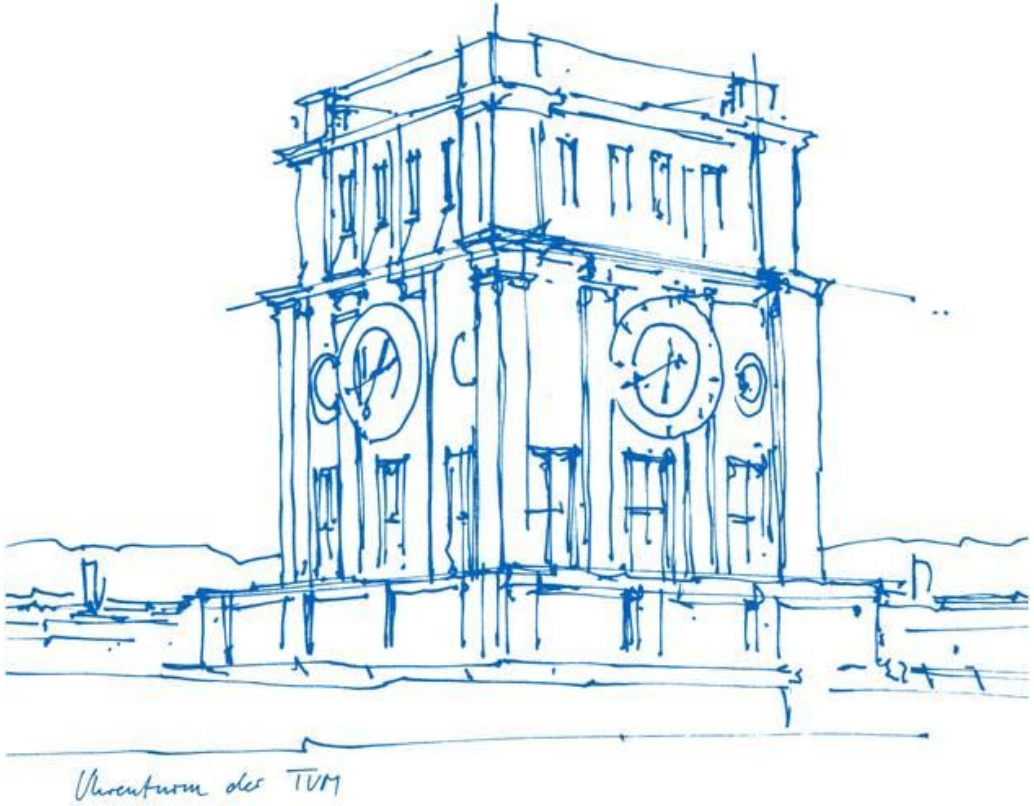
Chair of Real-Time Computer Systems
TUM Department of Electrical and Computer Engineering
Technical University of Munich

Mapping Artificial Neural Network Operations for Inference on Coral Edge TPU

Bachelor’s Thesis



Supervisor:
M.Sc. Alex Hoffman

Advising Professor:
Prof. Dr. Daniel Müller-Gritschneder

Student:
Ala Fnayou
ge69faf
03697266

Lehrstuhl für
Realzeit-Computersysteme

Motivation: Edge AI

Edge Computing + Artificial Intelligence = Edge AI

- ❖ Edge AI systems
  - ➢ Process data gathered by hardware devices locally
  - ➢ Eliminate privacy and security issues related to data transfer
  - ➢ Reduce network latency times for an improved user experience

⇒ Emergence of a new breed of revolutionary products

Improvement of Market value of Edge AI between 2018 and 2023 [1]

Amazon’s Alexa and Google Home [2]

Set of futuristic wearable products [3]

Motivation: Tiny Machine Learning

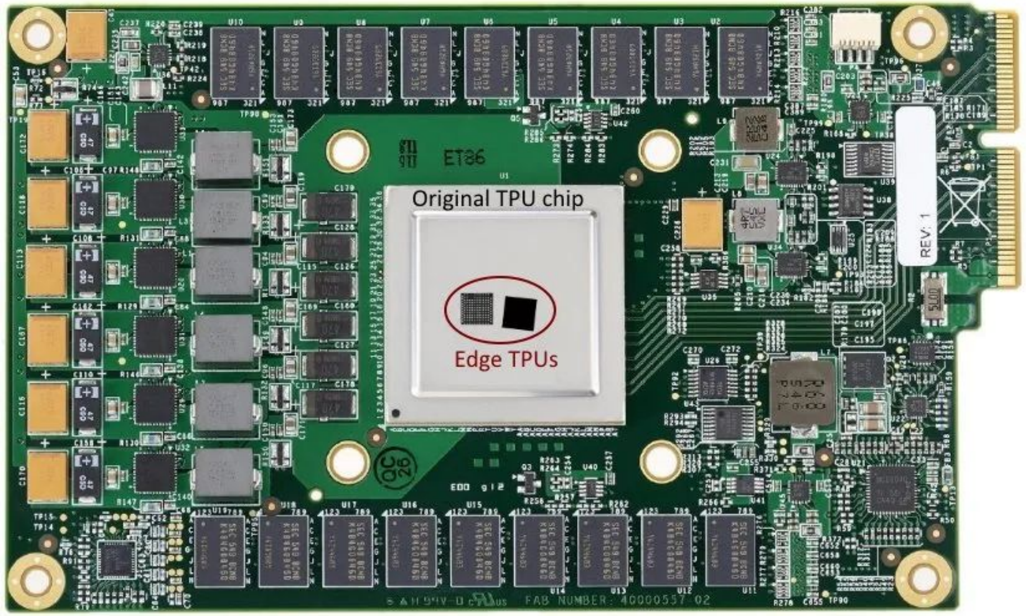
- Tiny ML represents the new trend of running ML algorithms on memory and power constrained platforms.
  - ➔ Requires the design of new optimization techniques aimed at the hardware and software level.

Hardware

Google's Coral Edge TPU

Software

Distributed Machine Learning based on the Design Space Exploration methodology

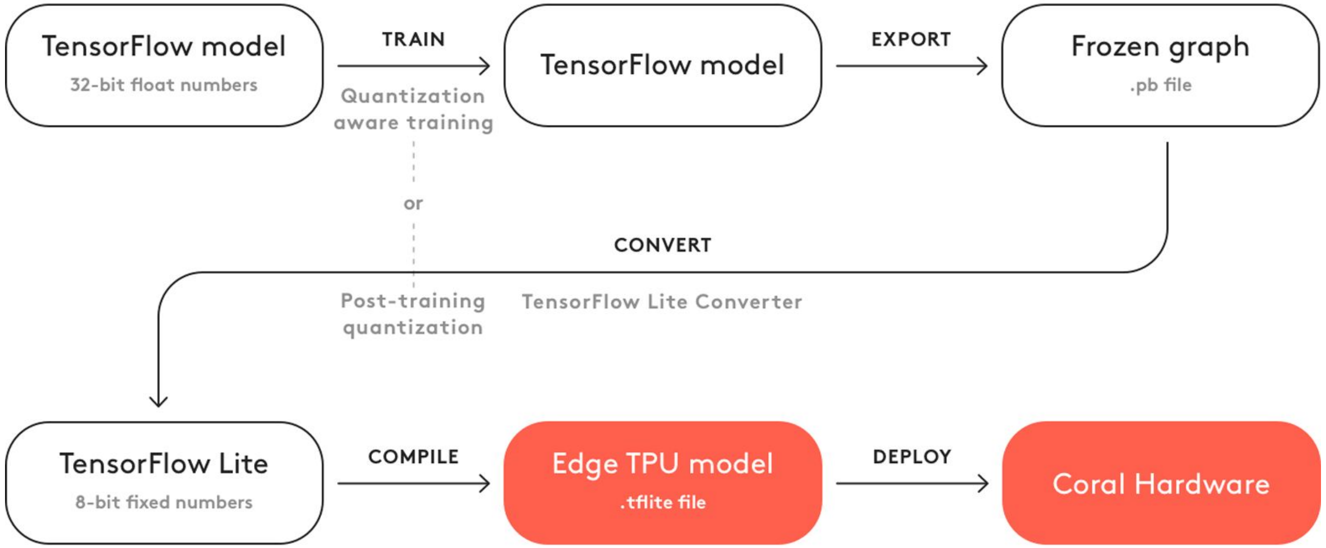


Comparison between TPU and Edge TPU chips [4]

Background: TensorFlow Models on the Edge TPU

- Design Space Exploration analysis generates an **operation-mapping** that distributes the inference of a ML model on heteregeneous hardware devices including Google's Coral Edge TPU USB Accelerator

- Running models on the Edge TPU requires
  - Tensor parameters to be quantized (8-bit fixed-point numbers; int8 or uint8)
  - The model to use only the operations supported by the Edge TPU



Workflow of deploying TF models on the Edge TPU [5]

Background: TensorFlow Lite and FlatBuffers

- ❖ TensorFlow Lite models are saved using an optimized file extension based on FlatBuffers, namely `.tflite`
- ❖ Using the `schema.fbs` file provided by TensorFlow and the FlatBuffers schema compiler it is possible to
  - ➢ Generate Python helper classes to access and construct serialized data in the model
  - ➢ Convert the model from binary to JSON
- ❖ A TF Lite model can be represented as
  - ➢ a **read-only** binary saved in the `.tflite` file extension
  - ➢ a **read/write** JSON saved in the `.json` file extension

⇒ **TensorFlow Lite models can be freely modified once converted to JSON**

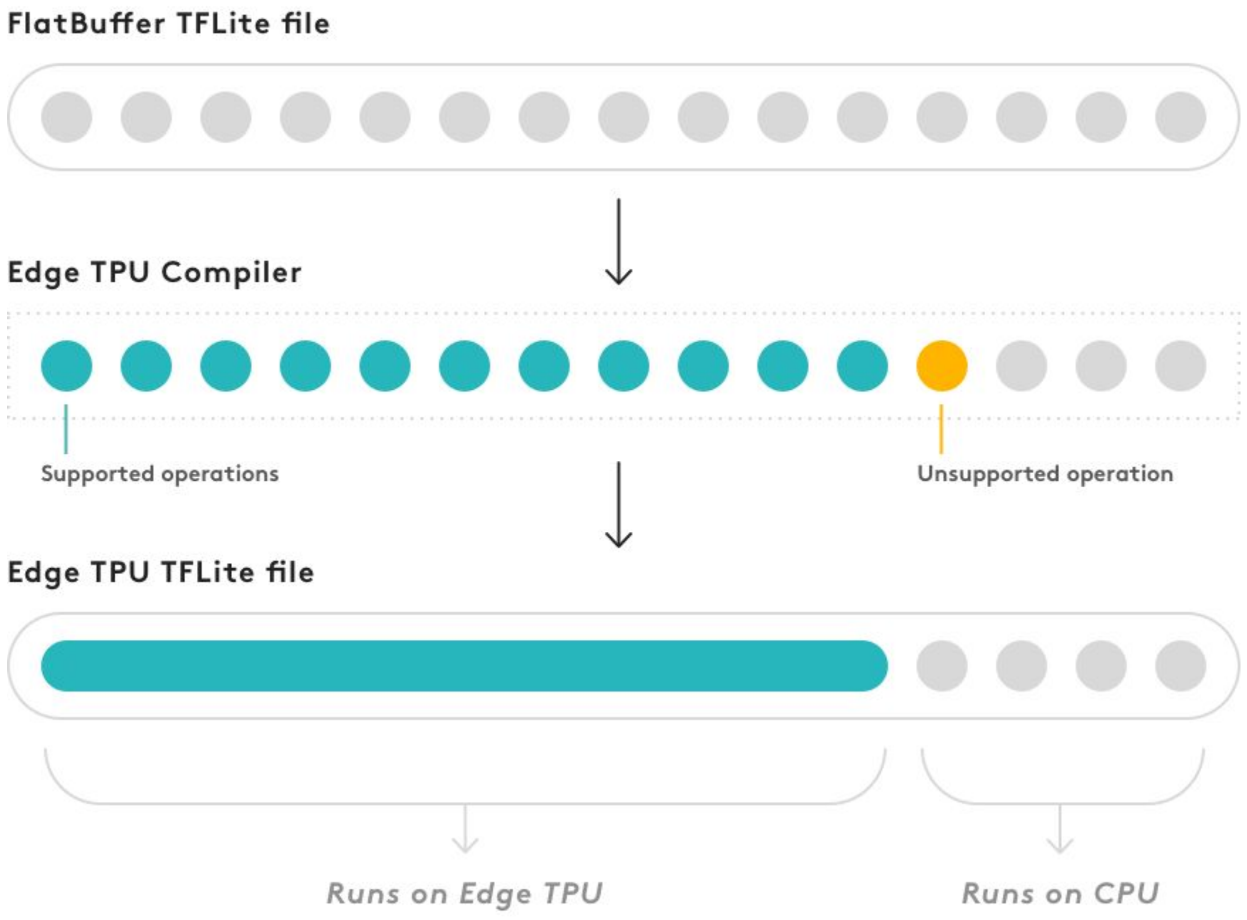
Problem Statement

- ❖ The Edge TPU Compiler
  - ➢ analyses the operations present in the model
  - ➢ stops When an unsupported operations is encountered

⇒ **Only one** portion of the model can be mapped to the Edge TPU

- ❖ The Edge TPU Compiler is closed-source software

⇒ **Impossible** to change internal behavior

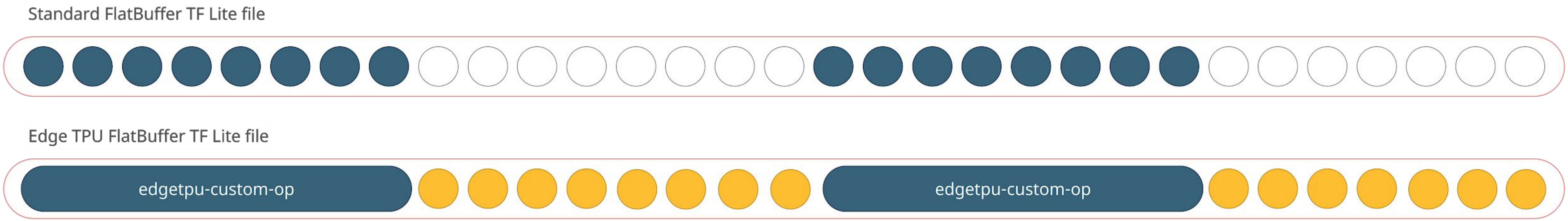


Behavior of the Edge TPU Compiler [6]

Problem Statement: Desired Behavior

- The described behavior makes the execution of an efficient operation-mapping generated by the DSE analysis **impossible** to realize
- The proposed solution succeeds at overcoming this challenge

⇒ Allows mapping of any operation either to the Edge TPU or to the CPU



Related Work: Software Level Approaches

- **Virtualizing AI at the Distributed Edge [7]**
  - ➢ Based on the IoT virtualization concept
  - ➢ Design of a virtualization layer responsible for the semantic description of AI-embedded IoT devices

⇒ Relieving the pressure on constrained devices

⇒ Targeting interoperability among AI-powered platforms.

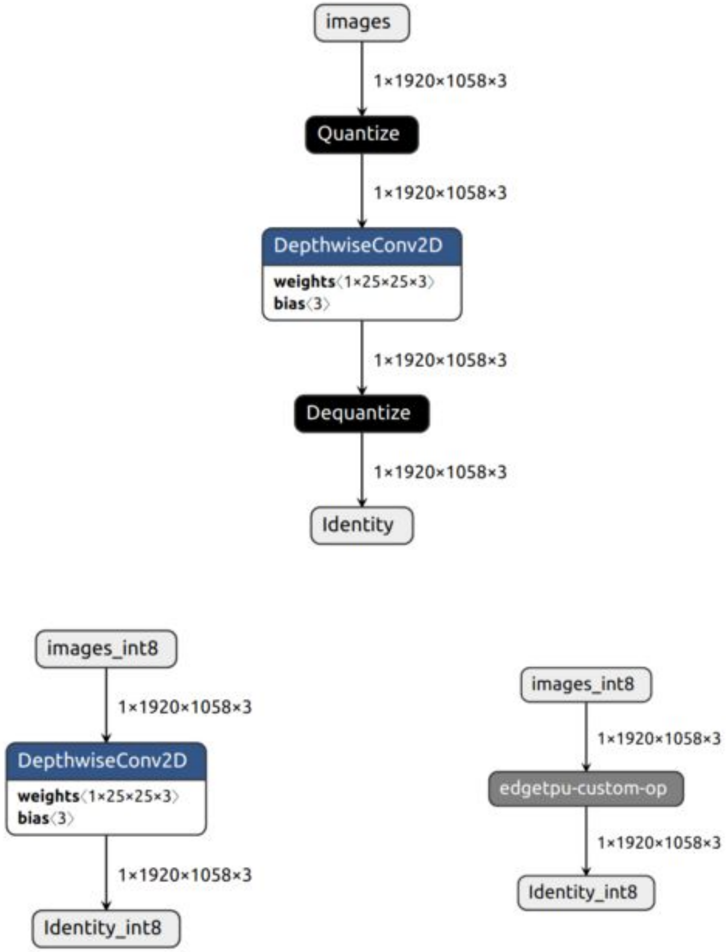
- **ADaptive Synchronous Parallel [8]**
  - ➢ a parameter synchronization model

⇒ Decreasing waiting time to a minimum while maintaining an optimized usage of computational resources.

Related Work: Upgrading a TF Lite model

- **Upgrading a TF Lite model to run Motion Blur on the Coral Edge TPU**
  - ➢ Converts TF Lite model to JSON
  - ➢ Removes unsupported operations
  - ➢ Changes Tensor data types

MotionBlur effect [9]



Model used to achieve MotionBlur visualized using Netron [9]

Goal and Approach

- **Goal of the thesis**
  - Map any operation present in a TF Lite model freely either to the Edge TPU or to the CPU.
- **Approach**
  - Use the JSON representation to modify the TF Lite model
  - Separate the operations mapped to the Edge TPU and save them into separate files
  - Compile the *submodels* separately using the `edgetpu-compiler`
  - Re-assemble the model to contain operations with different mapping targets

Implementation: TF Lite model in JSON

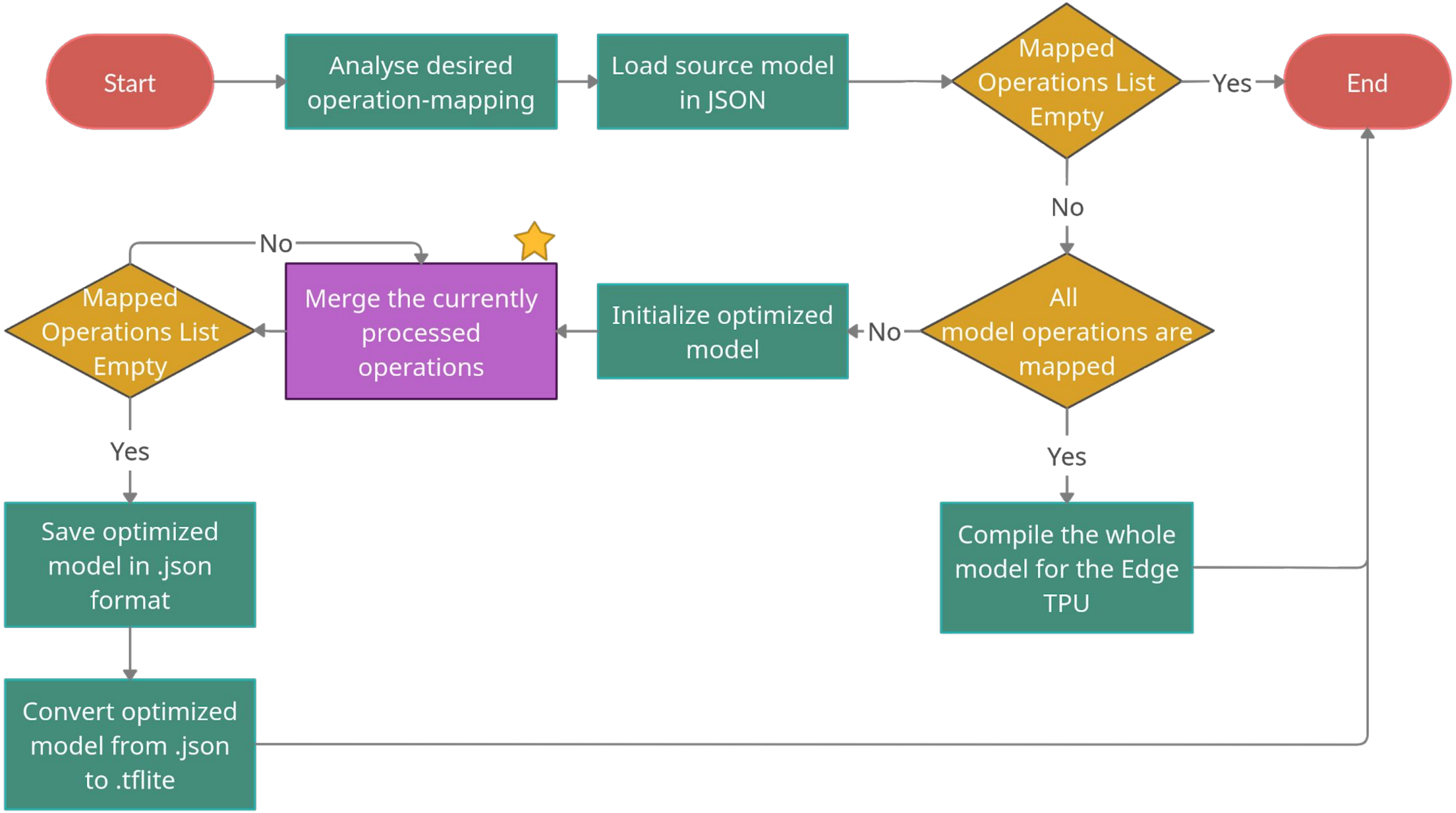
- Structure of a JSON file representing TF Lite model

```
{
  "version": 3,
  "operator_codes": [
     {
       "deprecated_builtin_code": 3,
       "version": 1,
       "builtin_code": "ADD"
     }
  ],
  "subgraphs": [
  ],
  "description": "TOCO Converted.",
  "buffers": [
  ],
  "metadata": [
     {
       "name": "min_runtime_version",
       "buffer": 0
     }
  ]
}
```

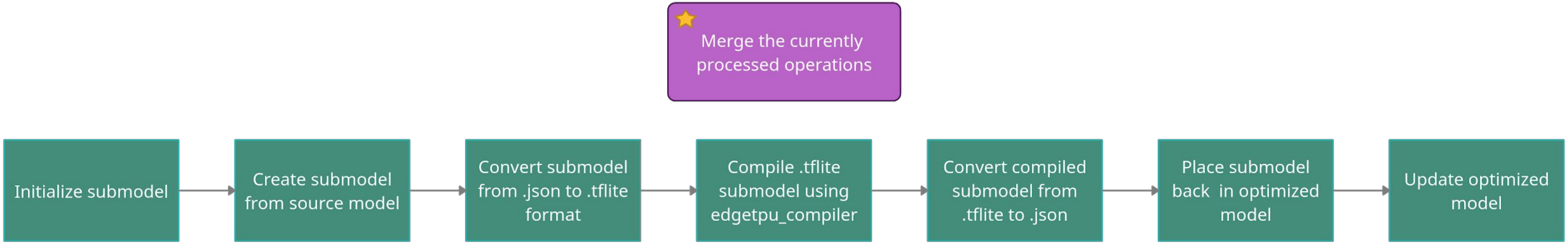
- Structure of the `"subgraphs"` element

```
"subgraphs": [
    {
      "tensors": [
      ],
      "inputs": [
      ],
      "outputs": [
      ],
      "operators": [
        {
          "opcode_index": 0,
          "inputs": [
          ],
          "outputs": [
          ],
          "builtin_options_type":,
          "builtin_options": {
          },
          "custom_options_format":,
          "mutating_variable_inputs": []
        }
      ]
    }
  ]
```

Implementation: Algorithm Flow Chart

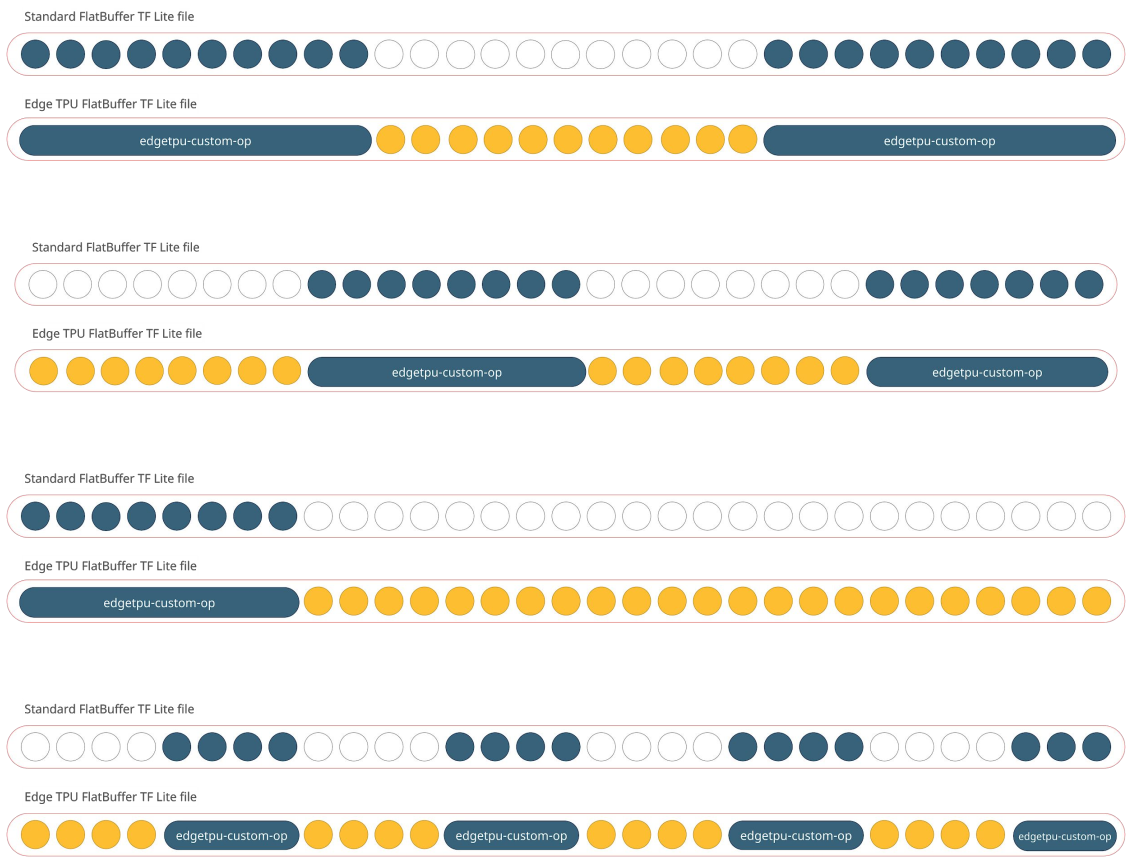


Implementation: Merging Operations

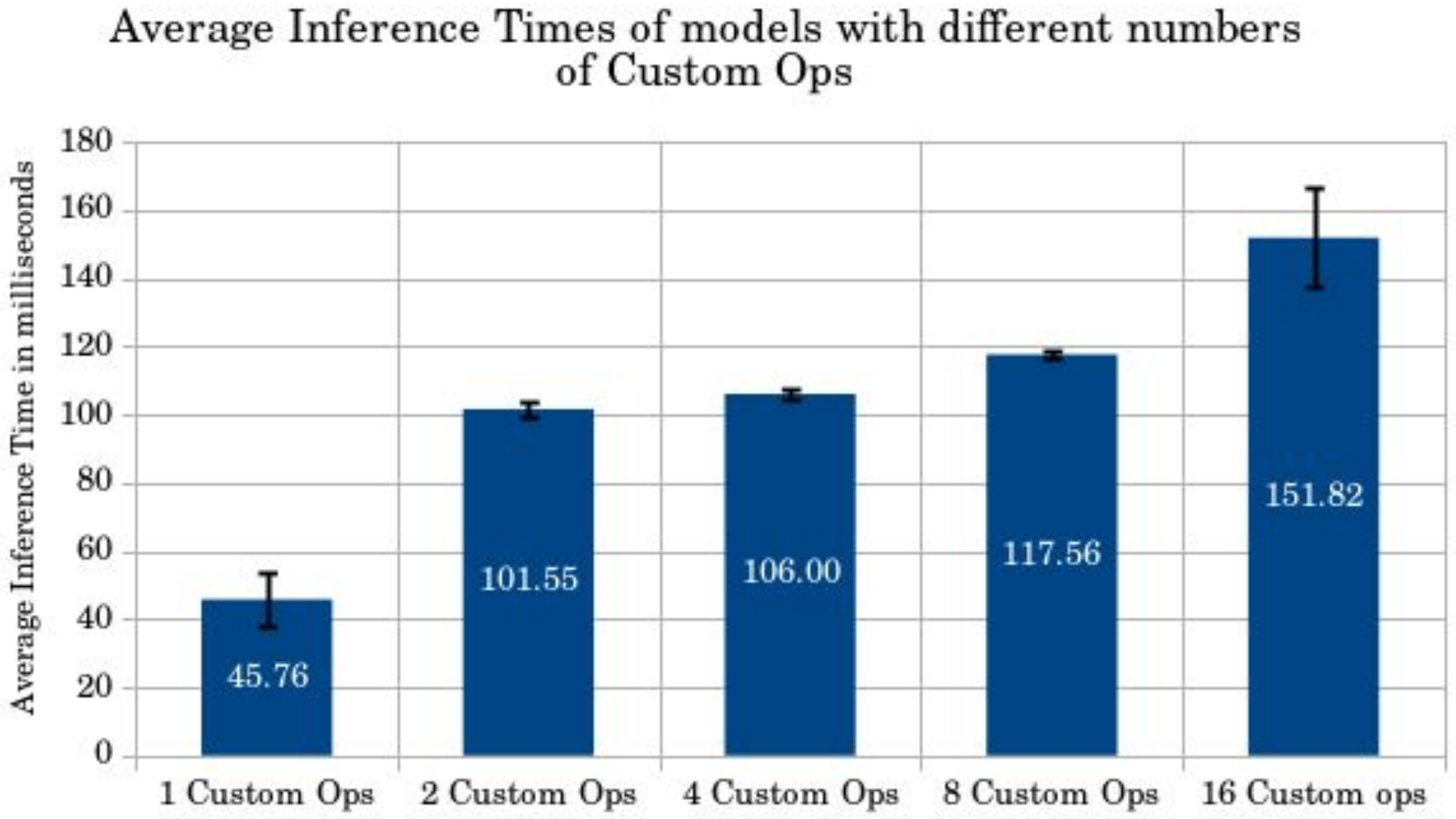


Experimental Setup

- A series of experiments was conducted involving
  - Creation of multiple optimized models, each representing a mapping scenario
  - Benchmarking each model
  - Gathering Python inference times
- The mapping scenarios aim at highlighting the effect of varying some parameters on the inference time
  - Number of `edgetpu-custom-ops` in the model
  - Total number of operations mapped to the Edge TPU
  - The target hardware on which the model starts its execution



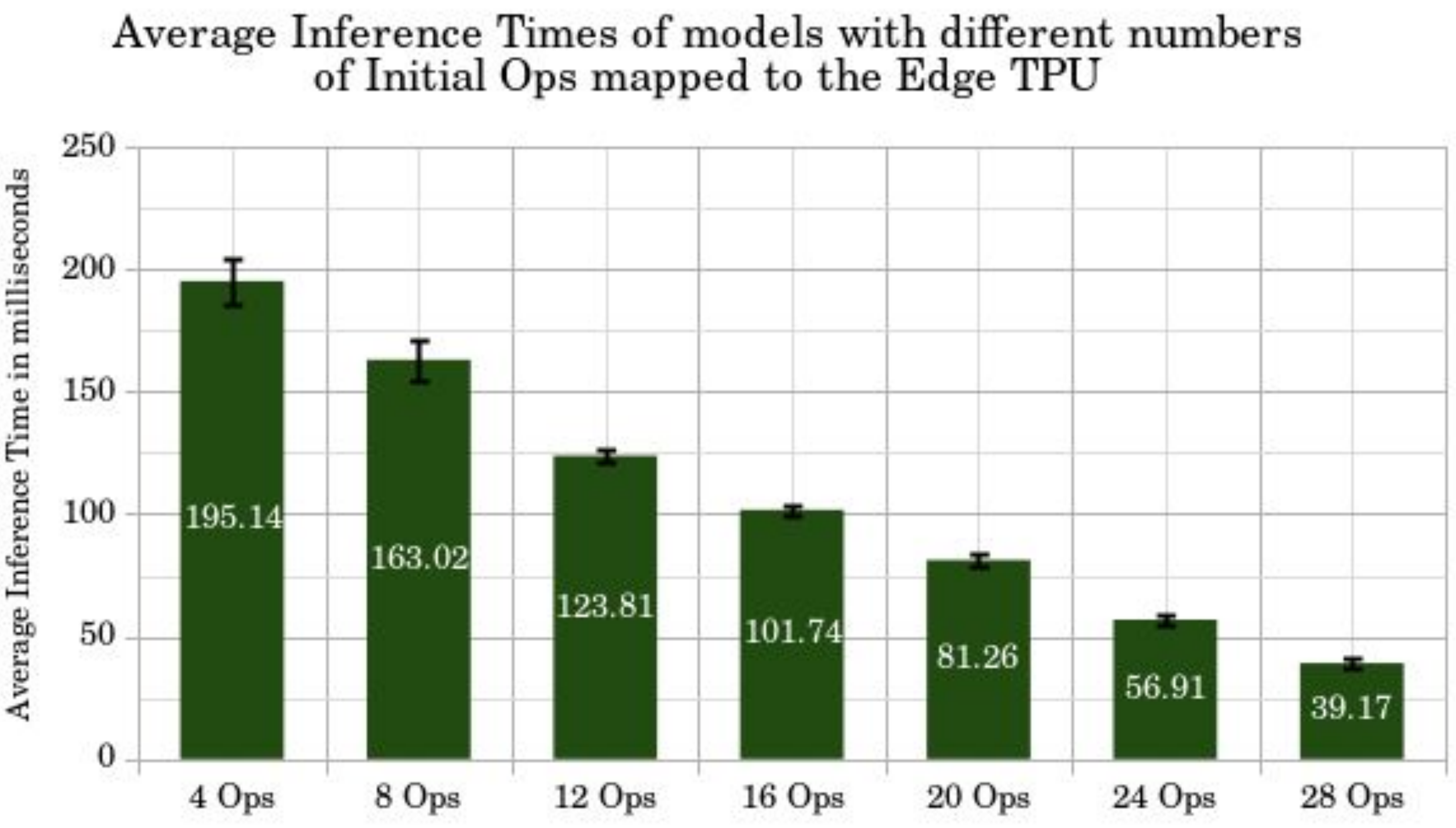
Results: Varying the number of edgetpu-custom-ops



N° of Custom Ops	Increase in %
1 Custom Ops	
2 Custom Ops	55%
3 Custom Ops	4%
4 Custom Ops	10%
5 Custom Ops	23%

⇒ Increasing the number of Custom Ops while maintaining the same number of operations mapped to the Edge TPU results in an increase in inference times

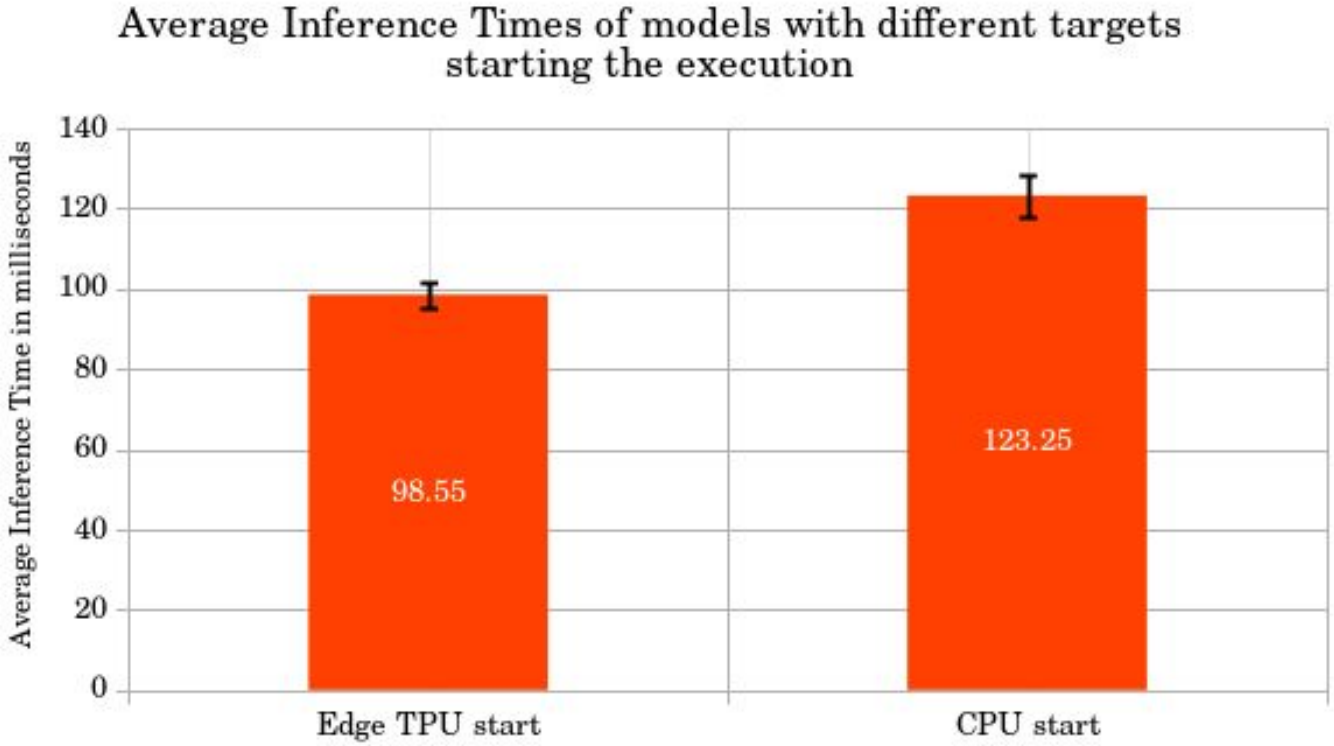
Results: Varying the total number of Initial operations mapped to the Edge TPU

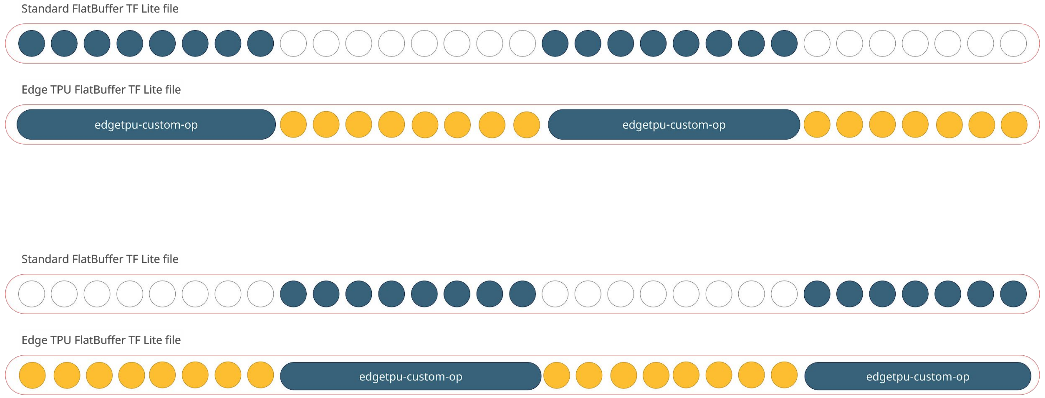


N° of Ops mapped to the Edge TPU	Decrease in %
4 Ops	
8 Ops	16%
12 Ops	24%
16 Ops	18%
20 Ops	20%
24 Ops	30%
28 Ops	32%

⇒ Increasing the total number of Ops mapped to the Edge TPU results in a decrease in inference times

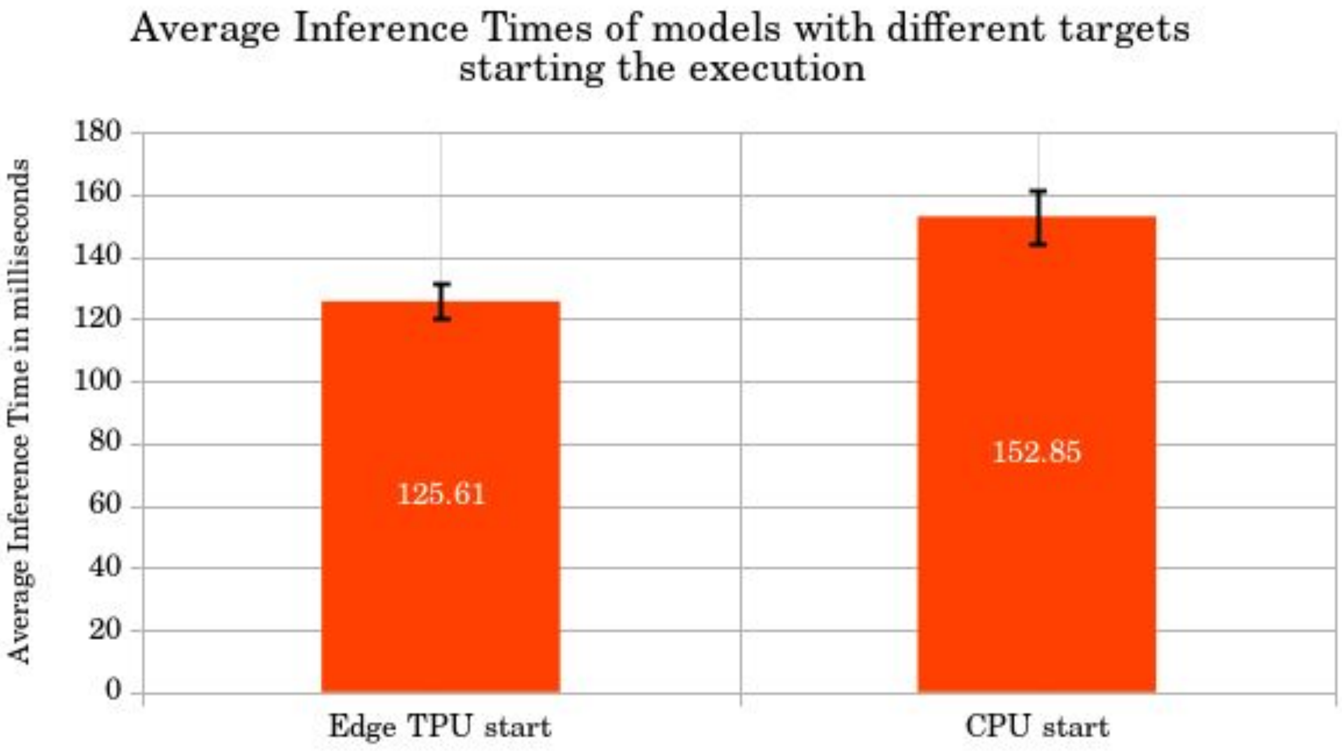
Results: Varying the target hardware on which the model starts its execution

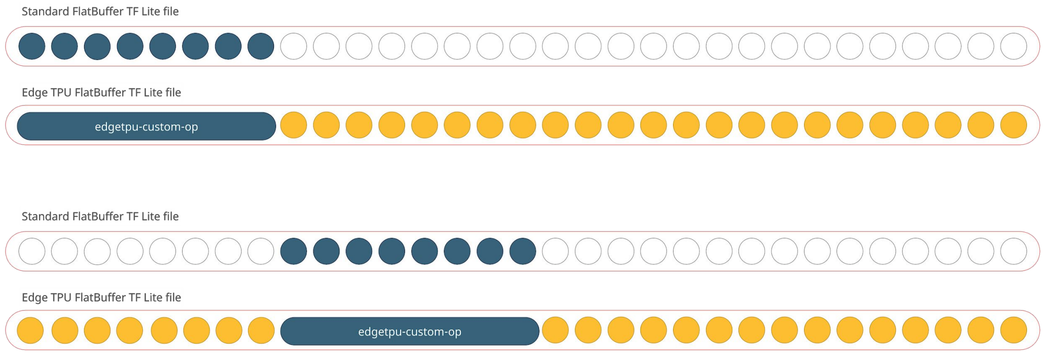




⇒ Starting execution on the CPU results in a 20% increase in inference times

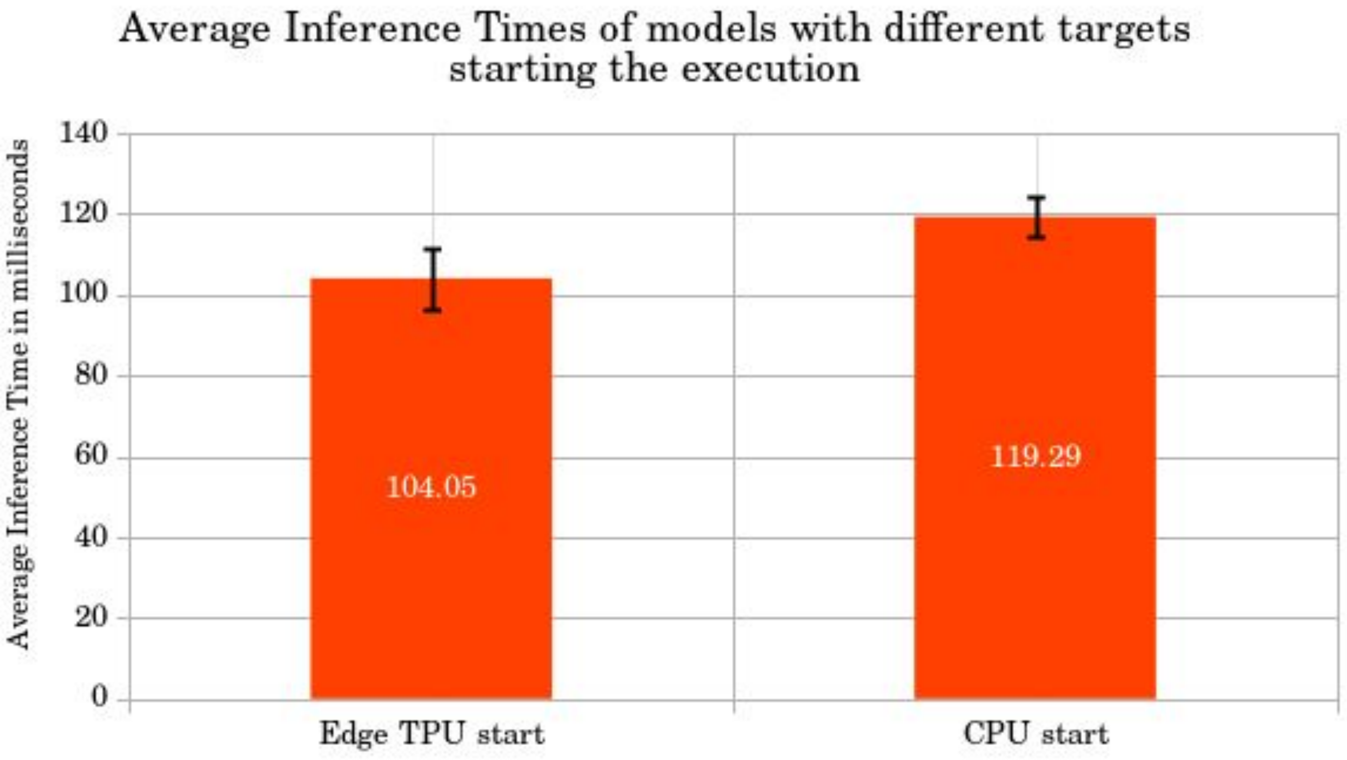
Results: Varying the target hardware on which the model starts its execution

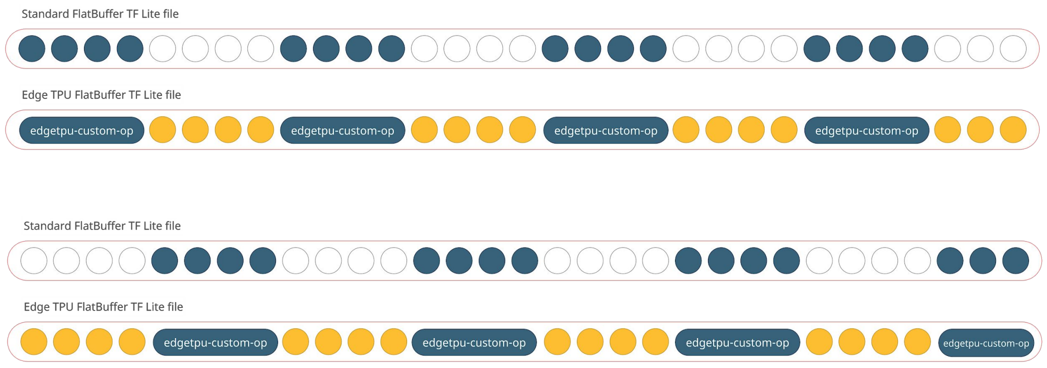




⇒ Starting execution on the CPU results in a 18% increase in inference times

Results: Varying the target hardware on which the model starts its execution





⇒ **Starting execution on the CPU results in a 15% increase in inference times**

Conclusions and Future Work

- Conclusions
  - TF Lite models can be freely modified and upgraded once converted to JSON
  - Any combination of operations present in a TF Lite model can be freely mapped to either the Edge TPU or a general-purpose CPU

- Future Work
  - Support bigger and more complex models
  - Support more hardware targets like GPUs and potentially, embedded-edge devices

Bibliography

[1] Vector ITC, Edge AI: The Future of Artificial Intelligence, [Online]. Available: https://www.vectoritcgroup.com/en/tech-magazine-en/artificial-intelligence-en/edge-ai-el-futuro-de-la-inteligencia-artificial/

[2] [Online]. Available: https://uk.pcmag.com/speakers/85210/amazon-echo-vs-google-home-which-voice-controlled-speaker-is-right-for-you

[3] [Online]. Available: https://www.sportswearable.net/global-smart-wearables-and-sports-clothings-market-2019/

[4] [Online]. Available: https://qengineering.eu/google-corals-tpu-explained.html

[5] Coral-Team. Tensorflow models on the edge tpu. [Online]. Available: https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview

[6] Coral-Team. Tensorflow models on the edge tpu. [Online]. Available: https://coral.ai/docs/edgetpu/models-intro/#compiling

[7] C. Campolo, G. Genovese, A. Iera, and A. Molinaro, “Virtualizing ai at the distributed edge towards intelligent iot applications,” Journal of Sensor and Actuator Networks, vol. 10, no. 1, 2021. [Online]. Available: https://www.mdpi.com/2224-2708/10/1/13

[8] H. Hu, D. Wang, and C. Wu, “Distributed machine learning through heterogeneous edge systems,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 05, pp. 7179–7186, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/ article/view/6207

[9] V. Markovtsev. Hacking google coral edge tpu: motion blur and lanczos resize. [Online]. Available: https://towardsdatascience.com/ hacking-google-coral-edge-tpu-motion-blur-and-lanczos-resize-9b60ebfaa552

Chair of Real-Time Computer Systems
TUM Department of Electrical and Computer Engineering
Technical University of Munich

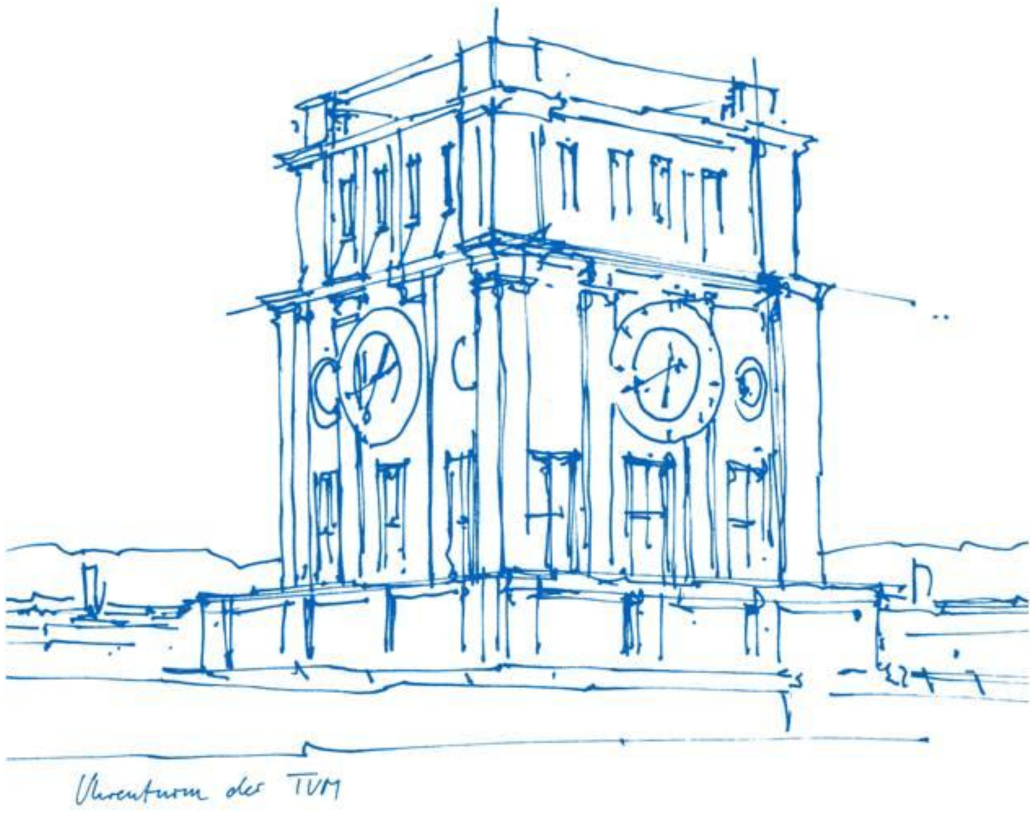
Benchmarking Inference on Google’s Coral Edge TPU

Research Internship

Supervisor:
M.Sc. Alex Hoffman

Advising Professor:
Prof. Dr. Daniel Müller-Gritschneder

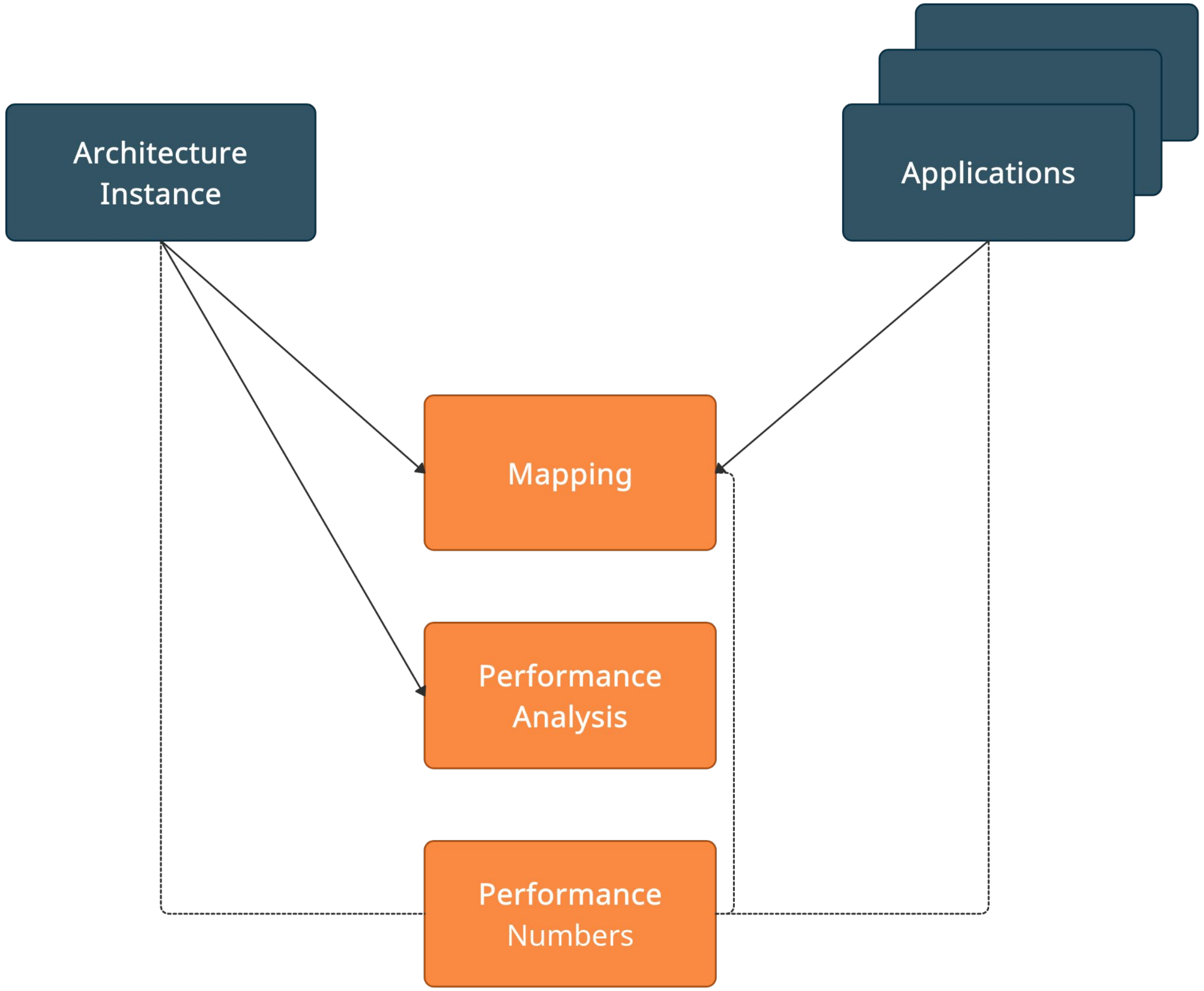
Student:
Daniel Duclos-Cavalcanti
ga74ped
03692475



Lehrstuhl für
Realzeit-Computersysteme

Introduction - Design Space Exploration

- **DSE**:
  - Optimal mapping to distribute inference of a ML model.
  - Done across heterogeneous hardware devices.



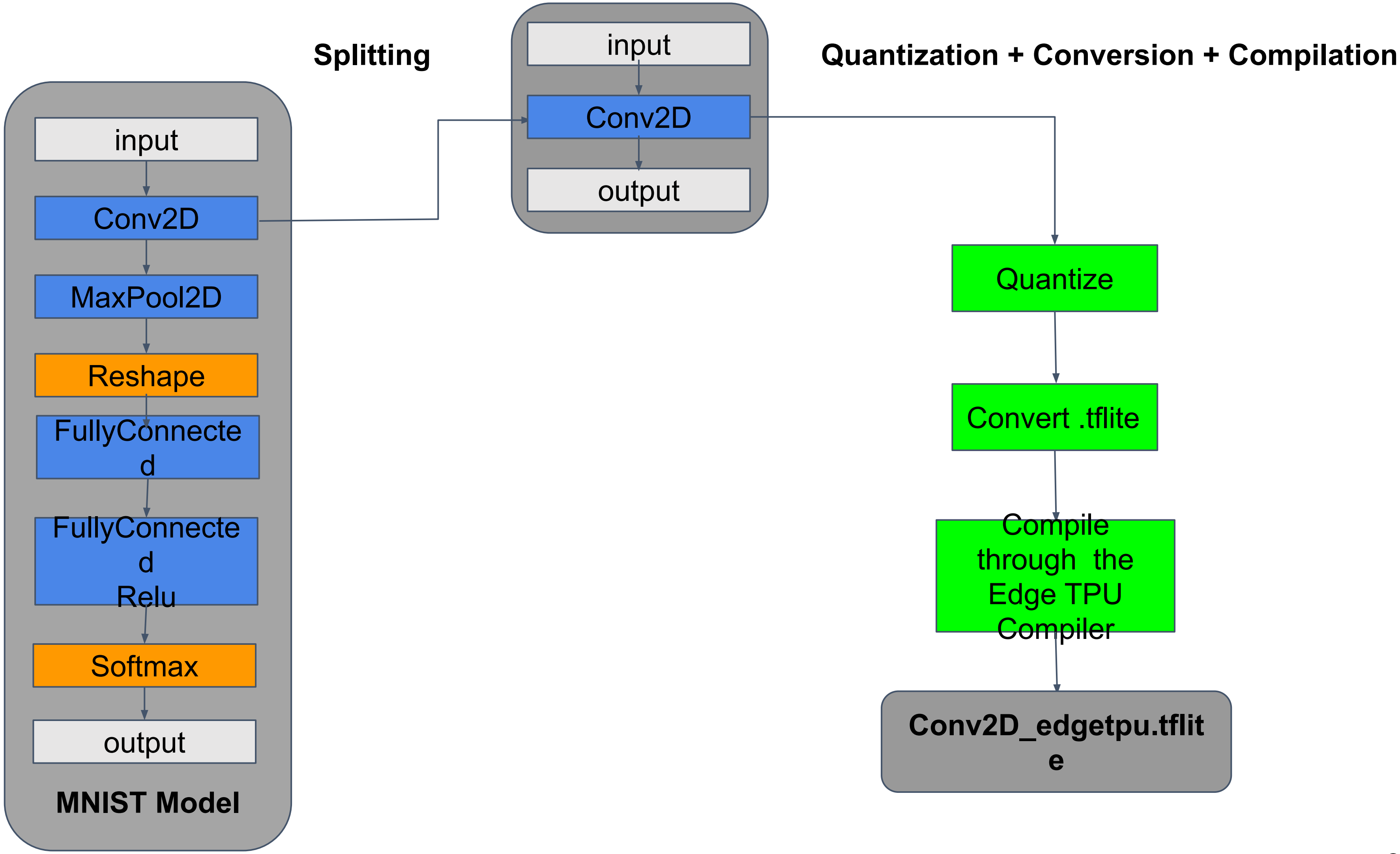
Motivation

- **Motivation**:
  - More granular or precise inference measurements, benchmark Google’s Coral Edge.

- **Problem Statement**:
  - Google’s Coral Edge USB Accelerator internal workings are closed source.
  - An analysis of the **USB traffic** occuring during inference is needed to obtain more precise results.

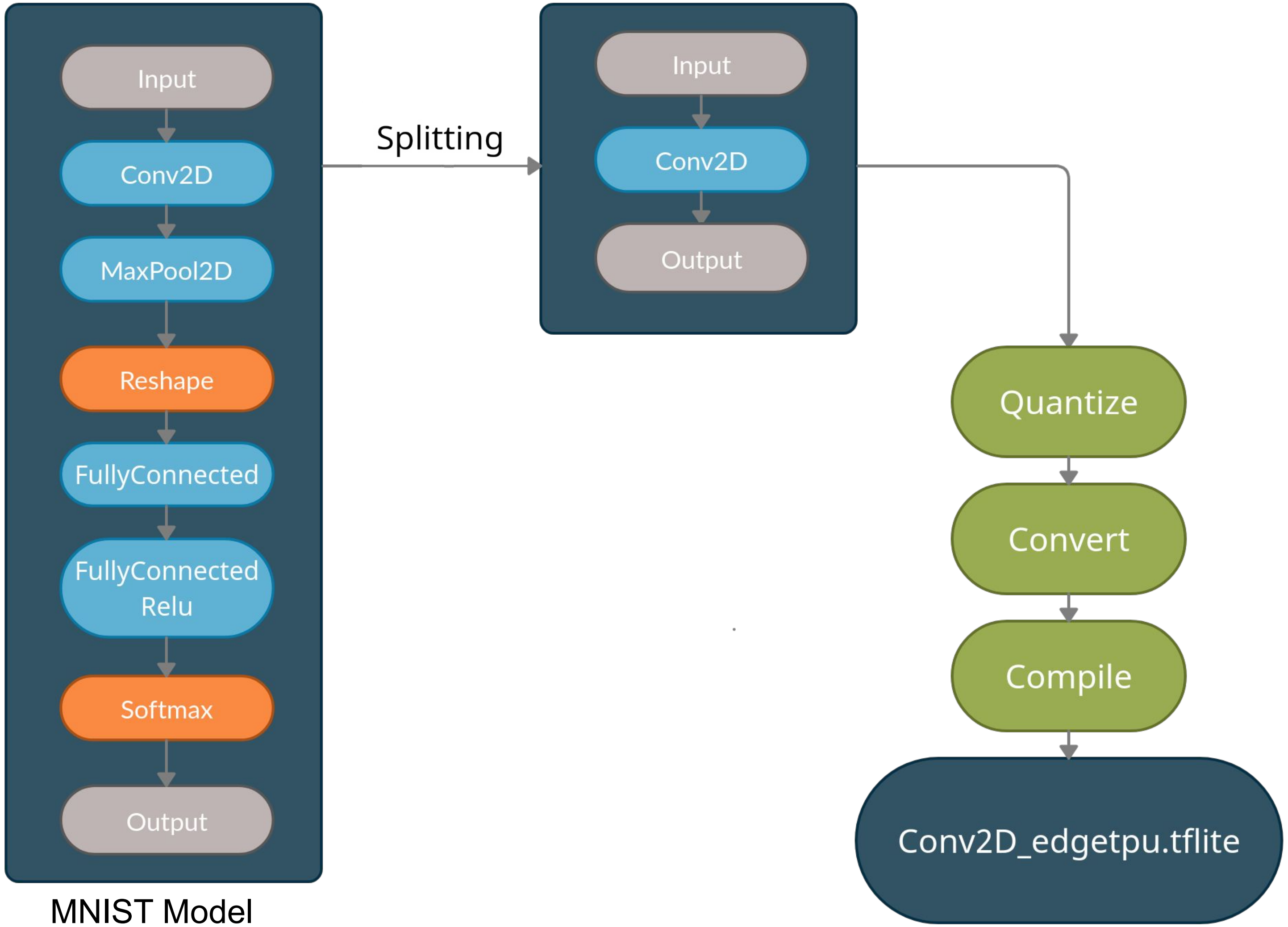
Background - Single Operation Splitting

- Flatbuffers
- Schema - Python Classes
- Single Layers



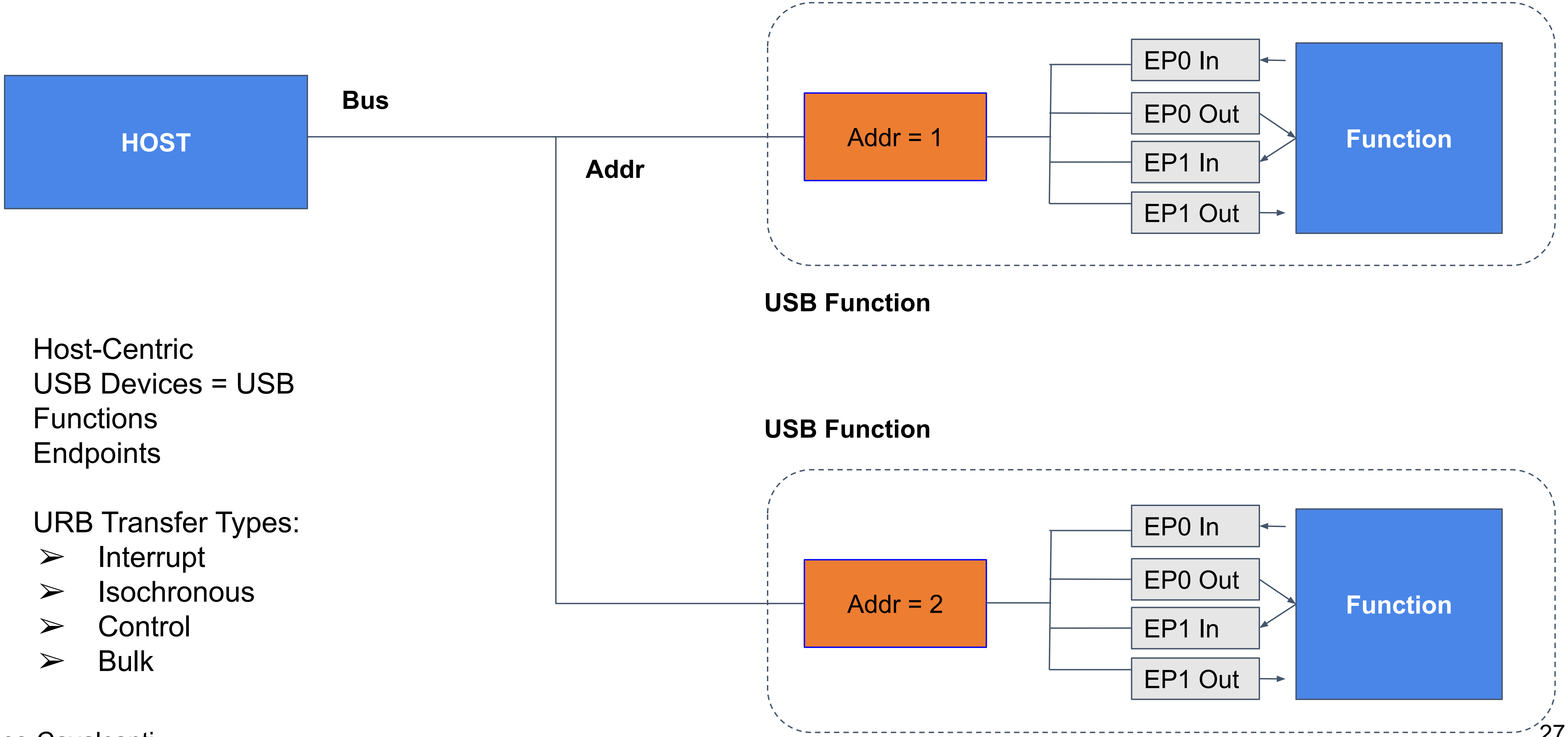
Background - Single Operation Splitting

- Split into Single Layers
- Flatbuffers
- Schema - Python Classes



MNIST Model

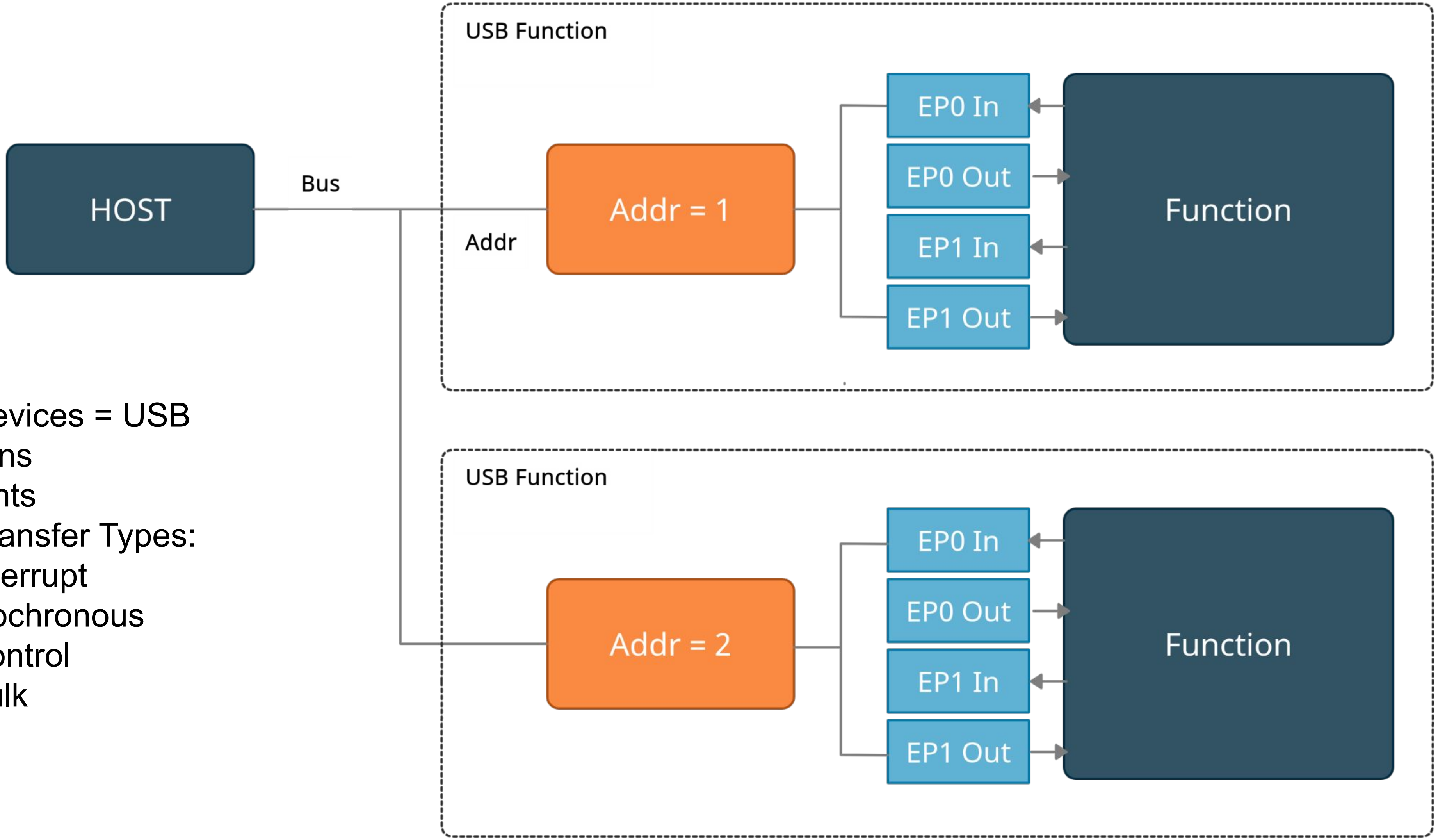
Background - USB Protocol



- Host-Centric
- USB Devices = USB Functions
- Endpoints

- URB Transfer Types:
  - Interrupt
  - Isochronous
  - Control
  - Bulk

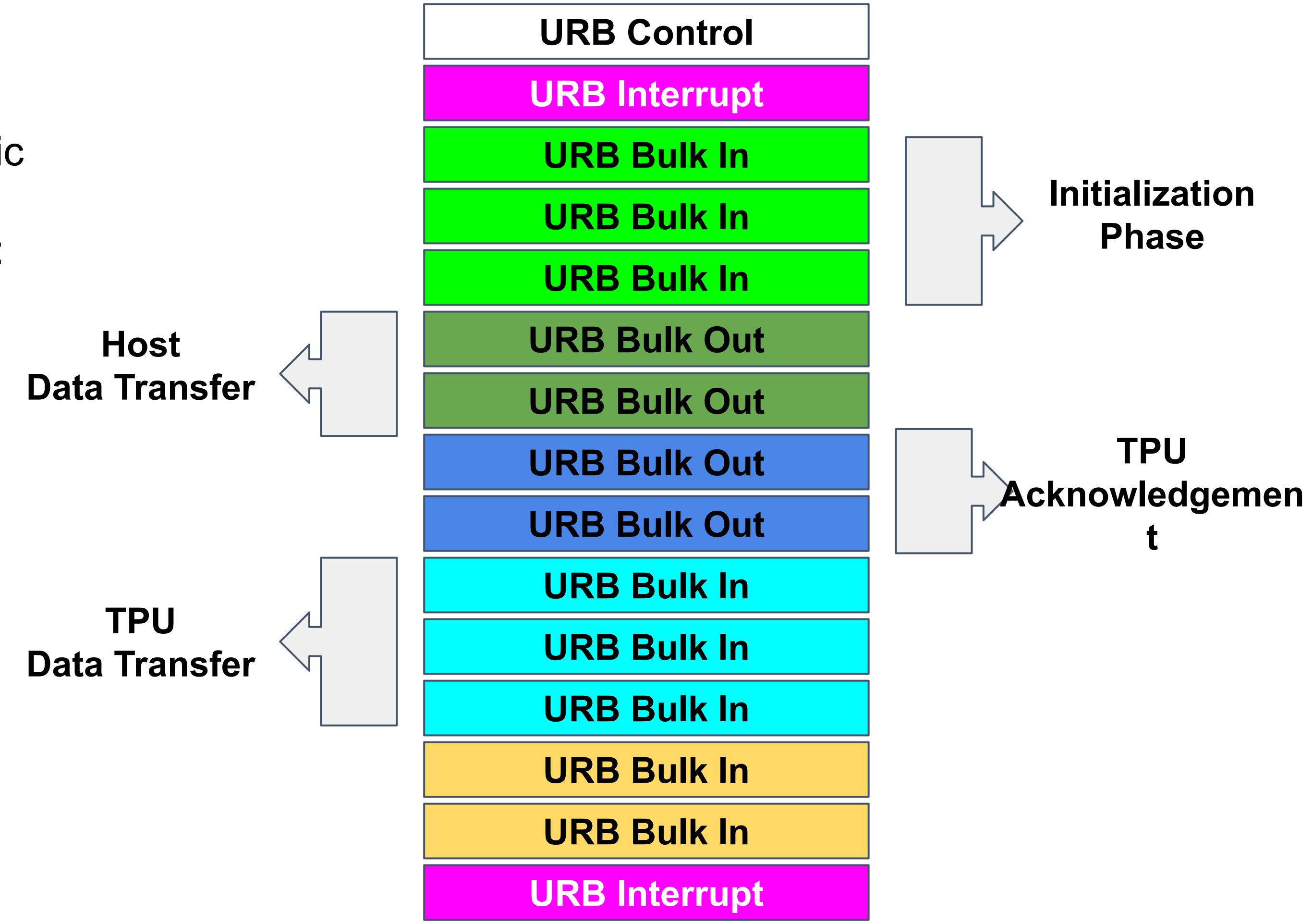
Background - USB Protocol



- USB Devices = USB Functions
- Endpoints
- URB Transfer Types:
  - Interrupt
  - Isochronous
  - Control
  - Bulk

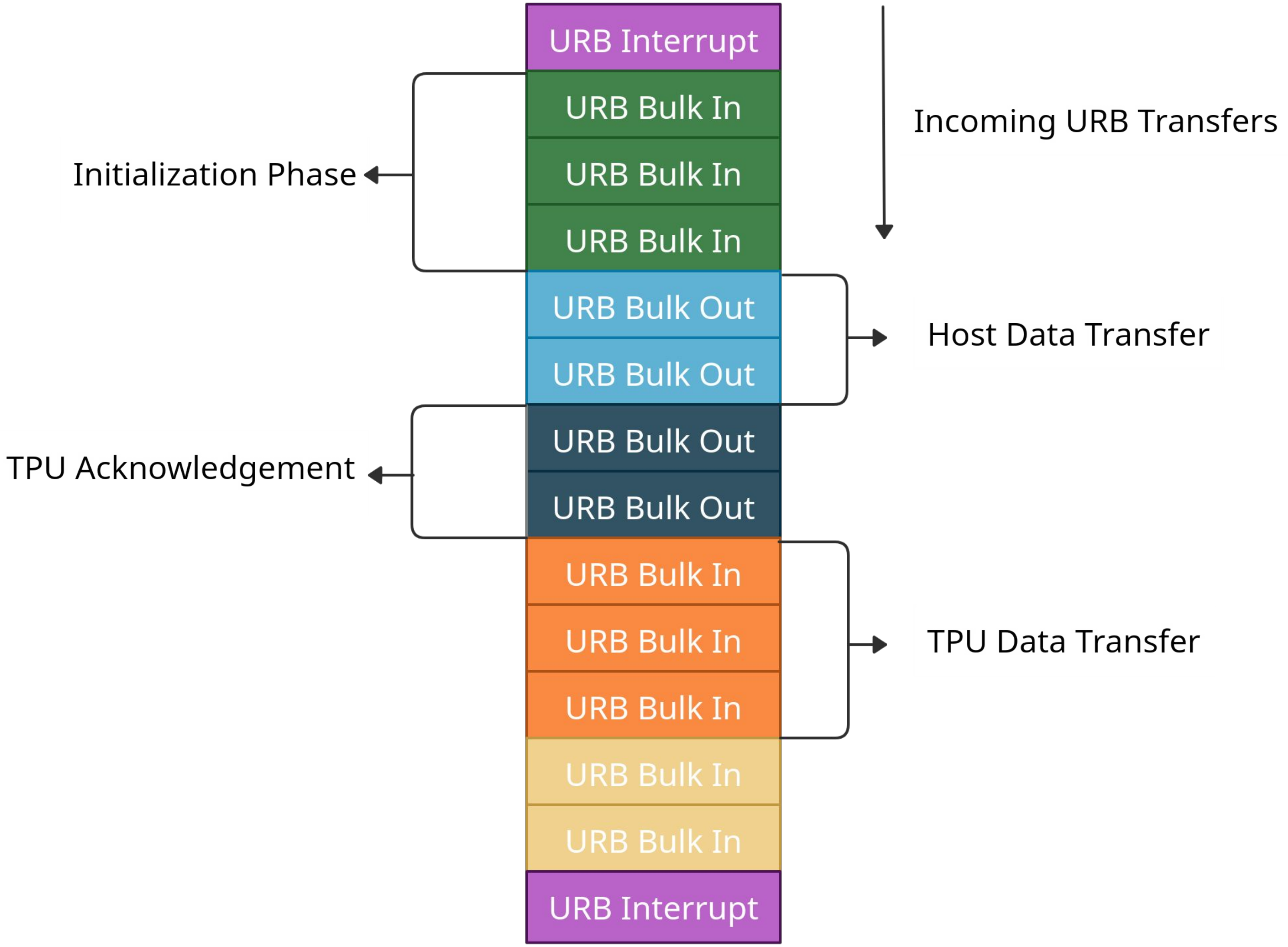
Background - USB Transactions

- Wireshark - captures USB traffic
- Main Communication Sections:
  - Initialization
  - Host-Data Transfer
  - TPU Acknowledgement
  - TPU-Data Transfer



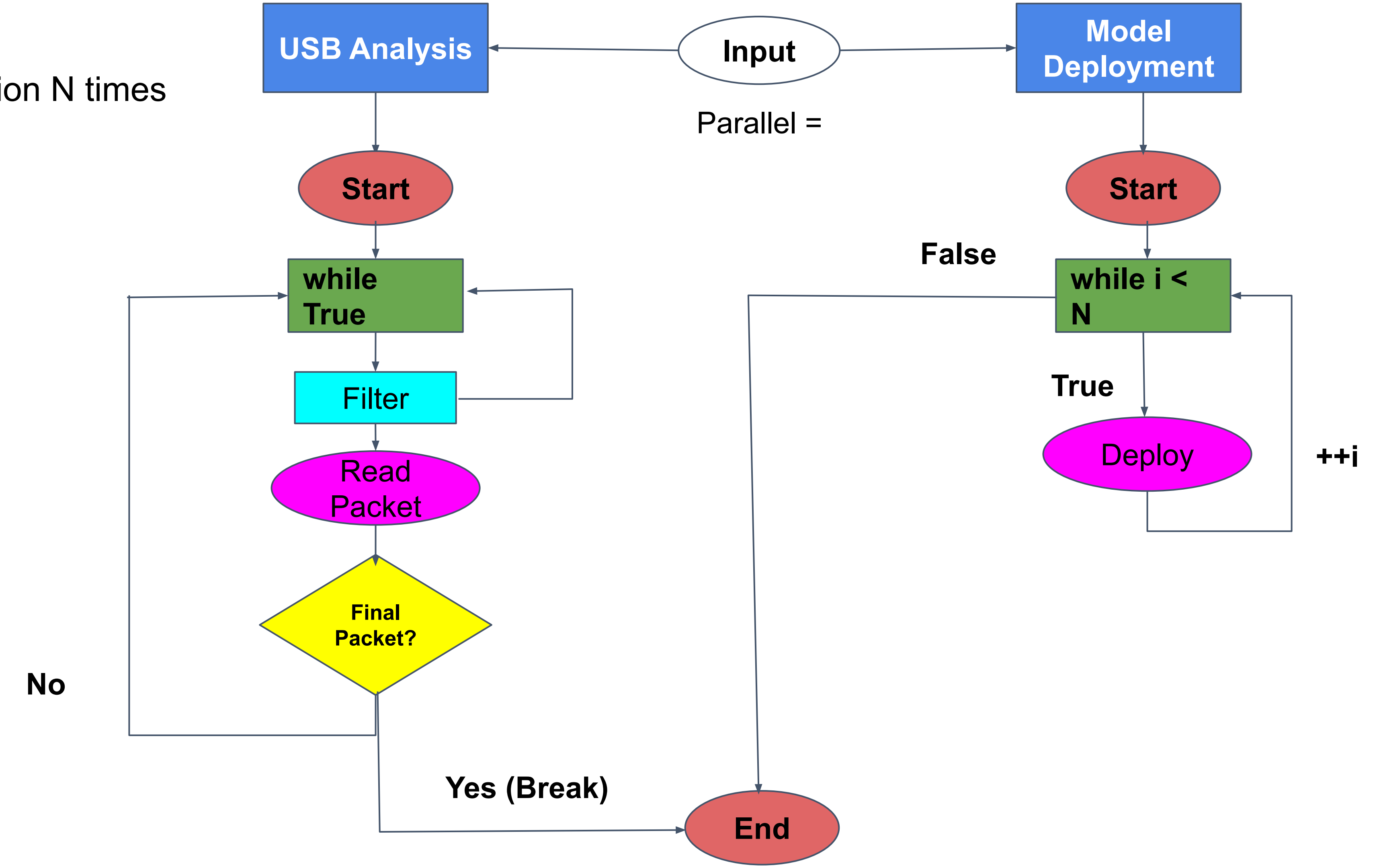
Background - USB Transactions

- Wireshark
  - Captures transfers
- Deployment
  - Interpreter Object
  - Delegates



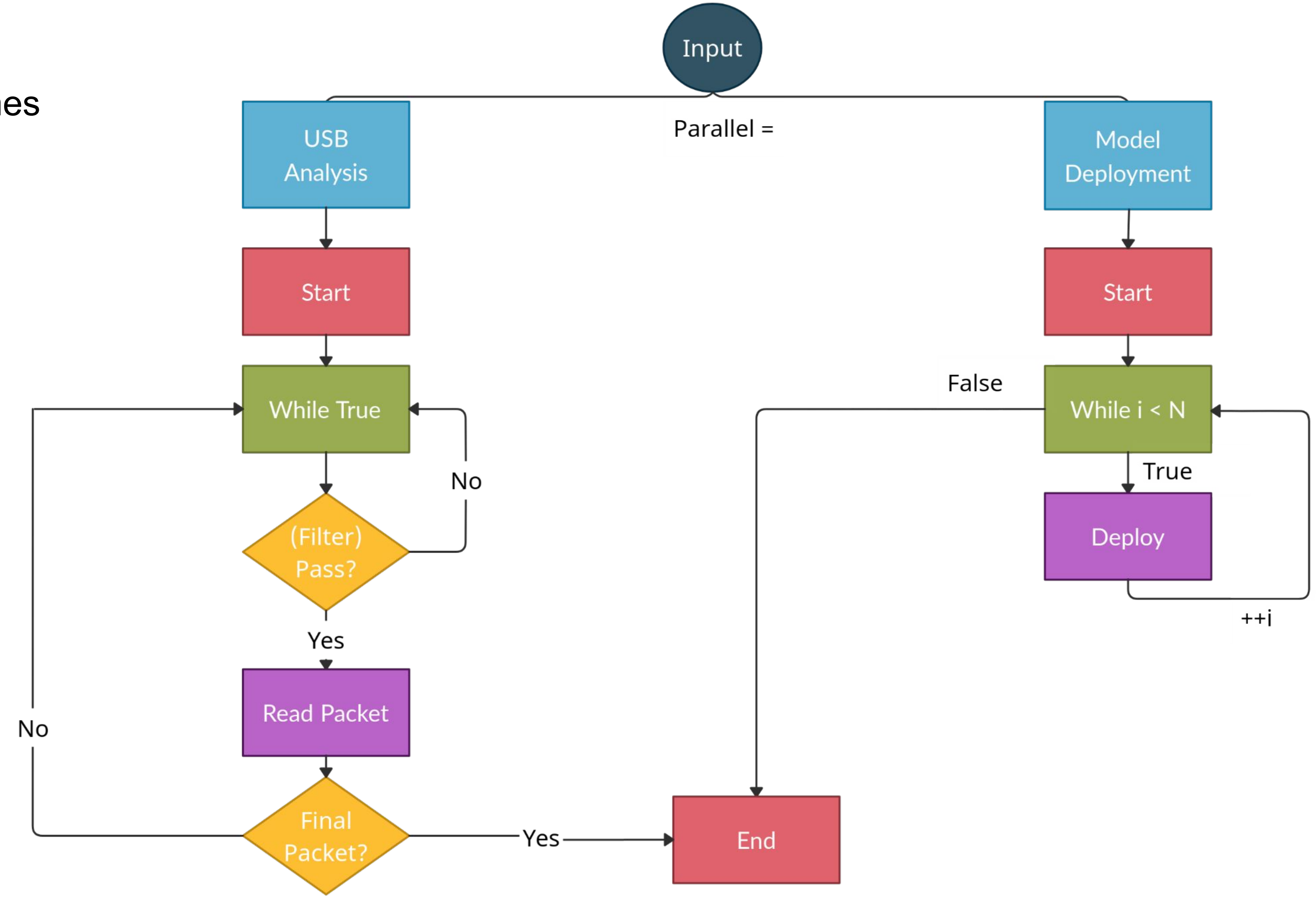
Implementation - USB Packet Analysis

- Parallel Execution N times
- Pyshark



Implementation - USB Packet Analysis

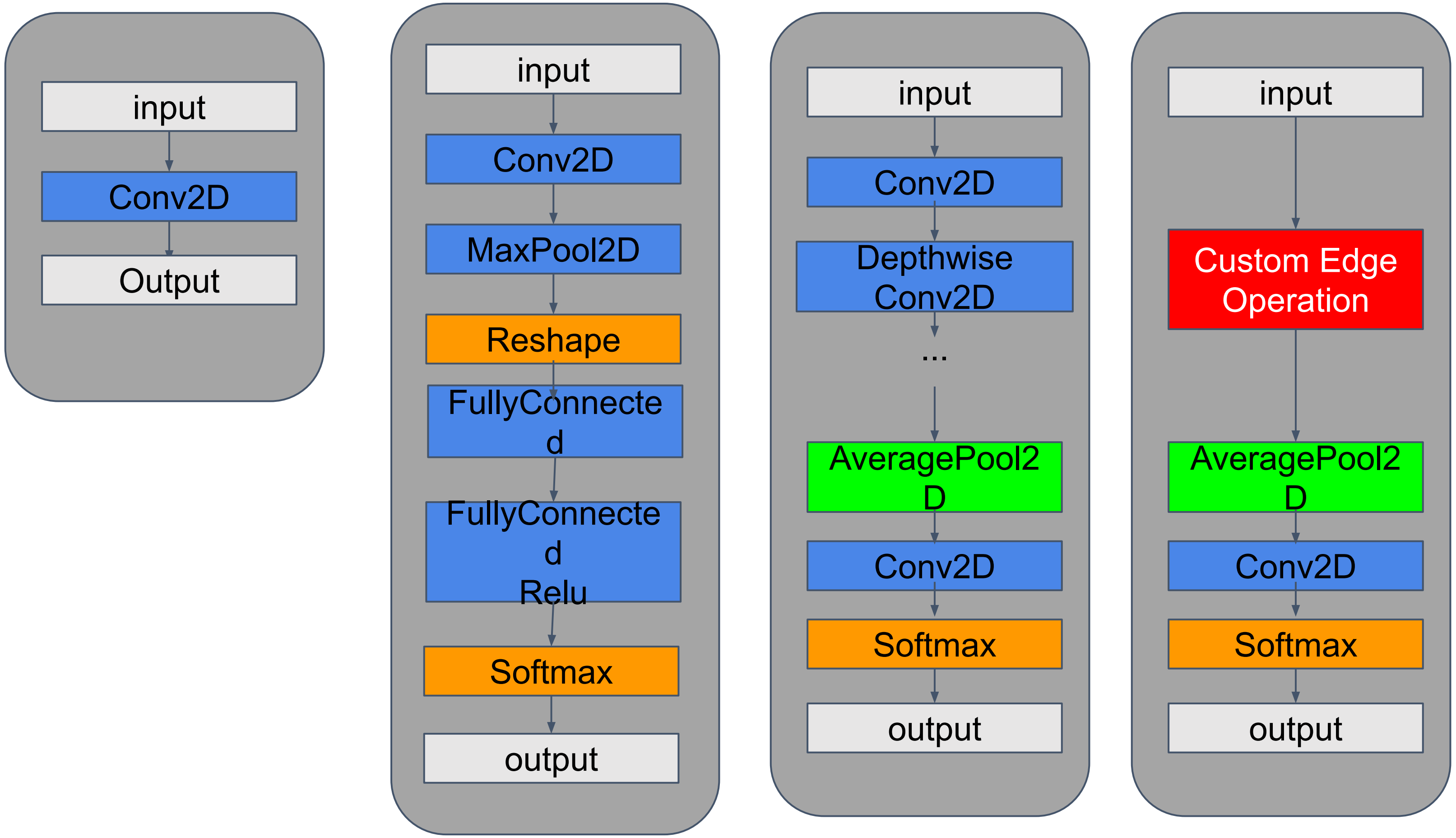
- ❖ Parallel Execution N times
- ❖ Pyshark



Implementation - Test Parameters

- Different Models:
  - MNIST
  - MNIST Layers
  - Mobilenet
  - Mapped Mobilenet

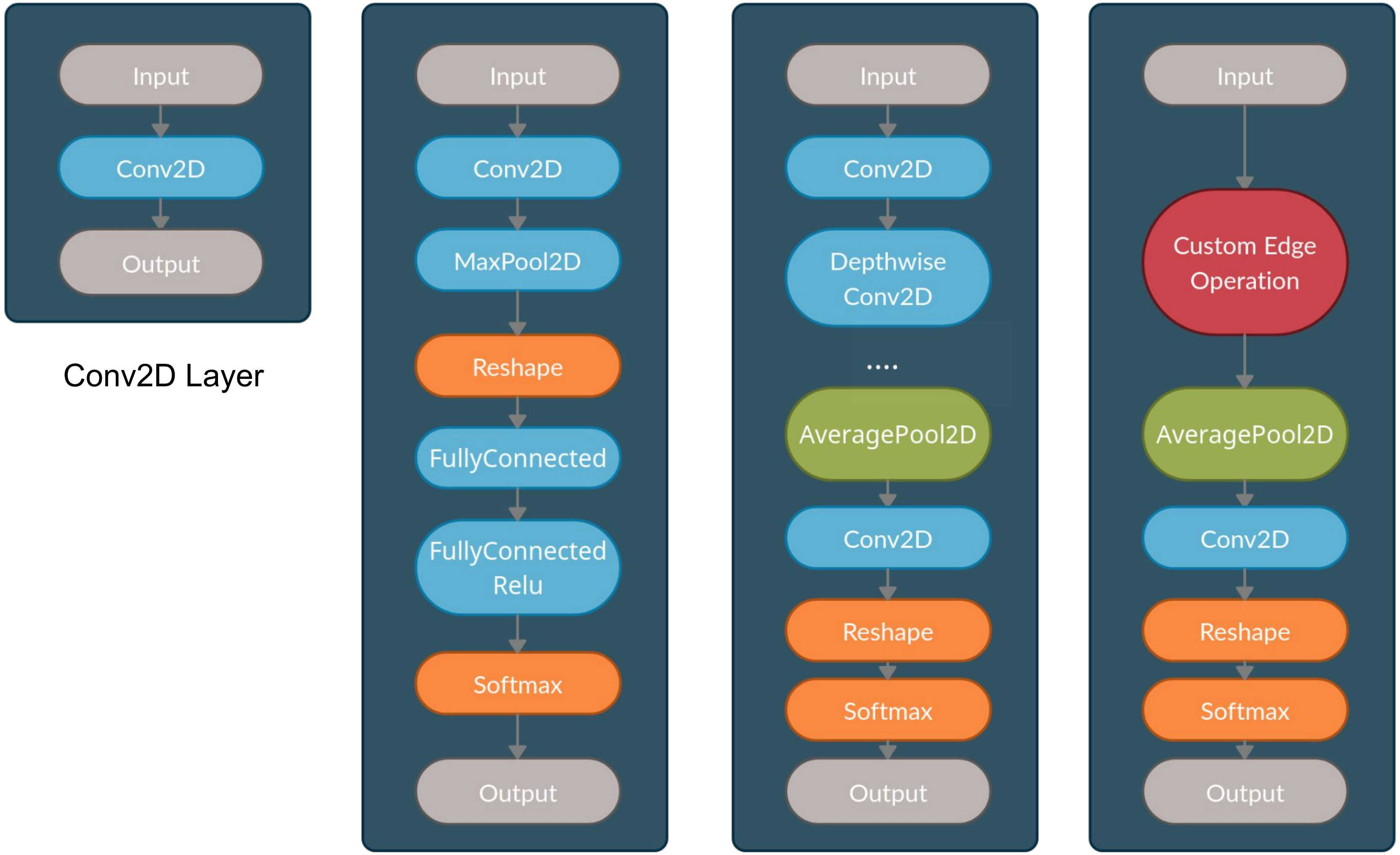
- Data Size - Complexity



Conv2D Layer MNIST Model Mobilenet Mapped Mobilenet

Implementation - Test Parameters

- Different Models:
  - MNIST Layers
  - MNIST
  - Mobilenet
  - Mapped Mobilenet
- Data Size - Complexity



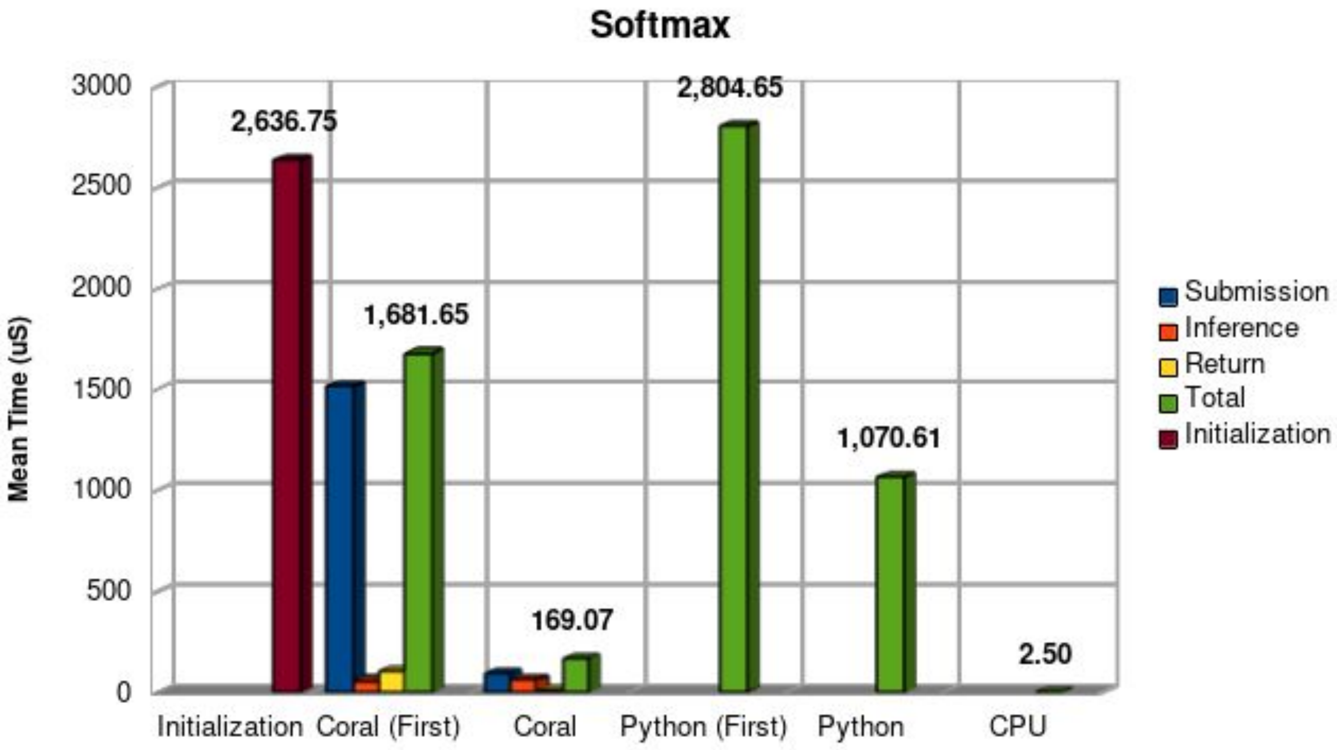
Conv2D Layer

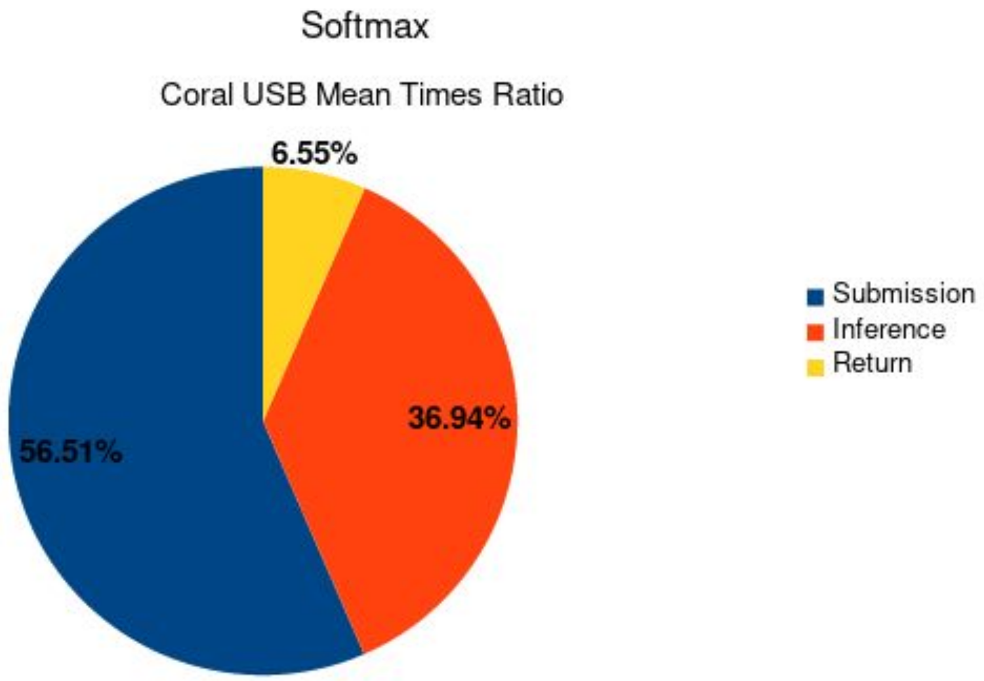
MNIST

Mobilenet

Mobilenet Mapped

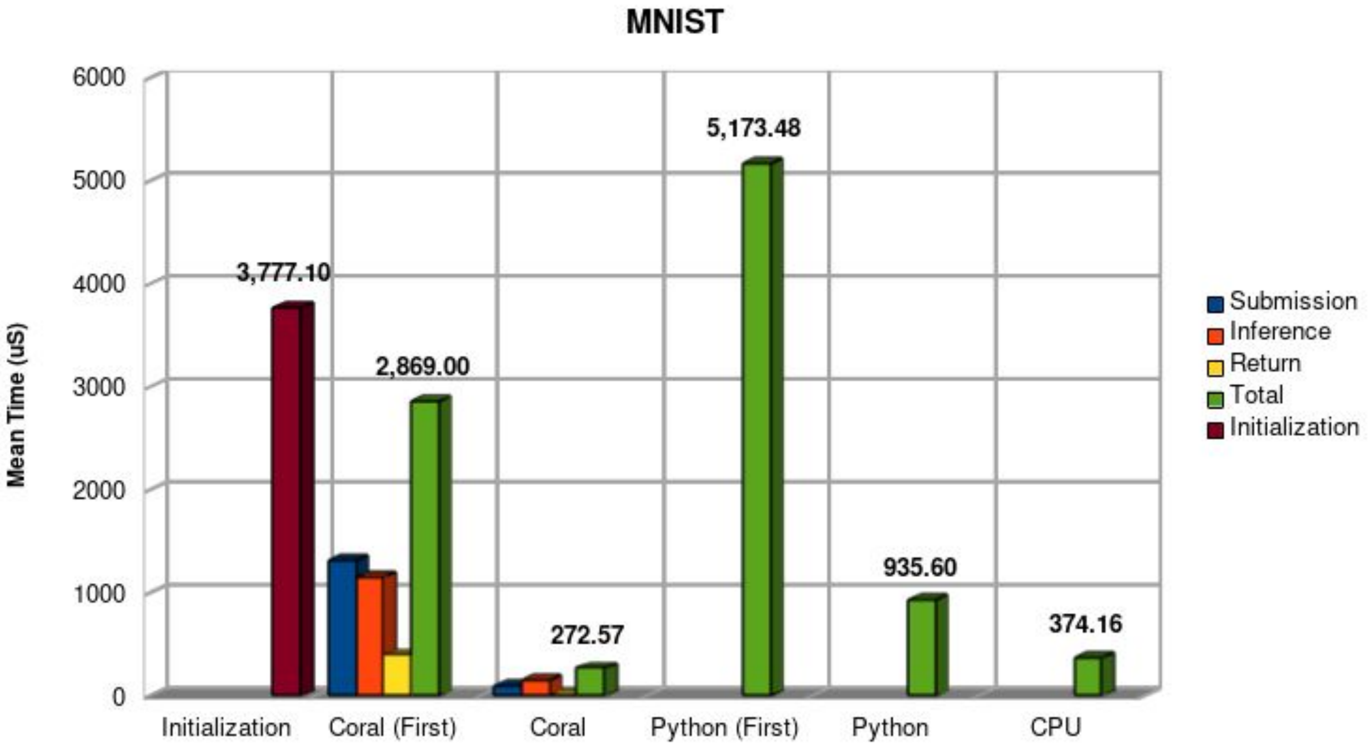
Results - Softmax Layer

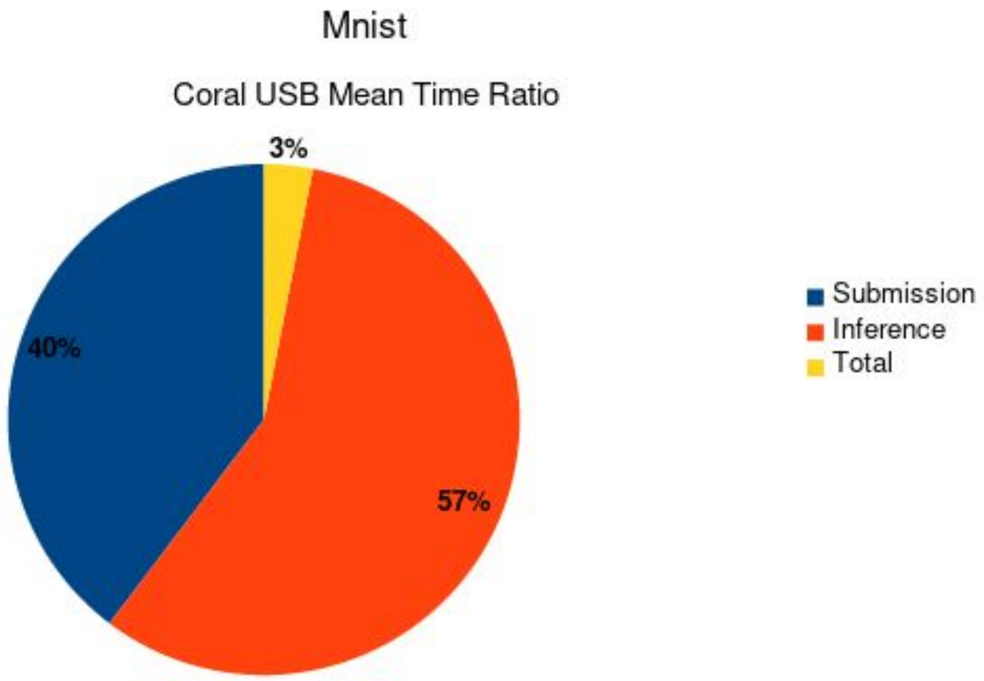




- Significant submission average within ratio
- Expressive difference in Python and USB Total times - 15.8%

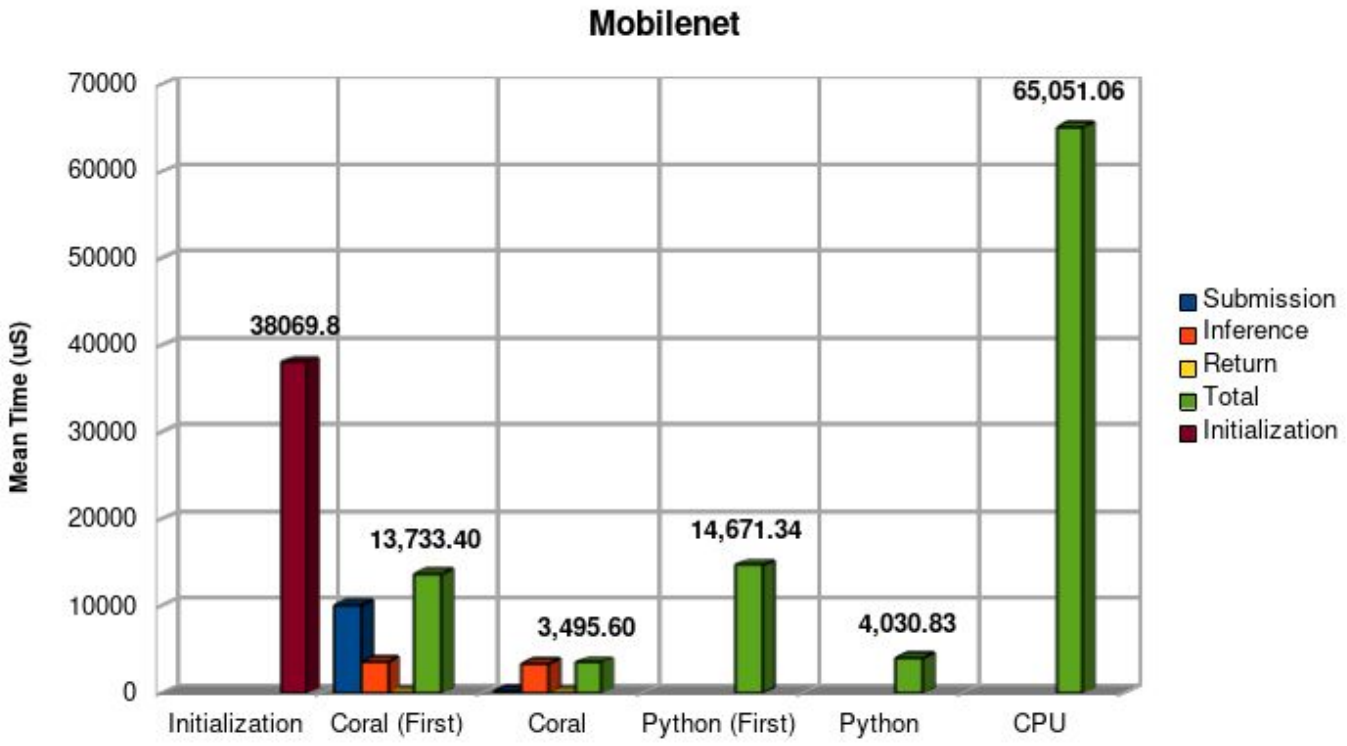
Results - MNIST Model

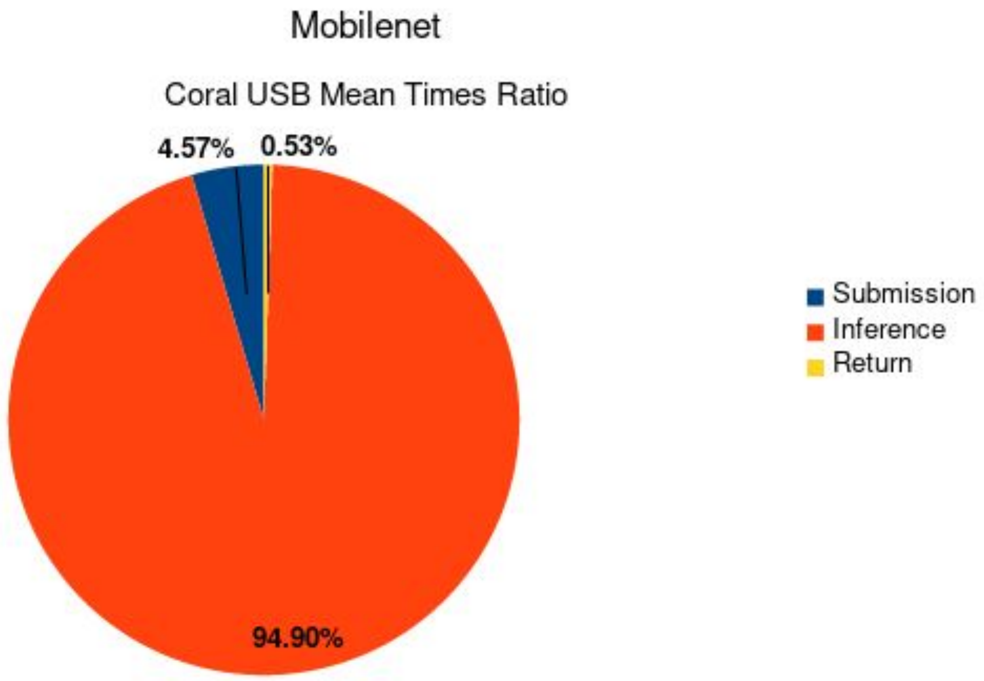




- Inference Ratio increases
- Less of a deviation between USB and Python total times - 29.1%

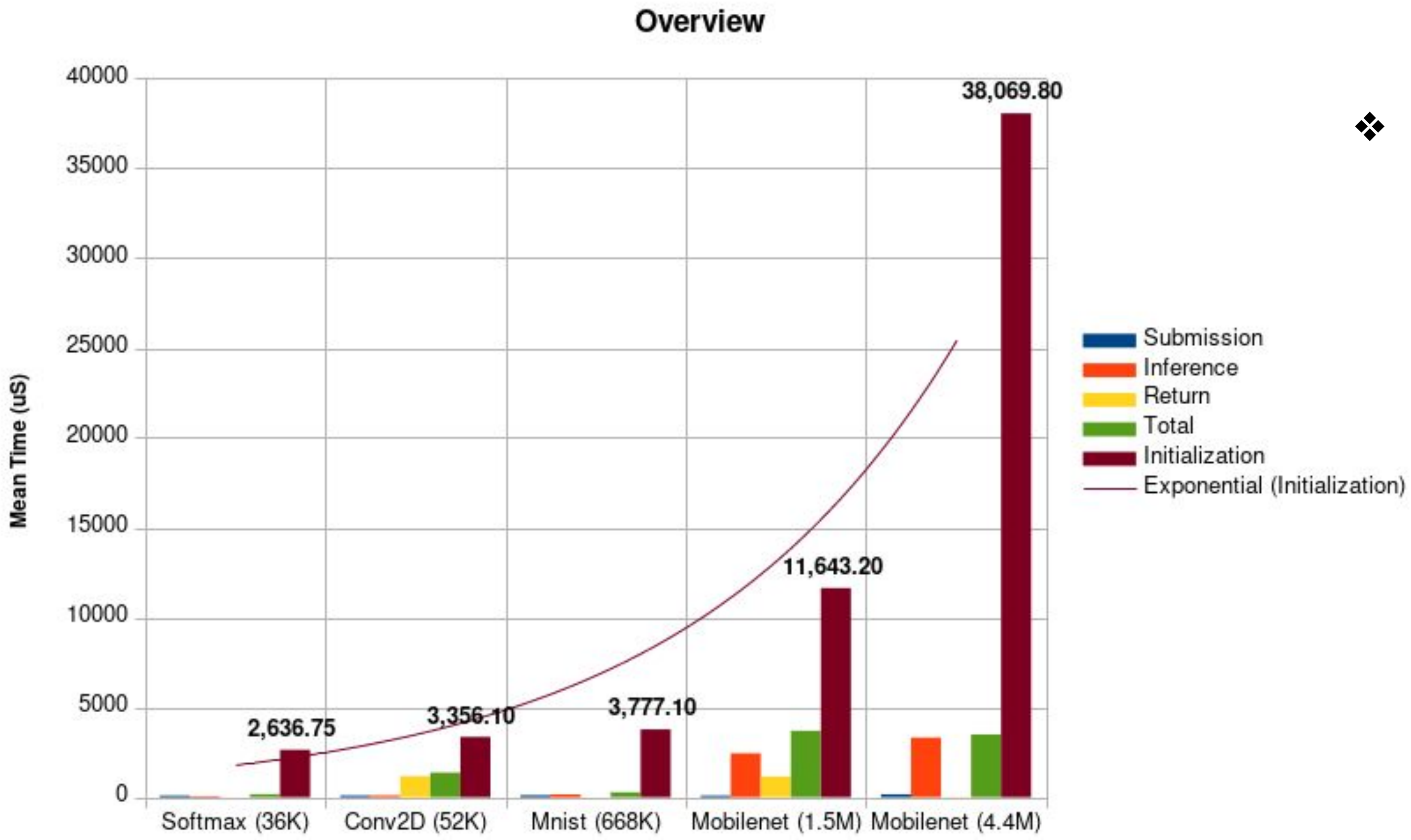
Results - Mobilenet Model





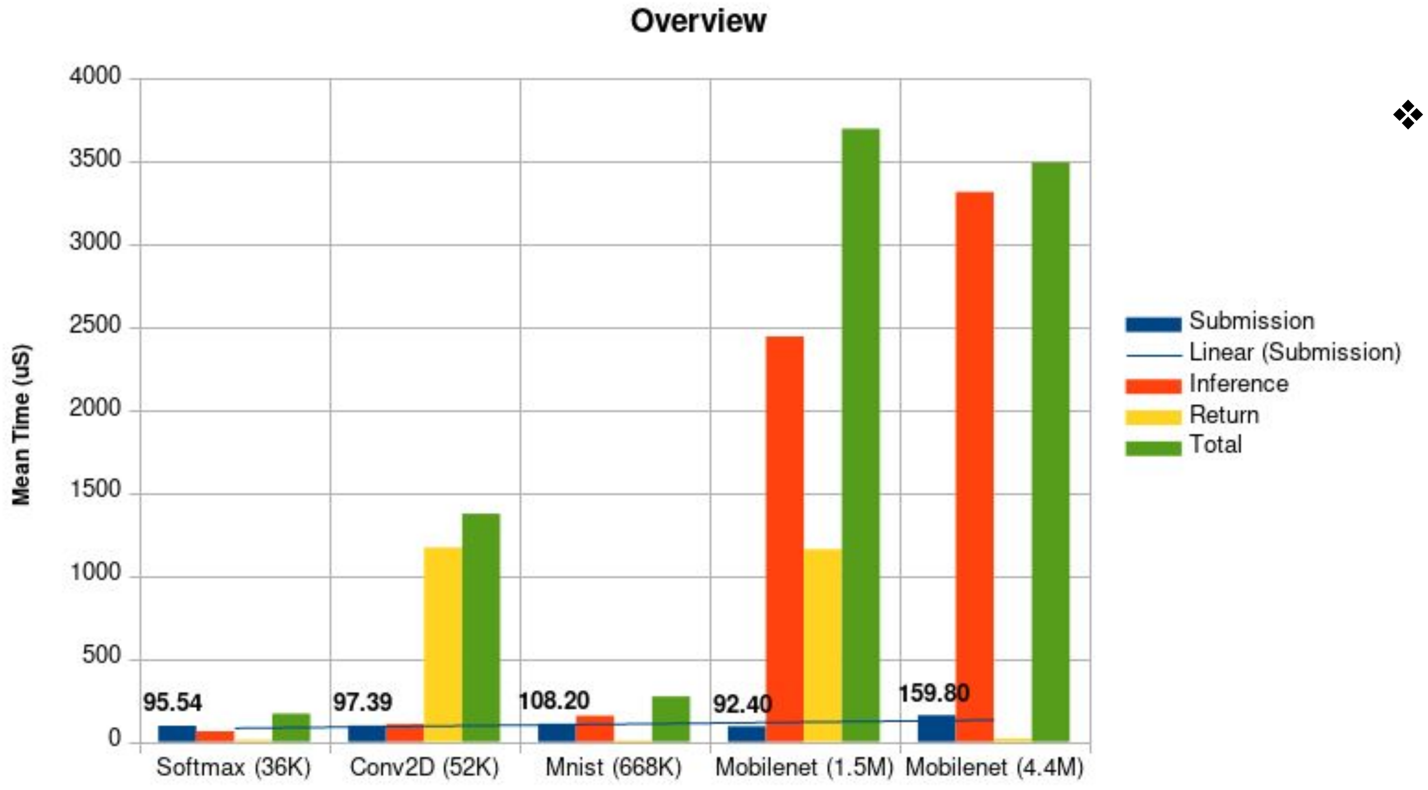
- ❖ Drastic increase in Inference Ratio
- ❖ Python and USB total times deviate only by 13.3%
- ❖ Significant difference in Submission times

Results - Overview



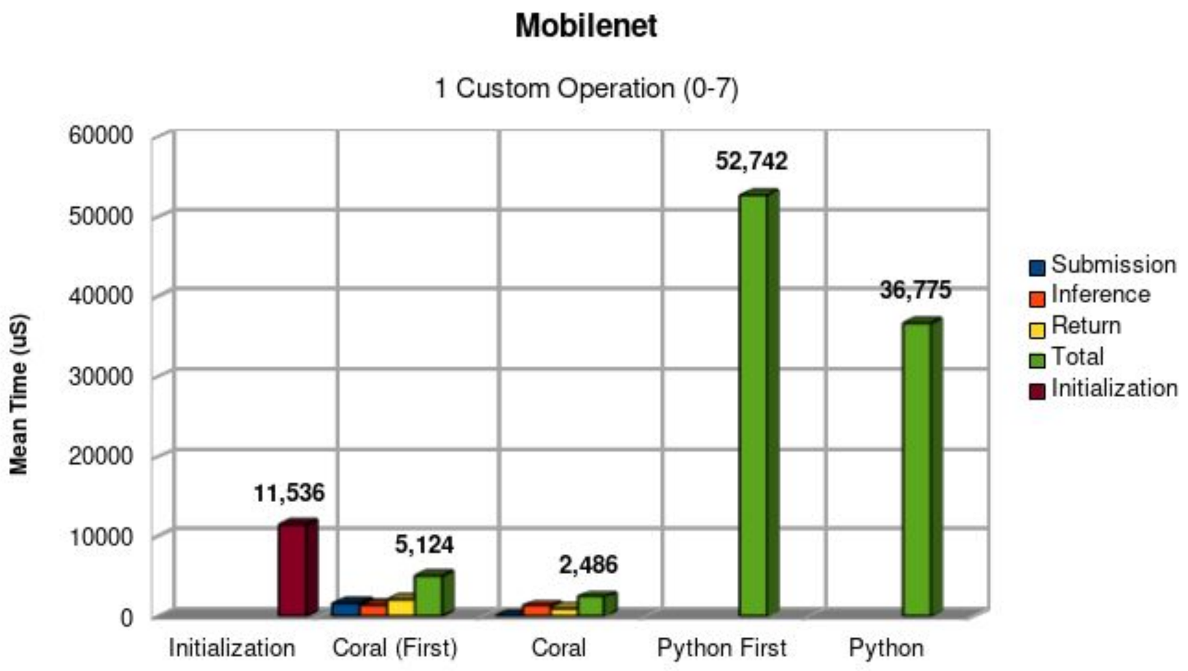
- ❖ Non-linear increase in Initialization Phase

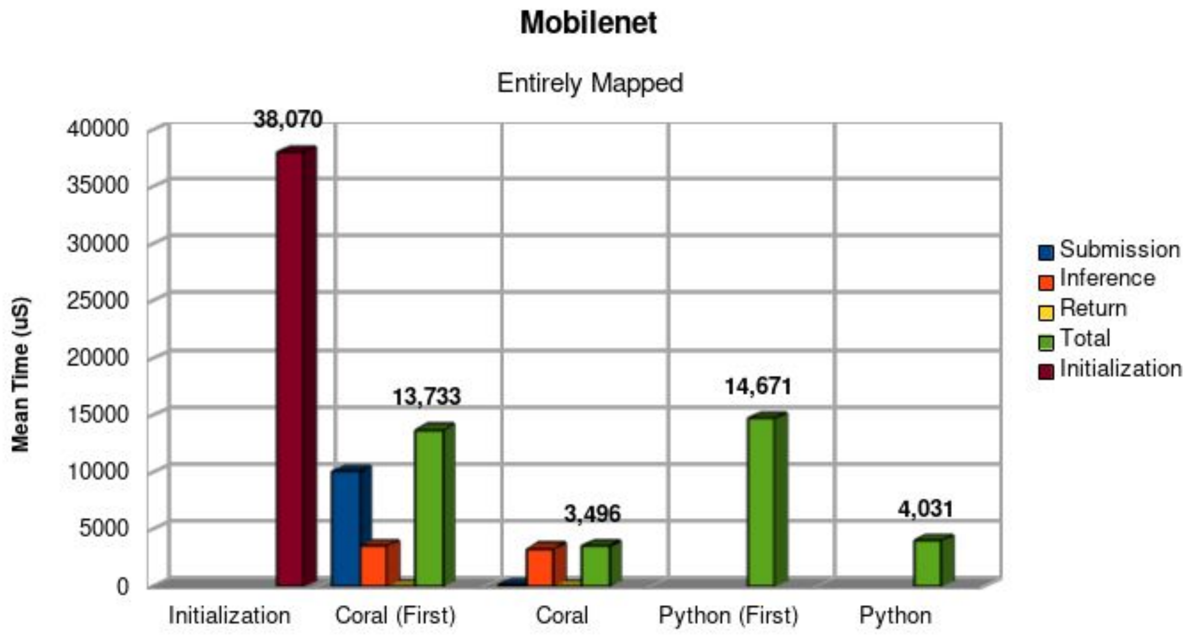
Results - Overview



- ❖ Linear increase in Submission time

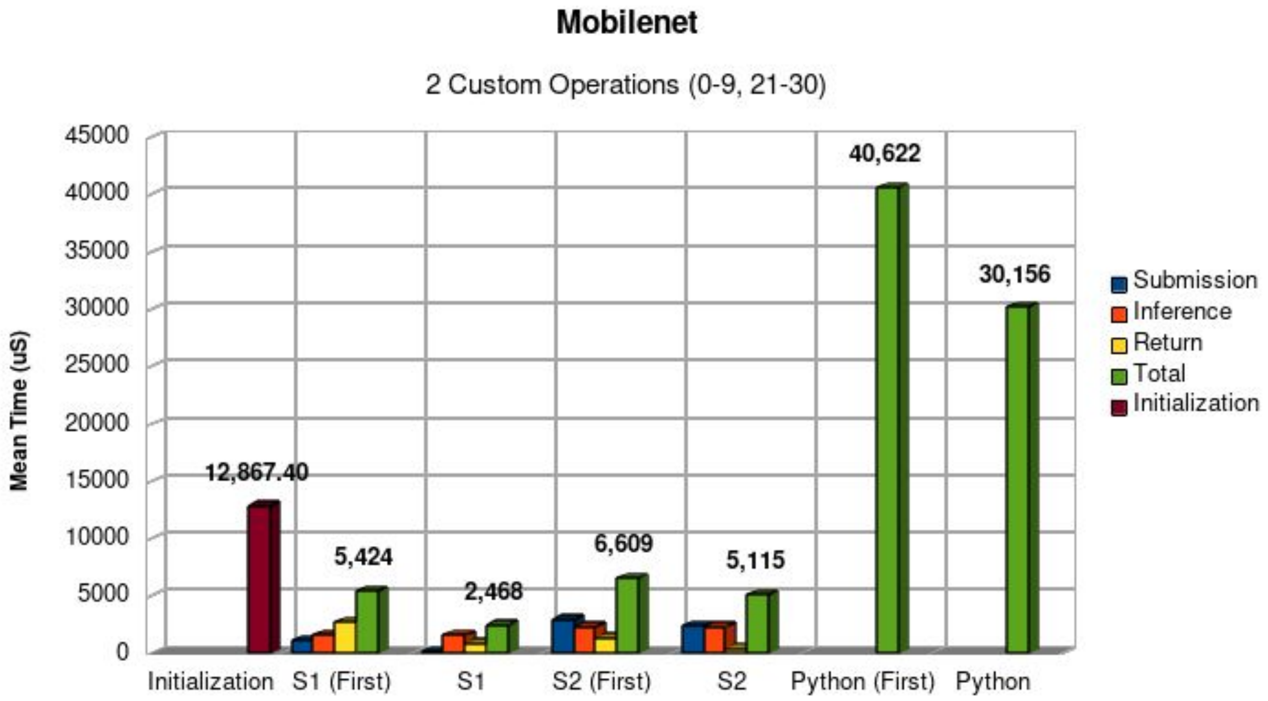
Results - Mapped Mobilenet Model

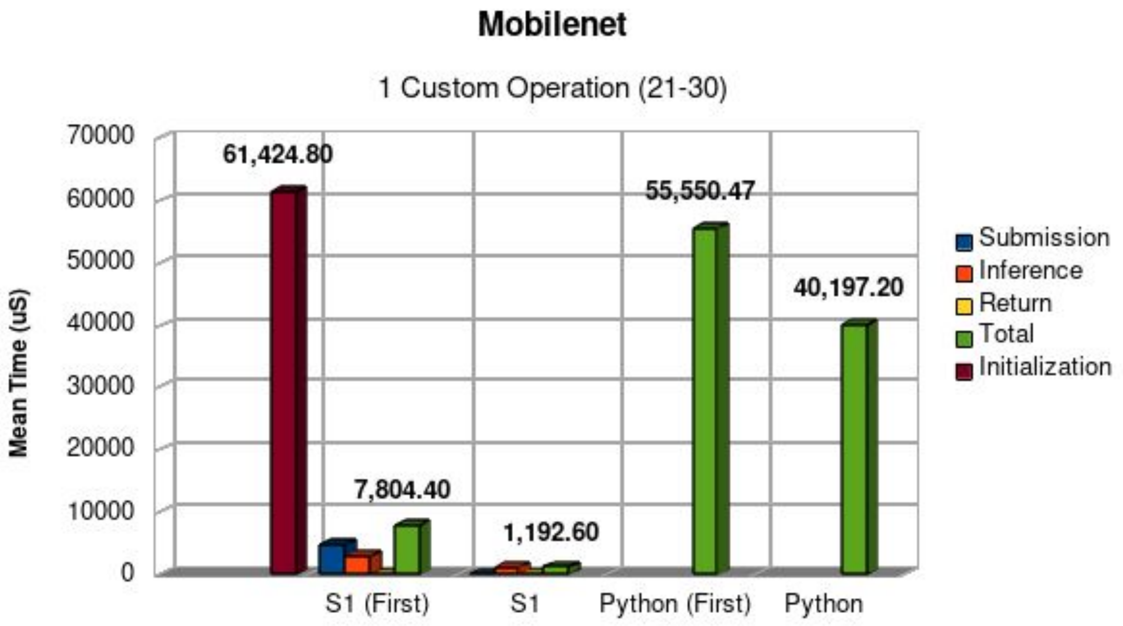




- More operations mapped to coral edge => faster performance

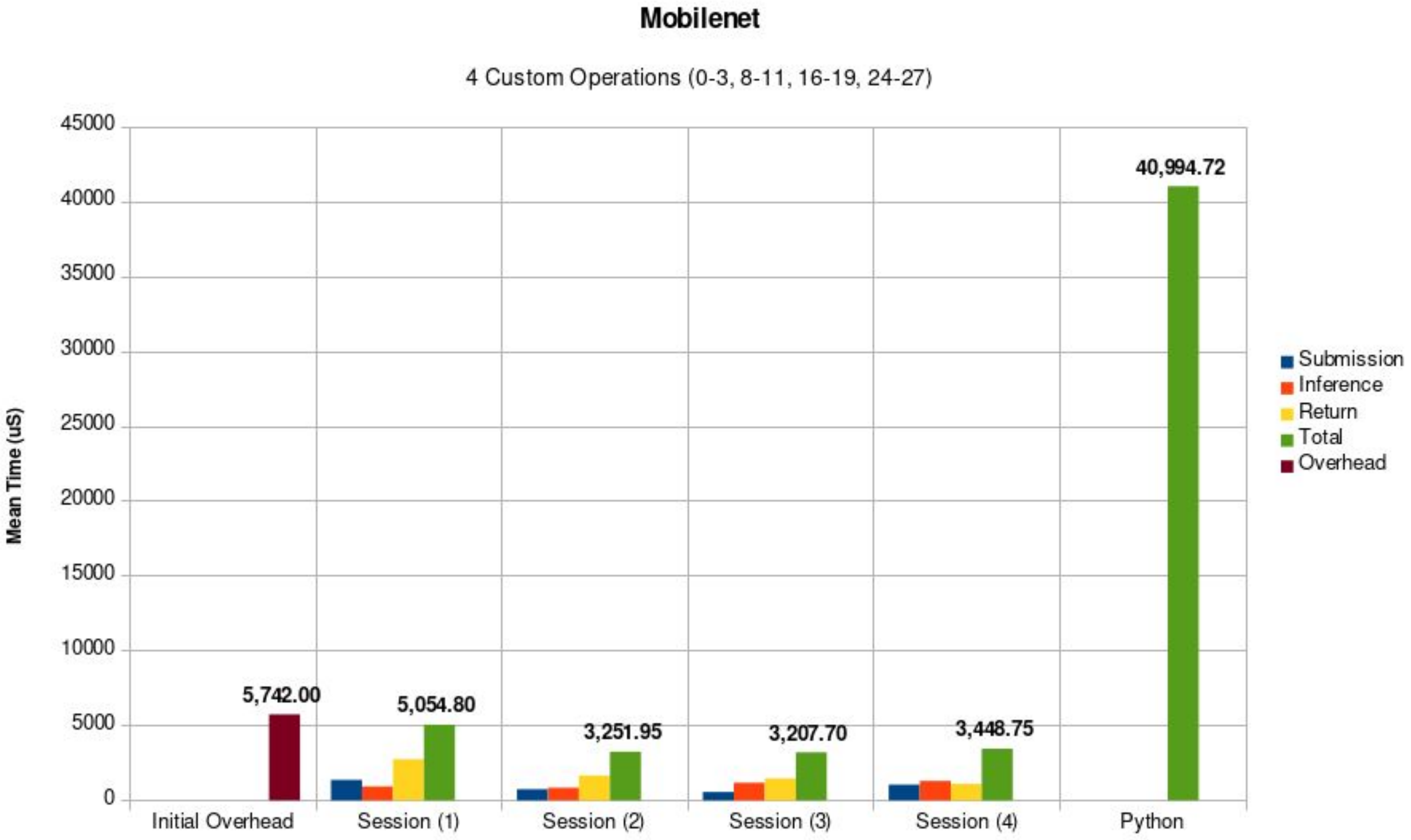
Results - Mapped Mobilenet Model





- ❖ Subsequent Operations show slower deployment.

Results - Mapped Mobilenet Model



Conclusions and Future Work

- Conclusions:
  - Single layer/less complex models show a larger deviation of total deployment time.
  - More Complex models show a significantly smaller deviation of total deployment time.
  - Submission times of data are proven to be linear with an increase in model complexity.
  - Initialization phase occurs once and has a non-linear growth with an increase in model complexity.
  - Subsequent mapped operations perform slower.
- Future Work
  - Test larger spectrum of Models with a more granular difference in complexity and larger number of mappings.

Questions