RAMON.space

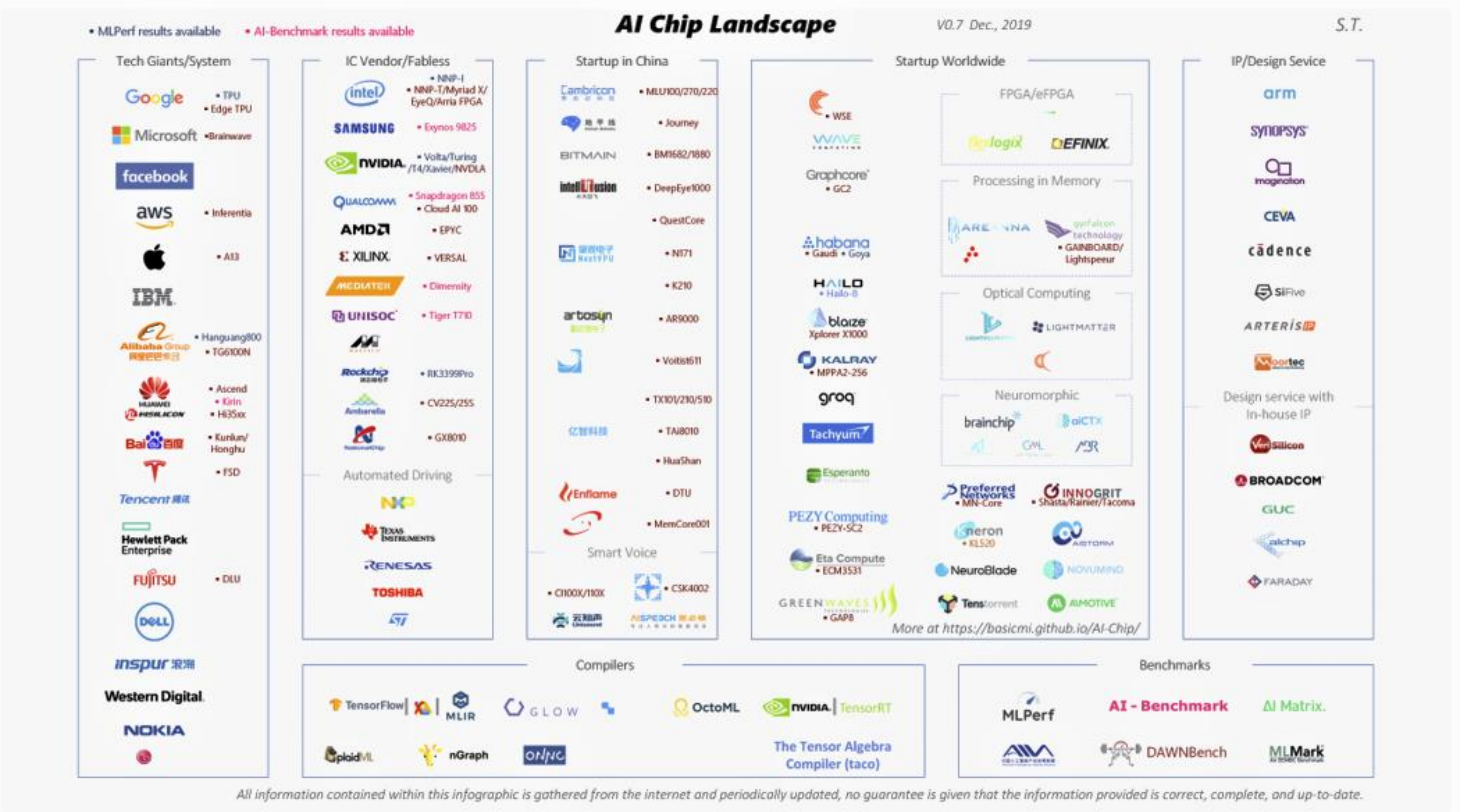
Ramon.Space is named in memory of Col. Ilan Ramon, Israeli astronaut who died on board the Columbia space shuttle, Feb. 1, 2003

Ramon Space RC64-based AI/ML Inference Engine

Session 6: AI Inference Frameworks and Acceleration on Space Devices
Tuesday, Jun 15, 2021, 5:35 PM - 5:55 PM

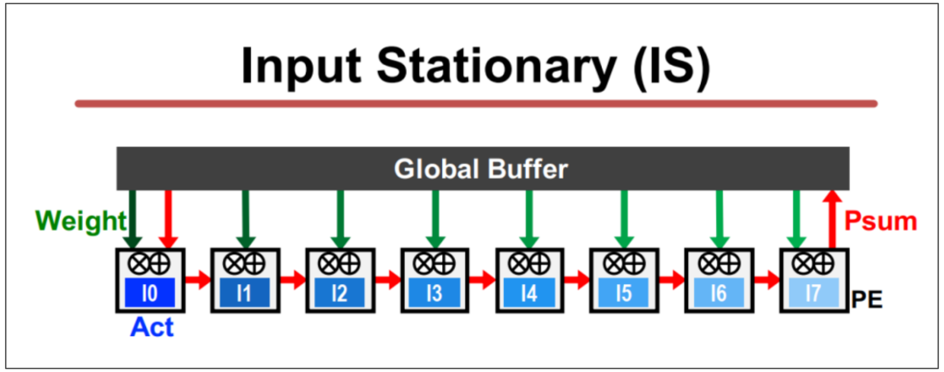
Ran Ginosar
David Goldfeld
Peleg Aviely
Roei Golan
Avraham Meir
Fredy Lange
Dov Alon
Tuvia Liran
Avi Shabtai

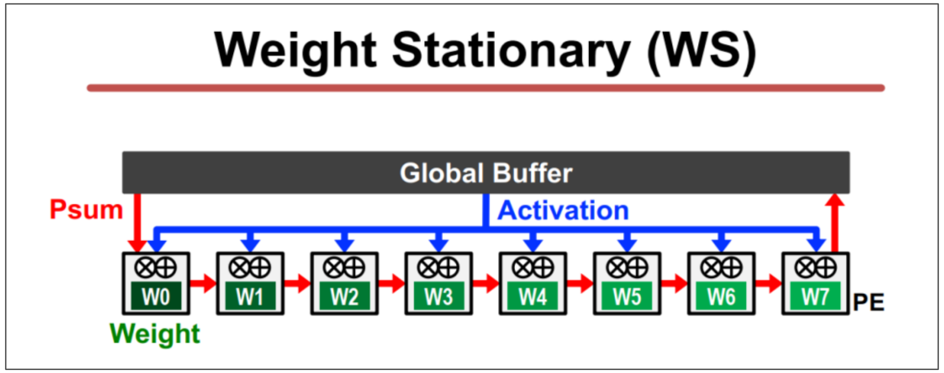
AI/ML Accelerators are Plenty

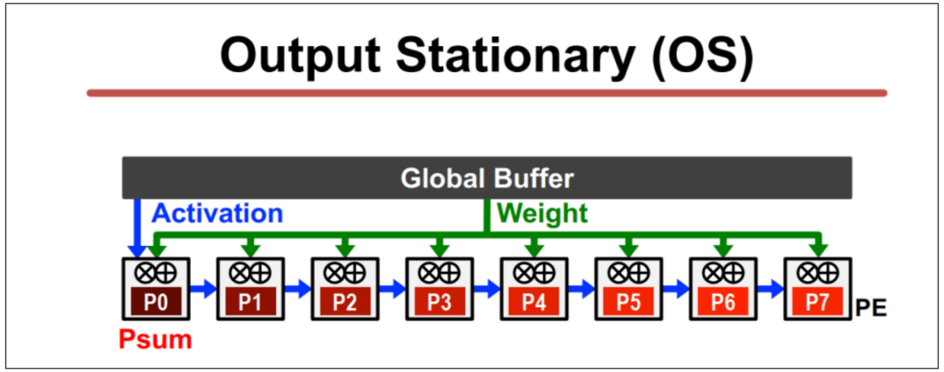


AI/ML Accelerators are Great

- Training and/or Inference
- Many different architectures
- High level interfaces
- Power-efficient, or high-performance
- Inference / edge computing:
  - Low power
  - Small models
  - Small data







https://www.rle.mit.edu/eems/wp-content/uploads/2019/06/Tutorial-on-DNN-05-DNN-Accelerator-Architectures.pdf

AI/ML Accelerators are NOT FOR SPACE

- Space is
  - Long term: 10—30 years
  - High TID: 100—300 kRad
  - High Latchup: SEL_{TH} LET > 70 MeV·cm^2/mg
  - Wide temperature span: −40°C to +100°C and higher
  - Many temperature cycles: 1,000,000 and more

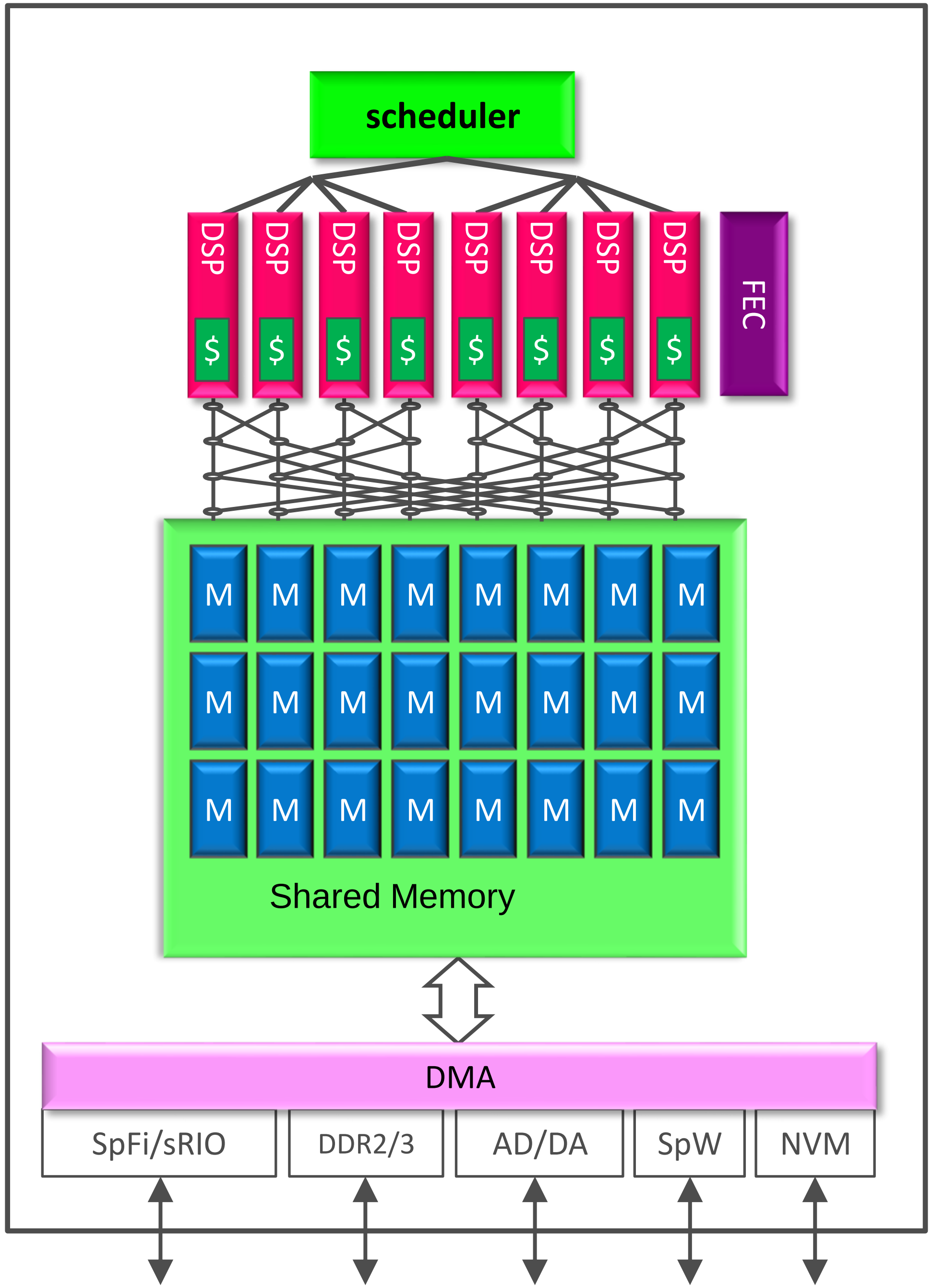
- Ramon Space RC64 is designed for high-end SPACE Machine Learning

Design Objectives: RC64 as a ML processor

- SPACE
- High performance-to-power ratio
  - Not simply high performance
  - Not simply low power
- Serves all/most ML models
  - Including not-yet-developed models (the field changes fast)
- Serves small & large models
- Serves small & big data
- No programming needed
- Compatible with ground ML
  - Develop/train/re-train on the ground
  - Repeatedly upload to SPACE

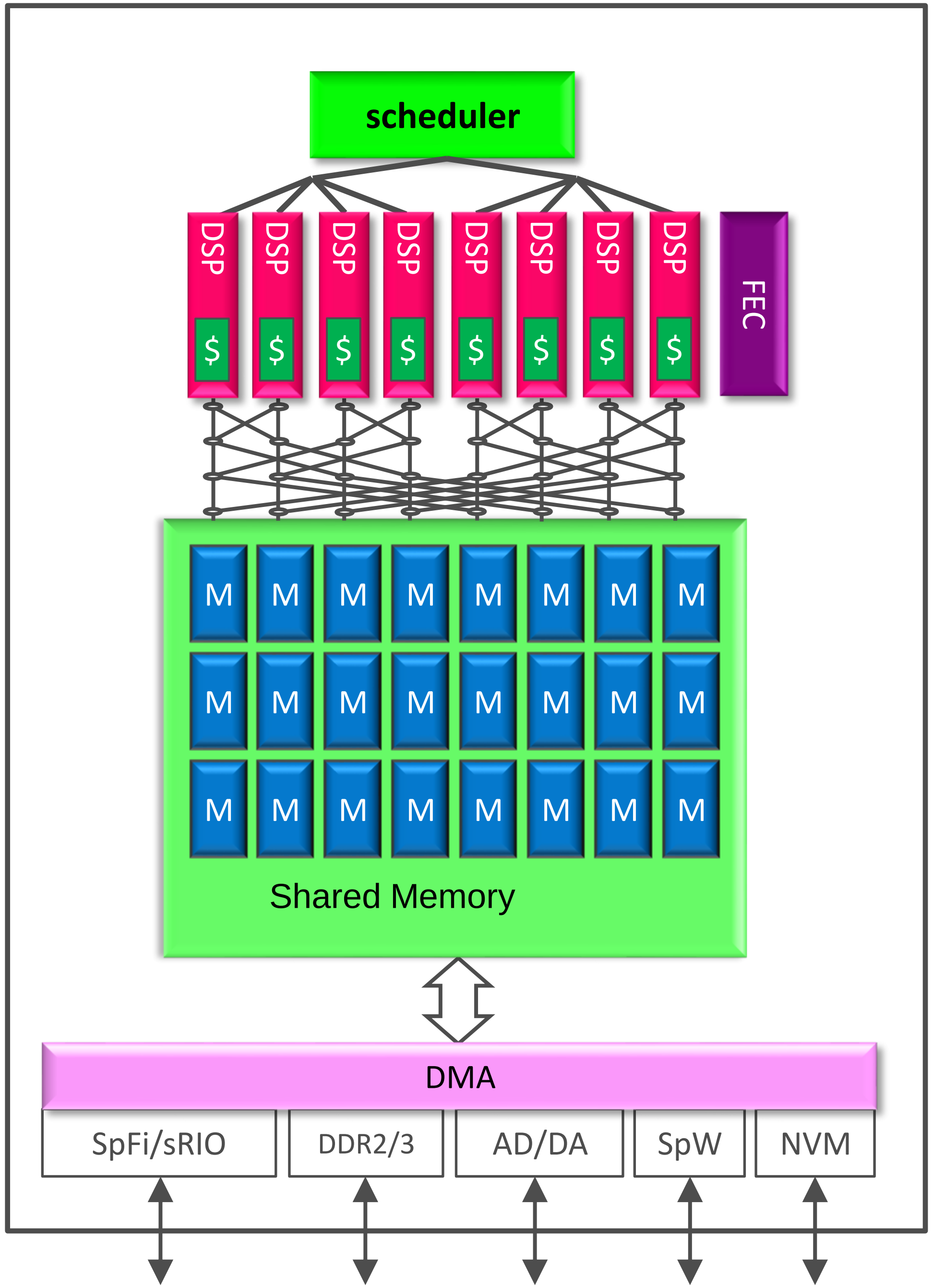
RC64

- Rad-Hard
- Built-in Fault Detection, Isolation & Recovery (FDIR)
- Many-core
  - 64 cores supporting DSP, ML & RISC
- 4 Mbyte shared memory
- Hardware Scheduler
- Fast I/O to DRAM
- Fast I/O to Storage
- Fast I/O Streaming



RC64

- Optimize performance-to-power ratio
- Low voltage & low frequency
  → low Joule/Operation (Watt/MIPS)
- → RC64 is not the fastest
  - Use many RC64 chips to meet required performance

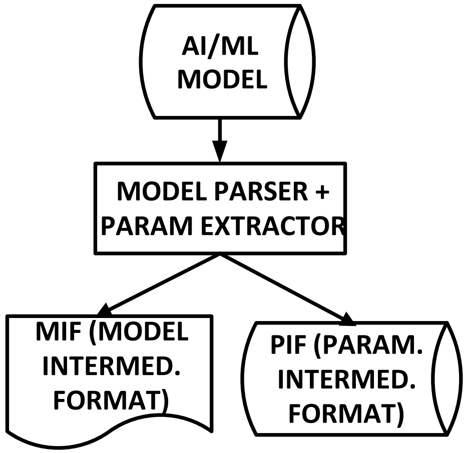


RC64 performing Machine Learning

- One layer at a time
  - Multiple RC64 chips can process multiple layers at same time, or multiple inputs
- Per layer:
  - Read inputs
  - Read weights
  - Perform work
  - Output activations

Inference Development Flow

On the ground



Model Parsing and Parameter Extraction

On the ground

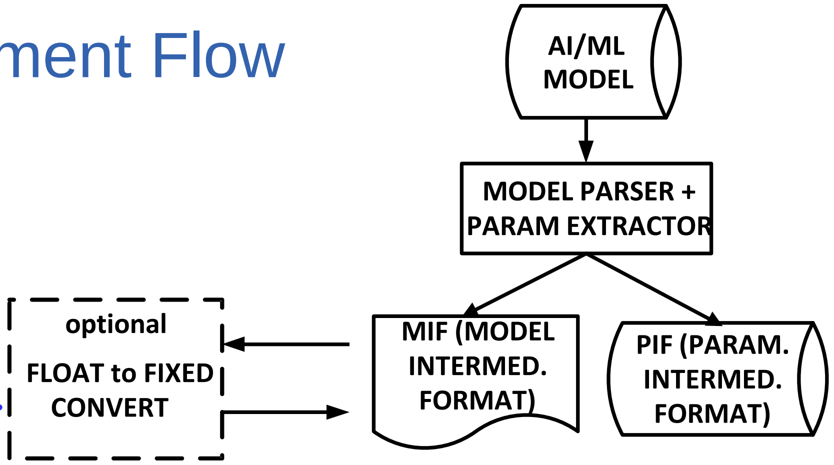
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv32_1 (Conv1D)            (None, 128, 32)           192
_________________________________________________________________
conv32_2 (Conv1D)            (None, 64, 32)            5152
_________________________________________________________________
conv32_3 (Conv1D)            (None, 32, 32)            5152
_________________________________________________________________
conv64_1 (Conv1D)            (None, 16, 64)            10304
_________________________________________________________________
conv64_2 (Conv1D)            (None, 8, 64)             20544
_________________________________________________________________
conv64_3 (Conv1D)            (None, 4, 64)             12352
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense128 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_soft_max (Dense)       (None, 4)                 516
_________________________________________________________________
activation_1 (Activation)    (None, 4)                 0
=================================================================
Total params: 87,108
Trainable params: 87,108
Non-trainable params: 0
_________________________________________________________________
```

```
0:
  name: conv32_1
  type: Conv1D
  output_shape: (128, 32)
  activation: relu
  filters: 32
  input_shape: (256, 1)
  kernel_size: (5,)
  padding: same
  strides: (2,)
  use_bias: true
1:
  name: conv32_2
  type: Conv1D
  output_shape: (64, 32)
  activation: relu
  filters: 32
  input_shape: (128, 32)
  kernel_size: (5,)
  padding: same
  strides: (2,)
  use_bias: true
```

Inference Development Flow

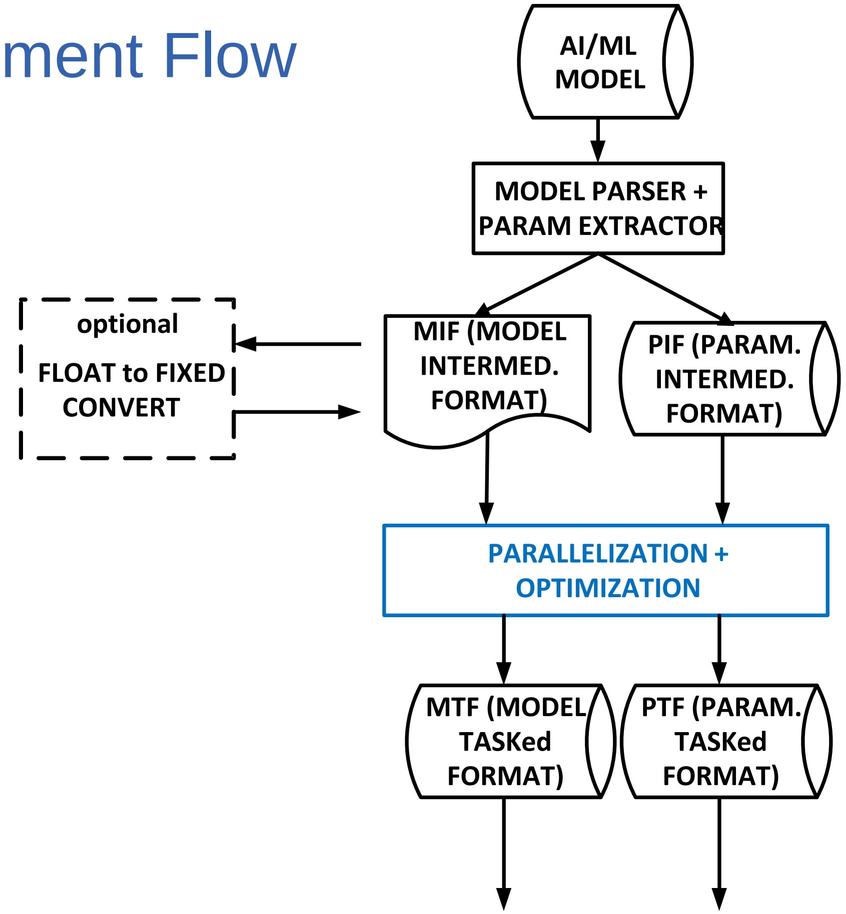
On the ground

Significant challenge is floating point to fixed point dynamic range analysis



Inference Development Flow

On the ground



Model Tasked Format (MTF)

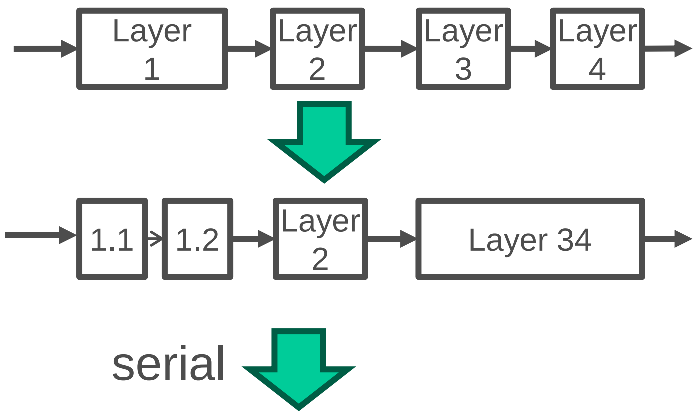
Field	Description
Layer Type	0 - Conv1D, 1 - Conv2D, 2 – Dense, 3 – LocallyConnected1D, 4 – DepthwiseConv2D, 5 – Activation, 6 – AveragePooling1D, 7 – AveragePooling2D 8 – MaxPool1D, 9 – MaxPool2D
Fragment input volume	An array containing the fragment input dimensions
Layer input volume	An array containing the entire layer input dimensions
Input location	The input volume location in memory
Layer output volume	An array containing the layer output volume dimensions
Output location	The output volume location in memory
Kernel size	The convolution kernel dimensions (can be 4D)
Kernel location	The convolution kernel location in memory (buffer and offset)
Strides	Convolution stride (1D or 2D)
Padding	If the fragment is on the volume border, pass the required padding (1D or 2D)
Use bias	In case the fragment computes a point in an output feature of the layer (rather than an intermediate result), it's possible to add bias to the result
Bias location	Bias vector location
Apply activation	In case the fragment computes a point in an output feature of the layer (rather than intermediate result), it's possible to apply activation to the result
Activation type	0 – ReLu, 1 – Sigmoid, 2 - linear
Save output buffer	Output of current layer should be retained for future calculations (used in residual connections)

Inference Development Flow

On the ground

MODEL

REFINED MODEL



Given model

Changed according to our library of layers / kernels

SERIAL CODE

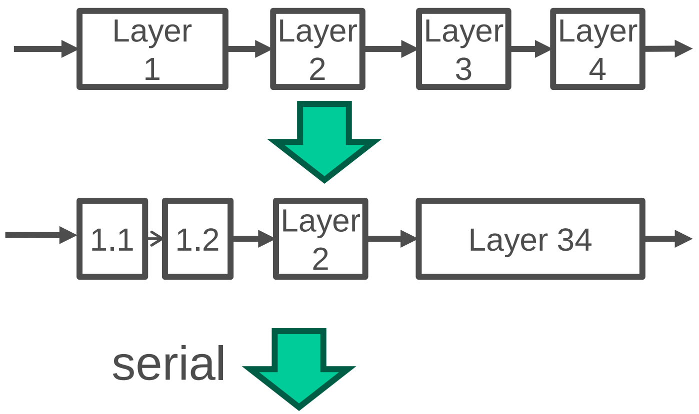
```
main()
{
    …
    conv(1.1,…);
    MaxPool(1.2,…);
    conv(2,…);
    bottle(34,… )
     //conv1,DW3,conv1
     …
}
```

Inference Development Flow

On the ground

MODEL

REFINED MODEL

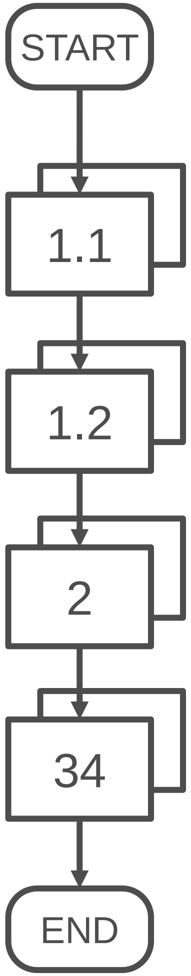


SERIAL CODE

```
main()
{
    …
    conv(1.1,…);
    MaxPool(1.2,…);
    conv(2,…);
    bottle(34,… )
     //conv1,DW3,conv1
     …
}
```

Task Parallelism

multiple concurrent instances of same task

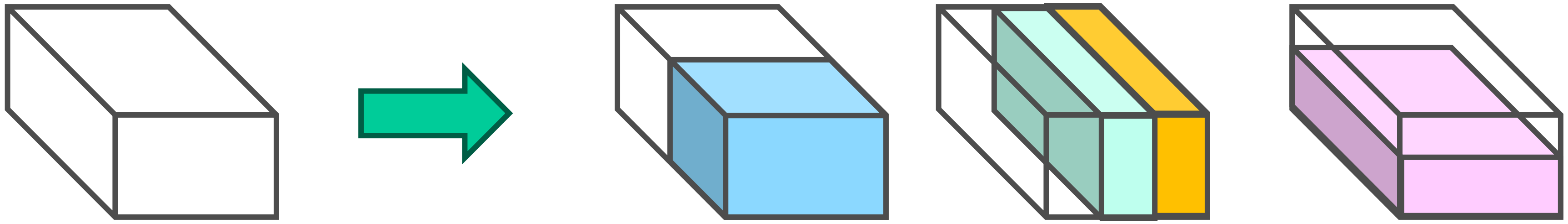


TASK PARALLEL CODE

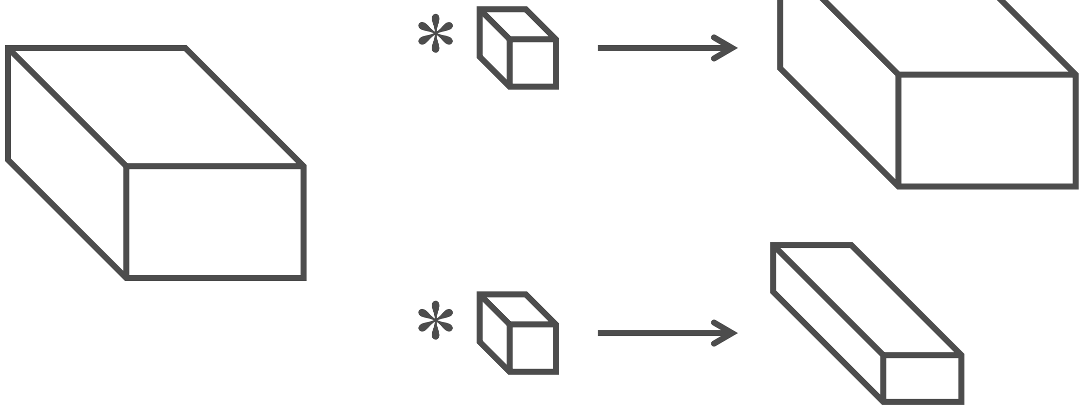
Ramon.Space

Parallelizing a Machine Learning Model

- Input parallelism

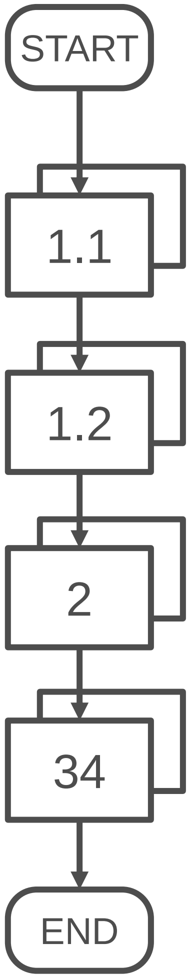


- Output parallelism



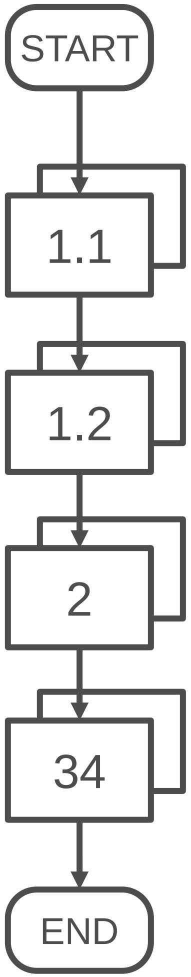
Inference Development Flow

TASK
PARALLEL
CODE

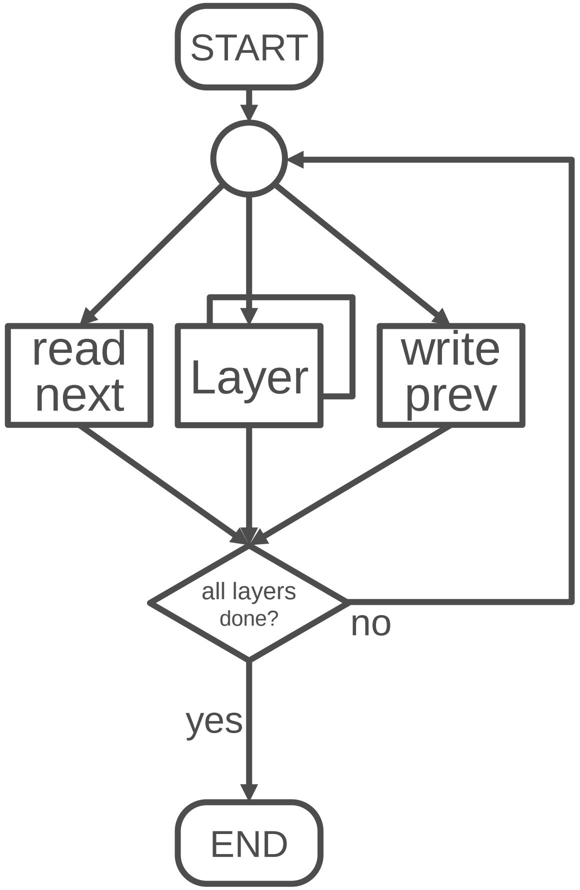


Inference Development Flow

TASK
PARALLEL
CODE



parameterized
generic
layer

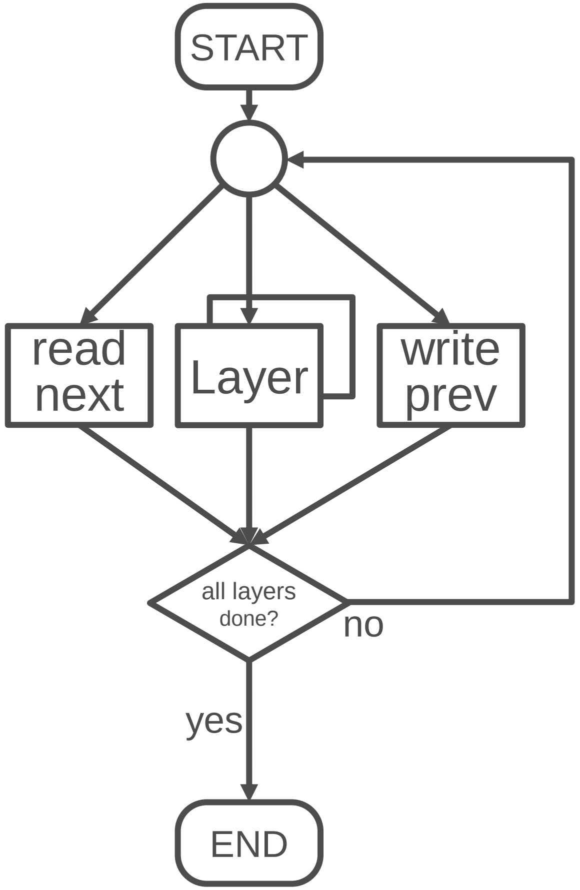


GENERIC
TASK
PARALLEL
CODE

Inference Development Flow

On the ground

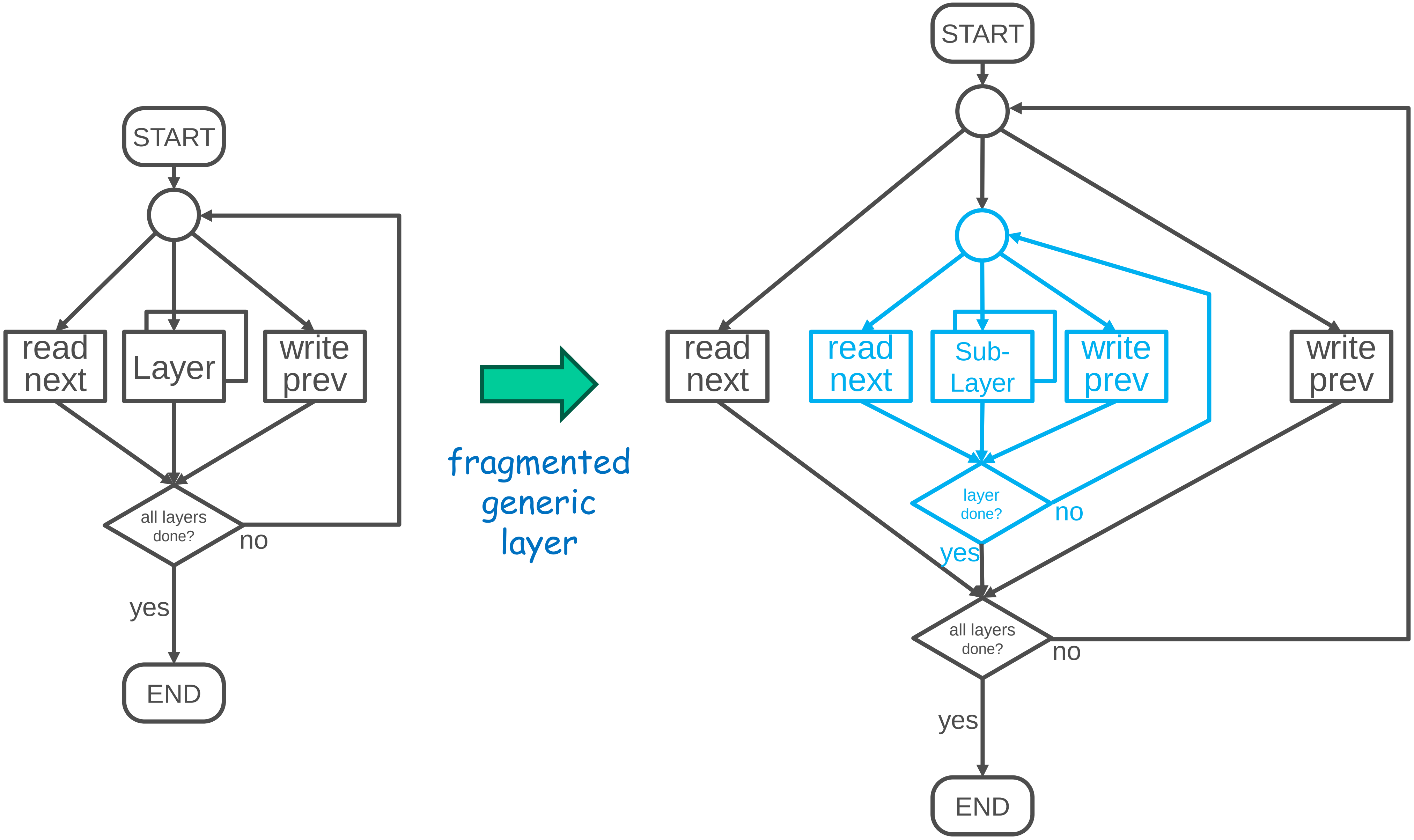
GENERIC
TASK
PARALLEL
CODE



Inference Development Flow

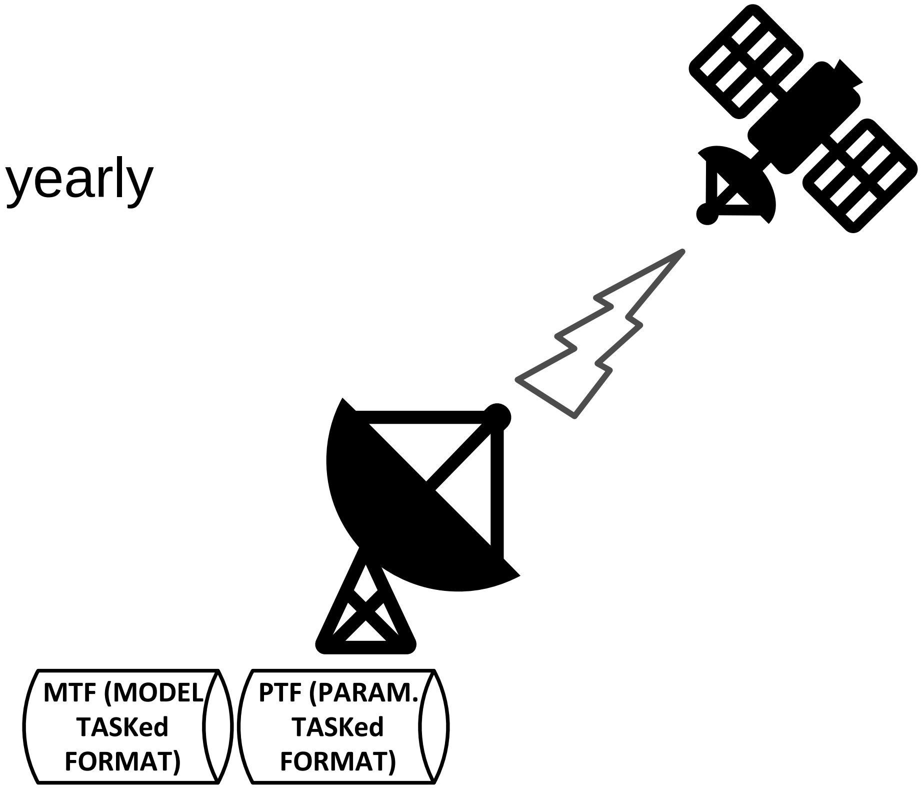
On the ground

GENERIC
TASK
PARALLEL
CODE



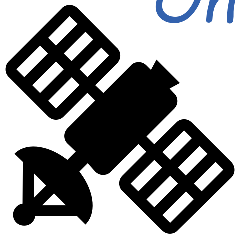
Model Beam Up

- As often as needed: Hourly, daily, yearly

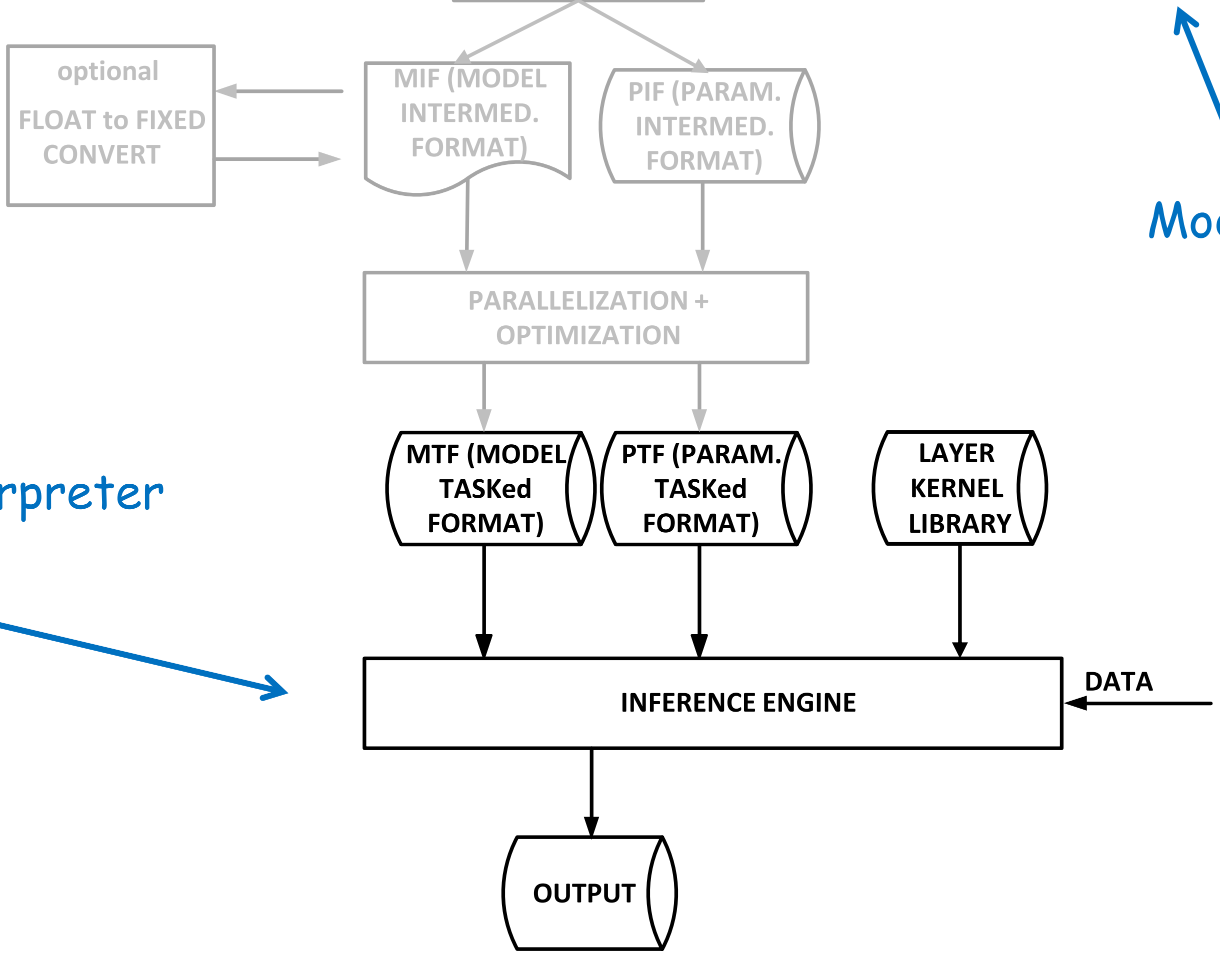


Inference Execution

On-board, in Space



Model executed in Space



Pre-existing Framework interpreter

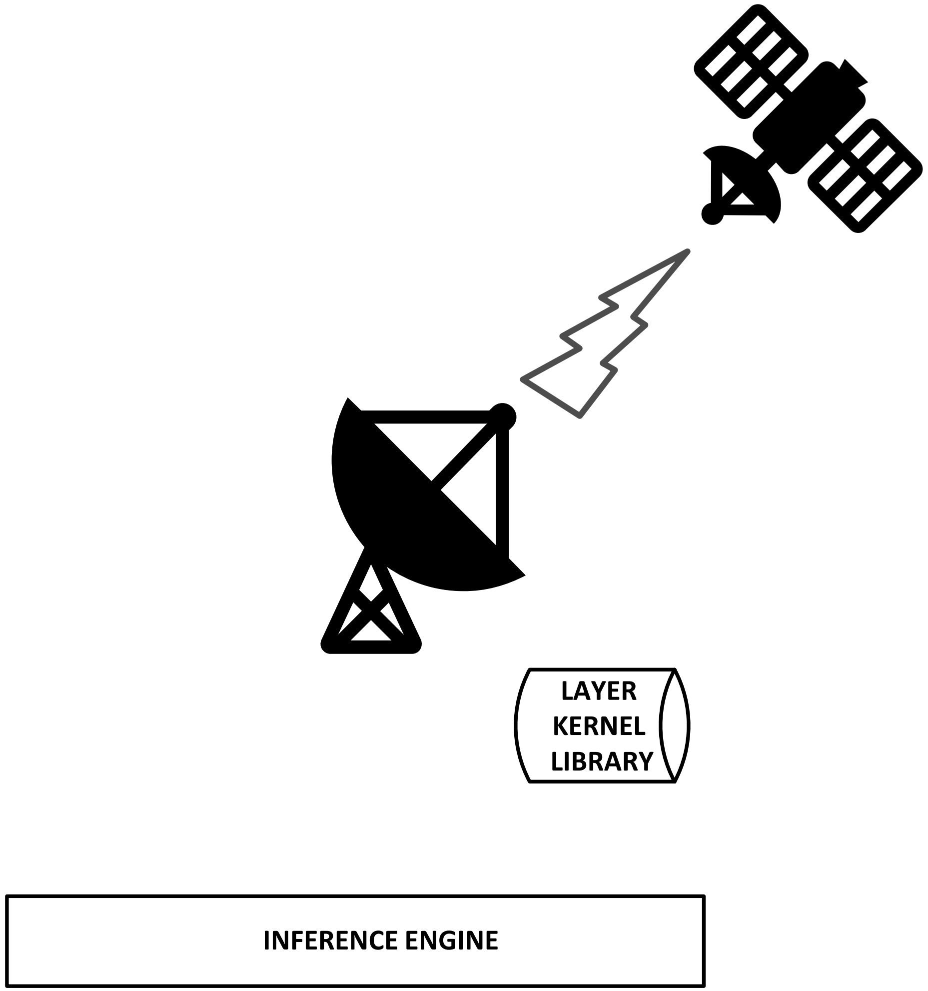
Scalable / upgradeable

Executes the model

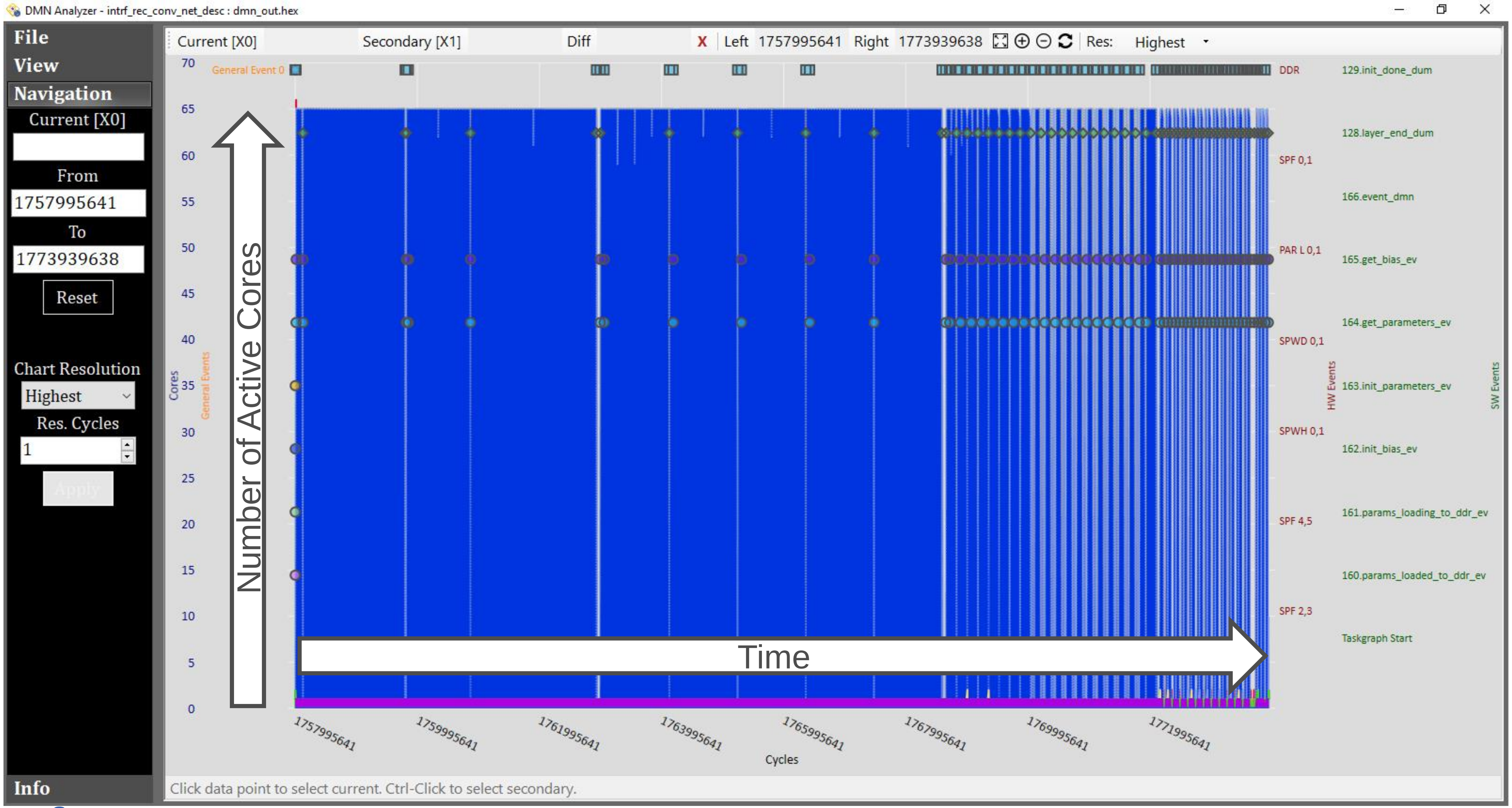
Streaming, or from Storage

Inference Engine Beam Up

- Should happen infrequently
- For new kernels
- For optimization
- For bug patching



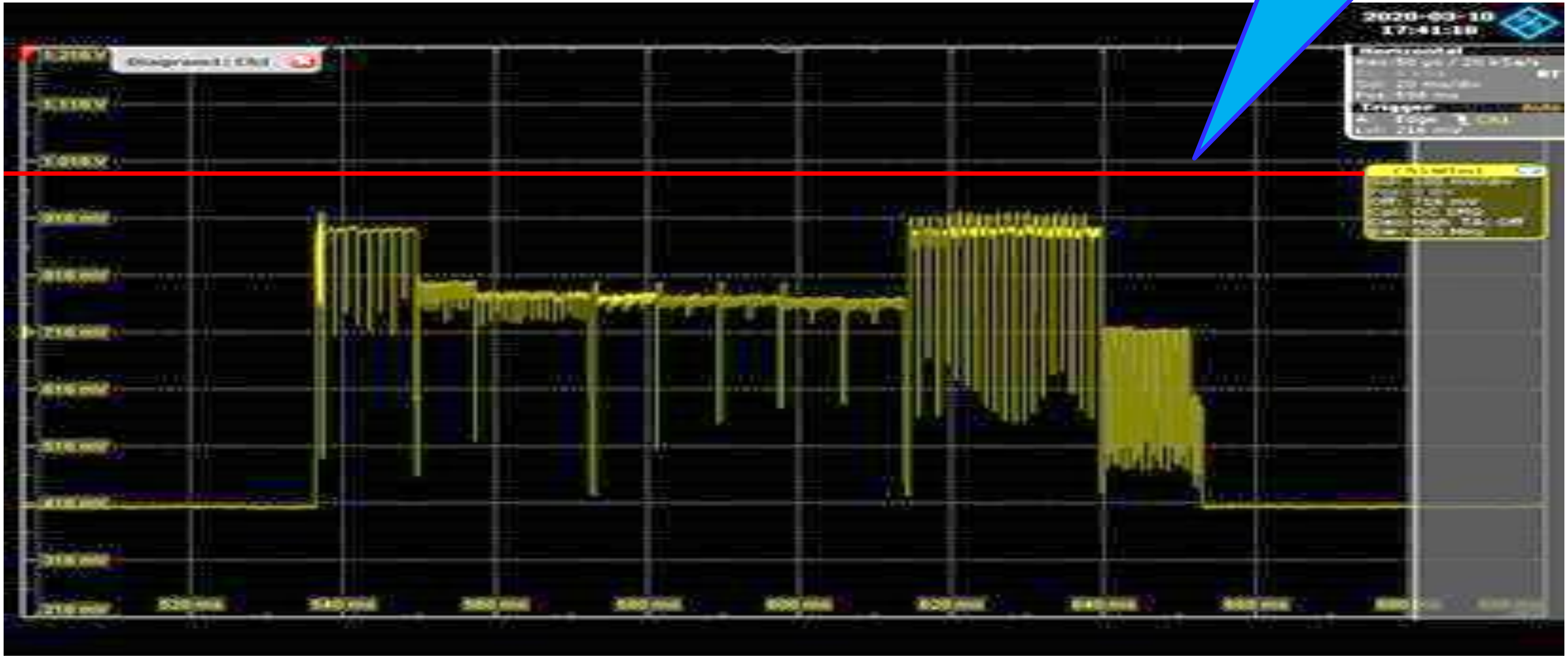
Core Activity Executing VGG Benchmark



Power Consumed Executing VGG Benchmark

- Note barriers between Layers
- Red bar around 4W
- Max power including I/O < 5W

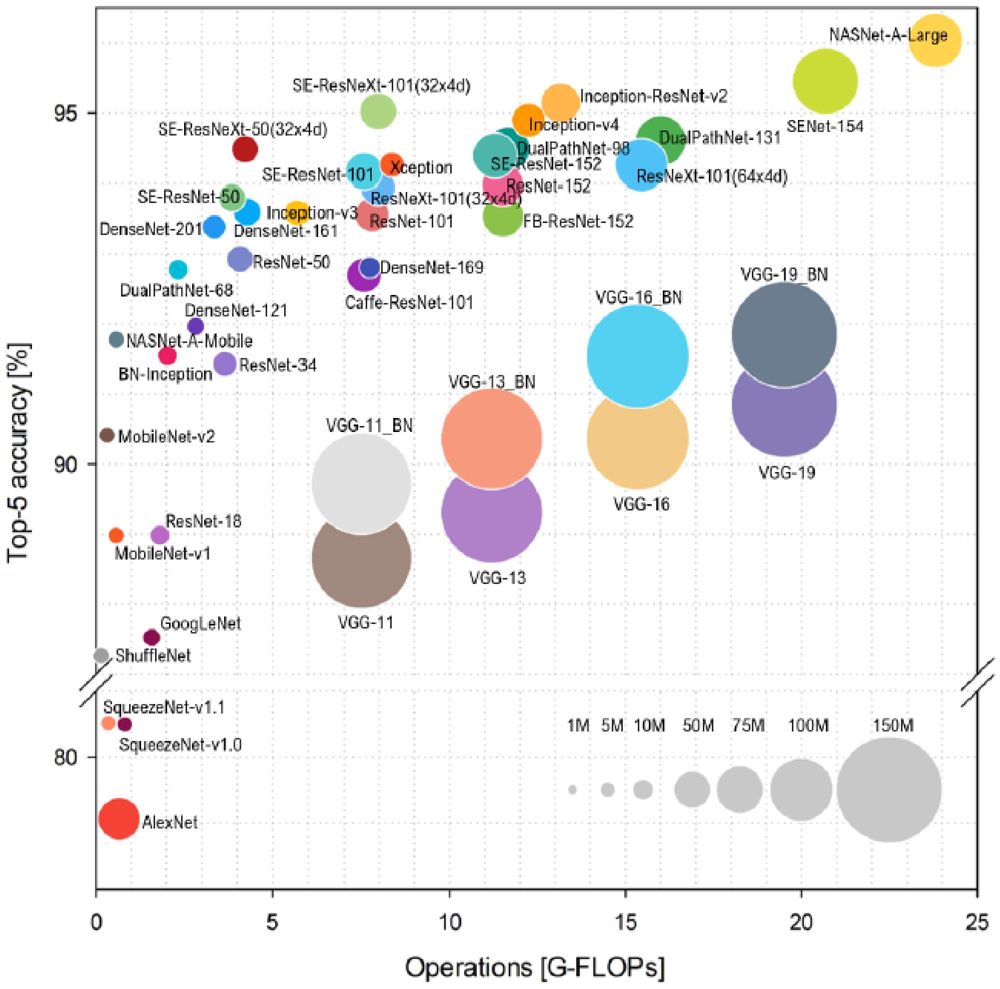
~4W Mark



Why VGG?

- Very large
  - 150M parameters
  - Needs lots of external memory
  - Needs streaming of both input and weights
- Very simple
  - 3 Kernels: Dense, Conv2D (3x3), Max Pooling
  - Needs little implementation effort
- Small images
  - 32×32 to 224×224 images
  - Challenges efficiency

- VGG probably typical of very large ‘future’ models



[Bianco, Simone, Remi Cadene, Luigi Celona, and Paolo Napoletano. "Benchmark analysis of representative deep neural network architectures." IEEE Access 6 (2018): 64270-64277.]

Comparison: VGG Benchmark

- VGG-19, 224×224 images

	Ramon Space RC64		
Space Ready	Yes		
Process	65nm		
Power	5 W		
Frames Per Second	2.8 FPS		
Perf/Power ratio	0.56 FPS/W		

VGG-19 requires 20GOP per frame. 2.8FPS rate consumes 56GOP/sec, 80% of RC64 peak performance

Comparison: VGG Benchmark

- VGG-19, 224×224 images

	Ramon Space RC64	Ramon Space RC256 (roadmap)	
Space Ready	Yes	Yes	
Process	65nm	16nm	
Power	5 W	5 W	
Frames Per Second	2.8 FPS	25 FPS	
Perf/Power ratio	0.56 FPS/W	5.0 FPS/W	

VGG-19 requires 20GOP per frame. 2.8FPS rate consumes 56GOP/sec, 80% of RC64 peak performance
25FPS rate would consume 500GOP/sec on RC256

Comparison: VGG Benchmark

- VGG-19, 224×224 images

	Ramon Space RC64	Ramon Space RC256 (roadmap)	Nvidia Jetson Nano (non-Space)
Space Ready	Yes	Yes	NO
Process	65nm	16nm	16nm
Power	5 W	5 W	10 W
Frames Per Second	2.8 FPS	25 FPS	10 FPS
Perf/Power ratio	0.56 FPS/W	5.0 FPS/W	1.0 FPS/W

https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks

Bianco, Simone, Remi Cadene, Luigi Celona, and Paolo Napoletano, "Benchmark analysis of representative deep neural network architectures," IEEE Access 6 (2018): 64270-64277

Beyond VGG

- ML in Space for EO/Remote Sensing
  - Cloud detection
  - Object identification
  - Change detection
- ML in Space for Communications
  - Spectrum analysis
  - Anomaly & interference detection
  - Modulation classifier
- ML in Space for Robotics, Vision Based Navigation, Docking & Landing
- ML in Space for Spectrum, Network & User Management
- ML in Space for Cybersecurity
- ML in Space for …

ML Requires Storage

- RC64 also serves as long-life rad-hard controller for storage
- Smallest storage product is 1 TByte
  - 10×10cm card
  - High endurance
  - 5 years lifetime at LEO
- Larger storage product under development for GEO
  - 100 TByte to 1 PByte
  - 20—30 years lifetime
- High end computing, storage & networking to enable data-centers in Space

Summary

- Ramon Space enables high-end Machine Learning in Space
- Challenges
  - Space conditions and lifetime
  - Big Data & large models
  - Low power
- Solutions
  - Inference Engine, interpreter of standard models
  - Scalable computing
  - Scalable, durable storage

Ramon.Space