we setted the **Power Mode** with `sudo /usr/sbin/nvpmode -m <x>` where in our case is 0: that indicates the Jetson Nano 10W Power Mode (<https://docs.ai-blox.com/pages/blox/hardware/stress-test.html>).

5. Results

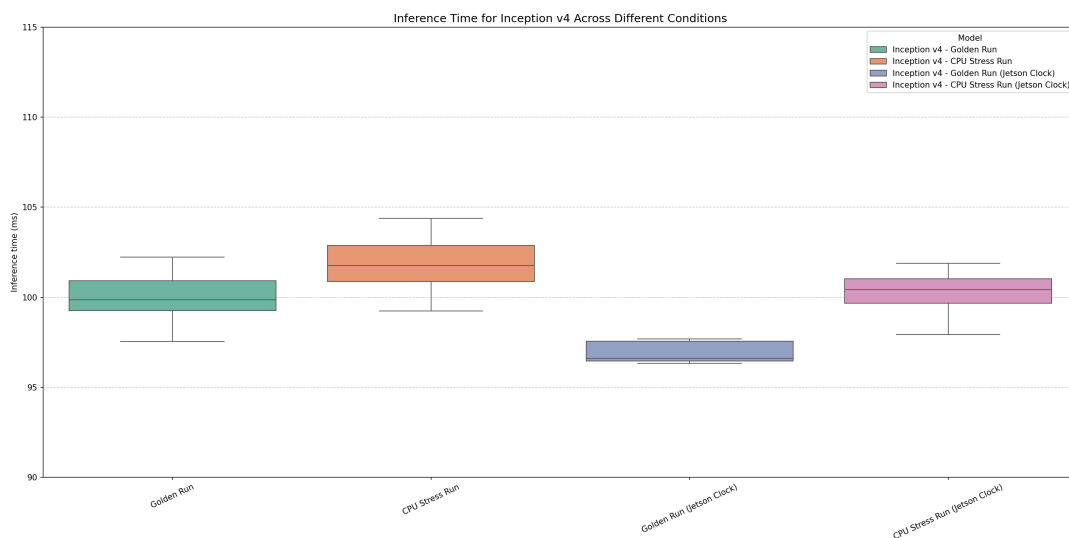
5.1. Stress tests with *stress_ng*



As the plot shows, the **Inception-V1 and Inception-V4** models remain highly effective even under stress: their inference times remain stable and closely aligned with those of the golden run except under the vm and memcopy stressors, where Inception-V4 shows a slight increase. Inception-V1, in contrast, demonstrates even better resilience, with its inference times deviating only minimally from the «no stress» baseline.

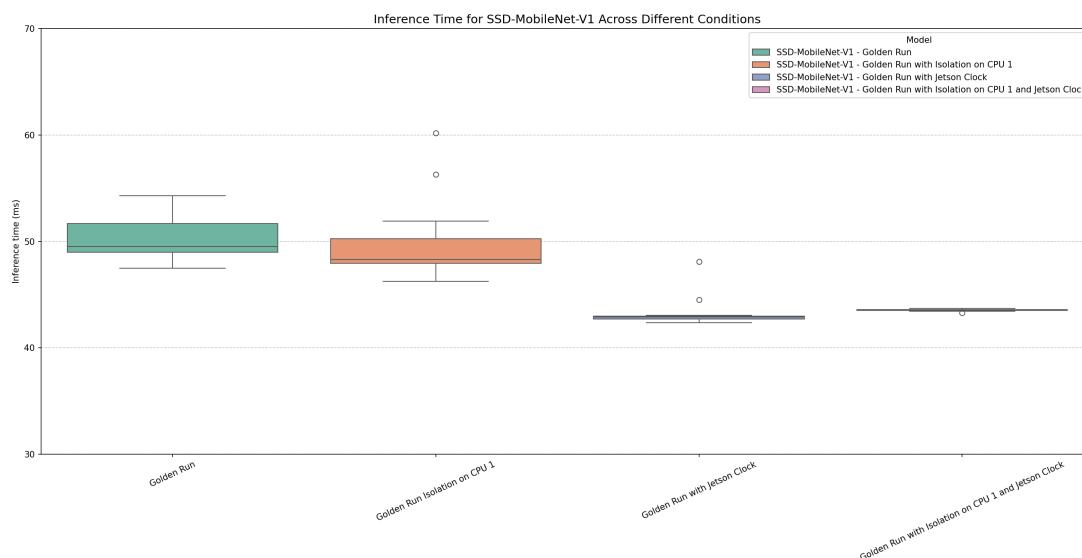
As for **SSD-MobileNet-V1 and SSD-MobileNet-V2**, their inference times vary significantly compared to the baseline. In particular, under vm and memcopy stressors, inference times increase dramatically with vm showing a marked lengthening of the inference intervals. Lastly, a slight increase in UDP inference time compared to the baseline can also be observed.

5.2. Golden run and CPU run tests with *jetson_clocks*



In this case, the jetson clock mode was enabled, which, as previously mentioned, sets the CPU frequencies to their maximum values. For this test, the Inception **V4 model** was used due to its high latency. As the plot shows, latencies are slightly lower and exhibit reduced variance compared to the golden run without jetson clock mode enabled. It is also observed that, under some **CPU stress**, both latency and variance slightly increase, although the results remain visually comparable to the golden run. Indeed, as anticipated by visual inspection, the **Kolmogorov-Smirnov (kstest2)** test rejects the null hypothesis when comparing the Golden Run distribution with the Golden Run (Jetson Clock). Conversely, the test does not reject the null hypothesis when comparing the Golden Run and the CPU Stress Run (Jetson Clock), indicating that these distributions are statistically similar.

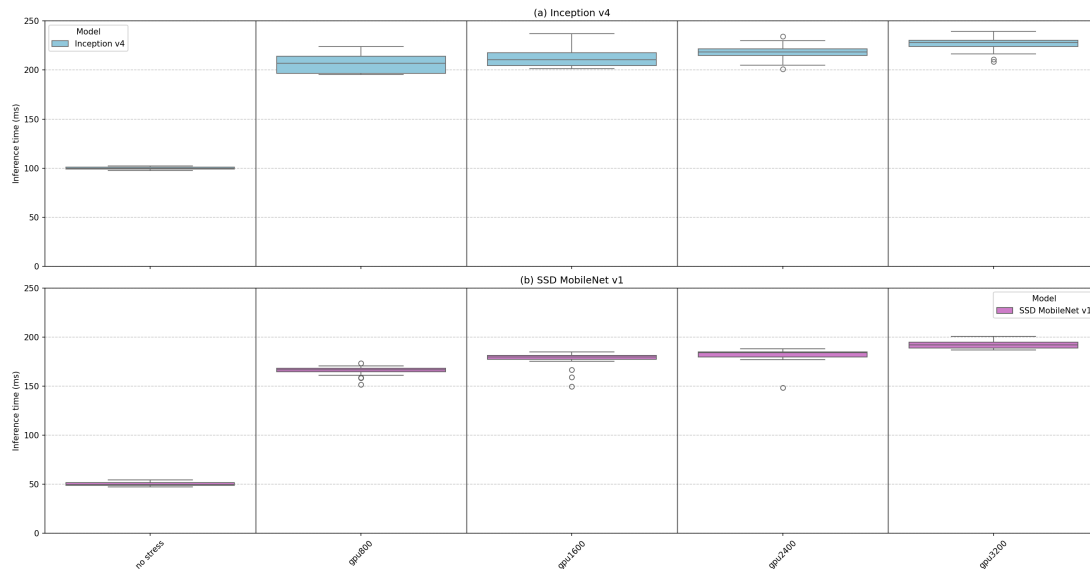
5.3. Golden run and CPU run tests with CPU isolation (*isol_cpu*, *rcu_nocbs*)



In this test, the inference time distributions of the *Golden Run* and the *Golden Run with Jetson Clock*, along with their respective runs using **isolation methods (isolcpus and rcu_nocbs)** for the **SSD-MobileNet-V1** model, are compared. As shown in the plot, isolating the inference task does not significantly improve latency or variance compared to the Golden Run.

Otherwise, the run with **Jetson Clock** mode and its corresponding version with isolation methods show a notable decrease in both latency and variance.

5.4. Stress tests with GPU stressors



In this final test, the stress was shifted to the GPU to create interference during the inference of the **Inception-V4** and **SSD-MobileNet-V1** models. As shown in the plot, when the dimension of the matrices increase, the inference times for Inception-V4 rise progressively, with a noticeable deviation from the baseline — although the variability remains relatively stable. As for SSD-MobileNet-V1, its behavior is similar to that of Inception-V4, but the inference times under stress are consistently lower (and also stable) compared to the other model.