







UNTETHER AI

Energy-Efficient AI Inference Acceleration with Untether AI



Untether AI Addresses Current and Future AI Pain Points

 Issue	Changing AI Workloads	Scalability	Efficiency and Performance	Accuracy
	<p>Neural net architectures are diverse today</p> <p>Will evolve in unknown ways in the future</p>	<p>Inference started in the cloud</p> <p>Will span from cloud to edge in the future</p>	<p>1% of WW electricity usage is in the datacenter*</p> <p>Forecast to be 15% due to the growth in inference deployments*</p>	<p>Deep quantization causes accuracy loss</p> <p>Analog techniques can drift</p>
 Solution	 <p>A flexible, programmable solution</p>	 <p>An architecture that scales</p>	 <p>Extreme power efficiency</p>	 <p>Support the right datatypes</p>

*source: Qualcomm

runAI200™ Device: Industry's First At-Memory Computation Engine

502 TOPs

- Optimized for INT8 inference
- 75W TDP, 50W typical

204MB on-chip SRAM

- 260 TB/s SRAM bandwidth

Scalable voltage/frequency

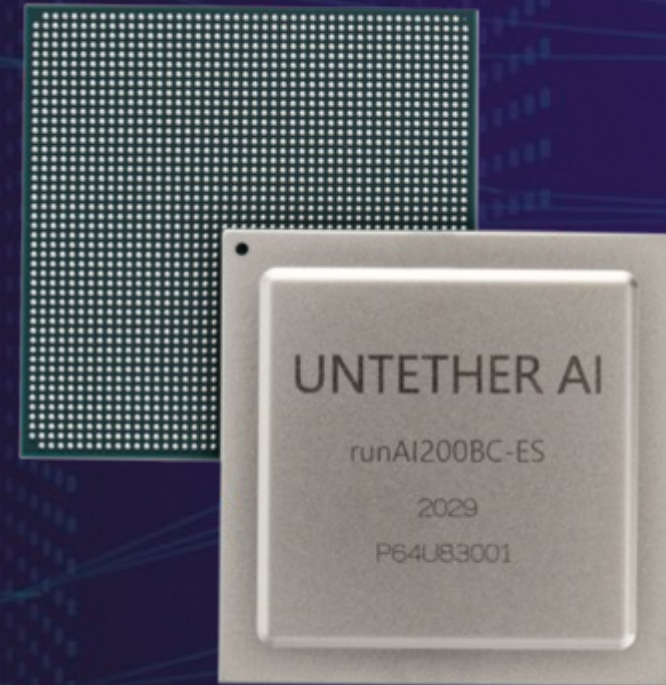
- “Eco” and “Sport” modes, 8 TOPs/W

18.4B Transistors on TSMC 16nm

- 47.5 x 47.5 mm

AI-tailored interconnect

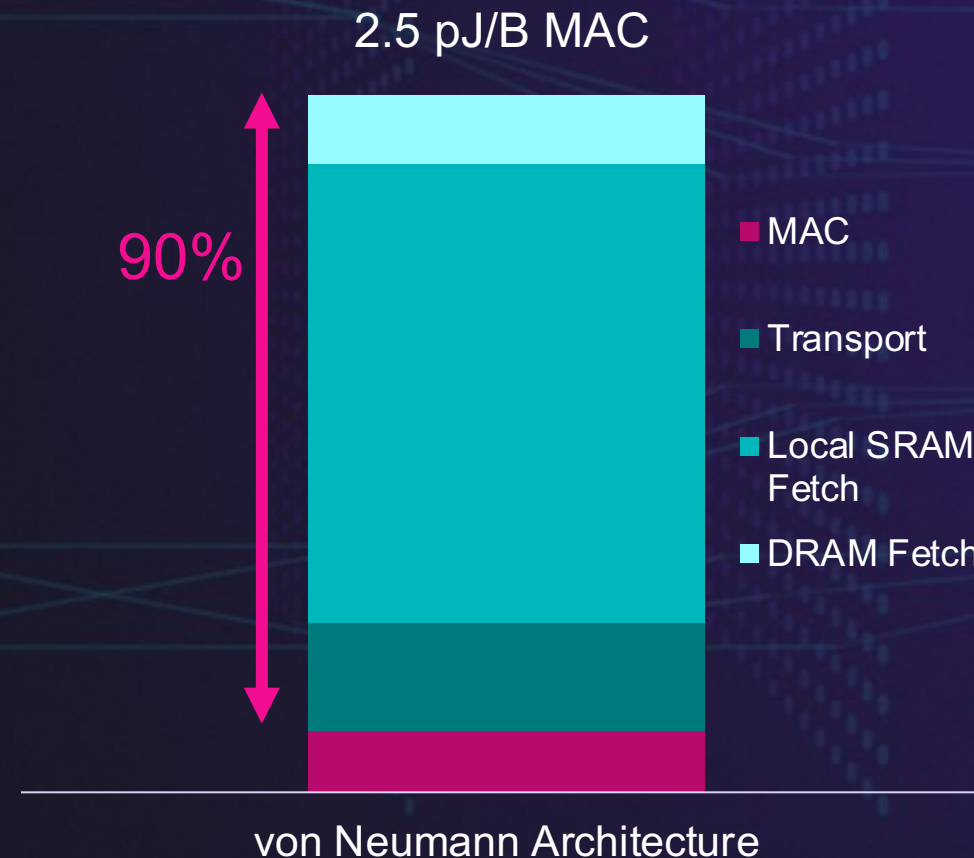
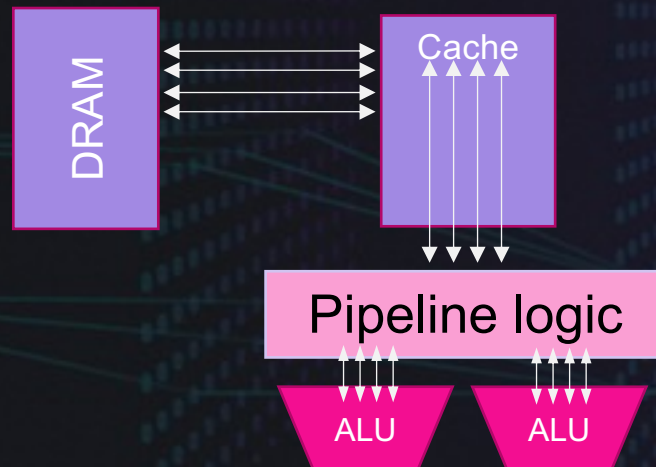
- PCIe Gen4 x16



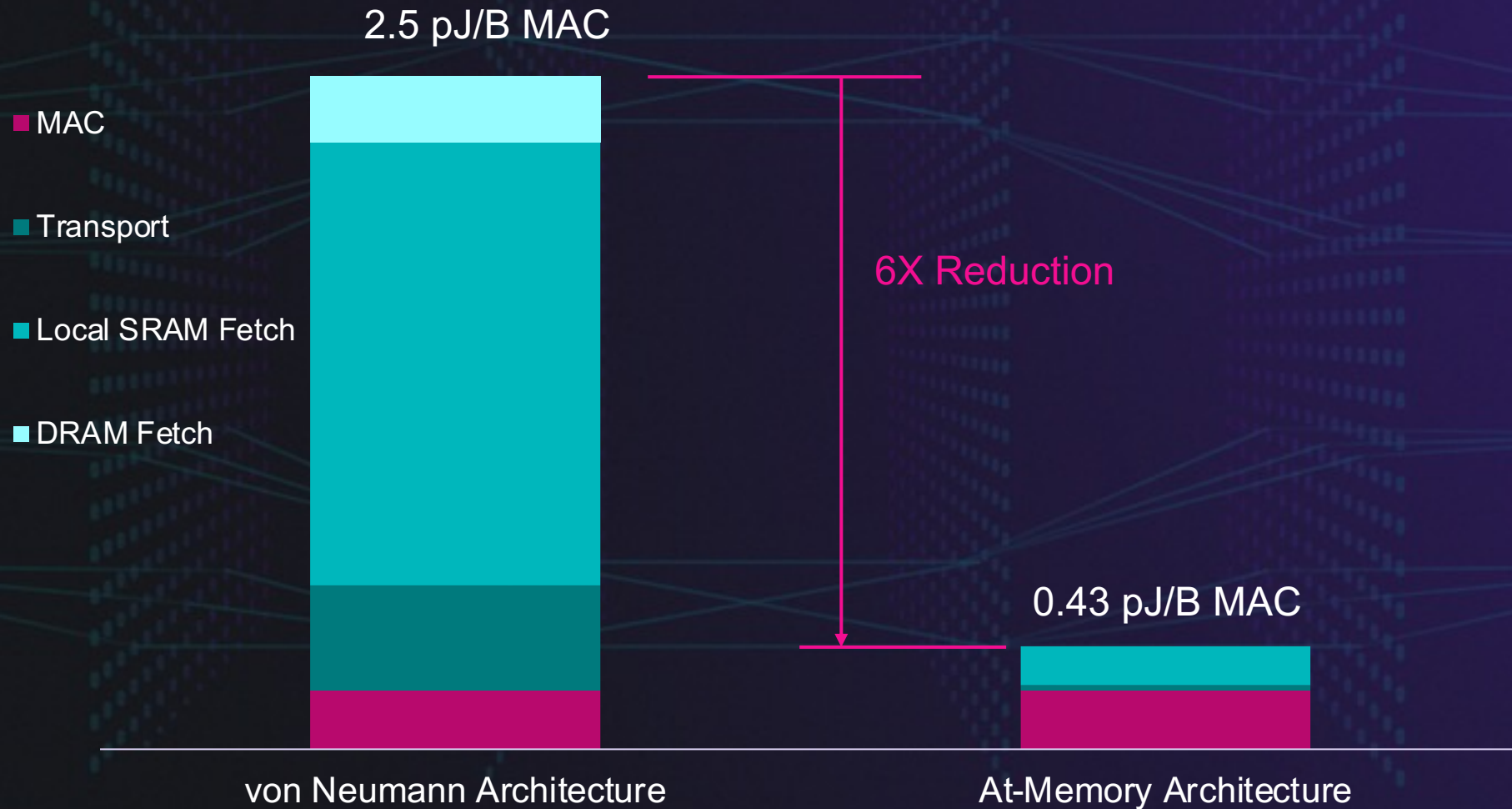
Von Neumann Approach is Wasteful

Data movement accounts for up to **90%** of energy expended in a MAC operation

Von Neumann Architectures

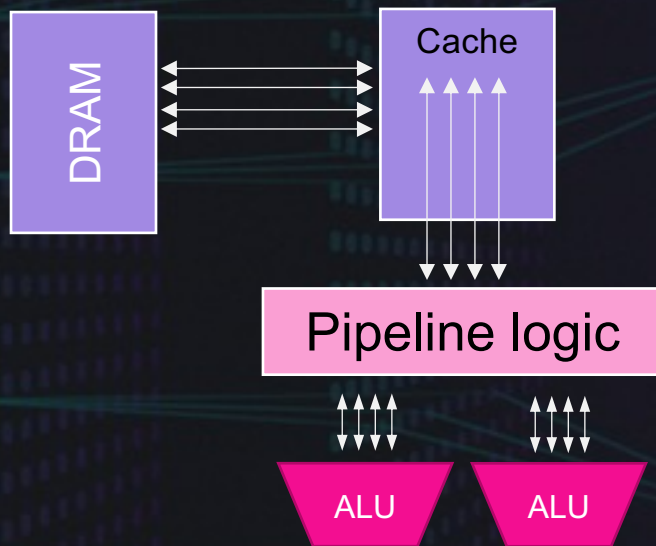


At-Memory Computing is 6x More Efficient



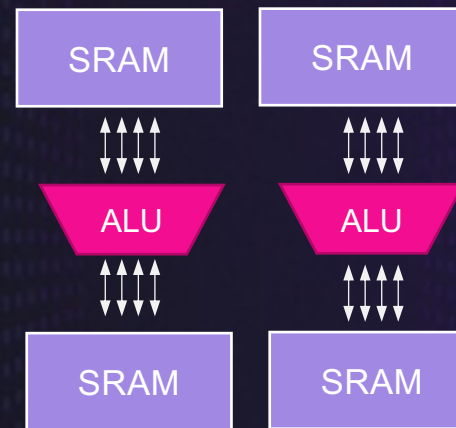
At-Memory Compute Is the Sweet Spot for AI Acceleration

Near Memory/
Von Neumann Architectures



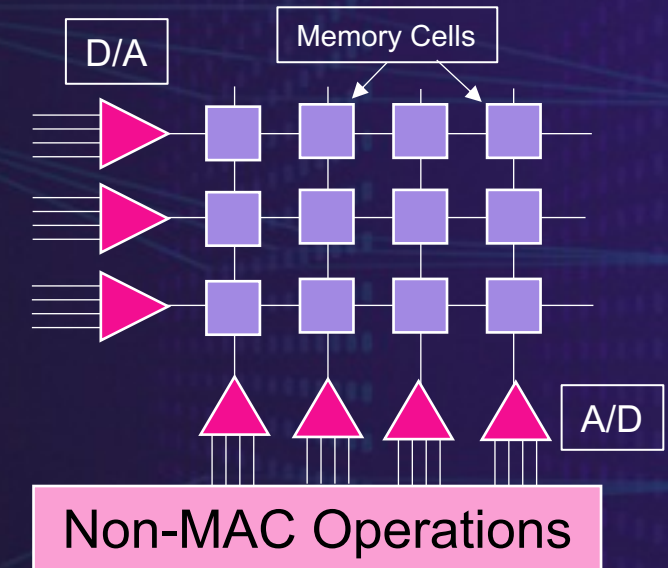
- Long, narrow busses
- Deep/shared cache

At-Memory Computation



- Short, massively parallel direct connections
- Dedicated, optimized memory for efficiency and bandwidth

In-Memory Computation



- Multi-value memory cell
- Analog techniques used for multiply-accumulate
- A/D and D/A support circuitry
- Digital processors for non-MAC operations

Innovative runAI Architecture



imAlpine Software Development Kit

Standard Frameworks and Formats

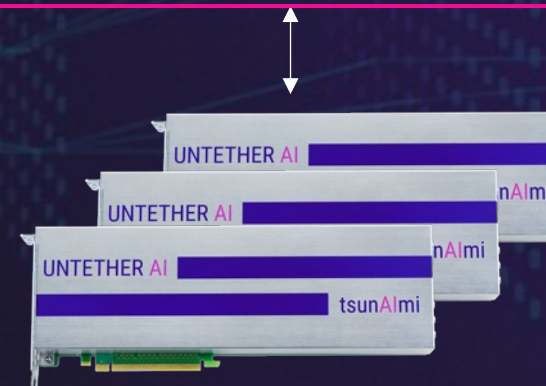
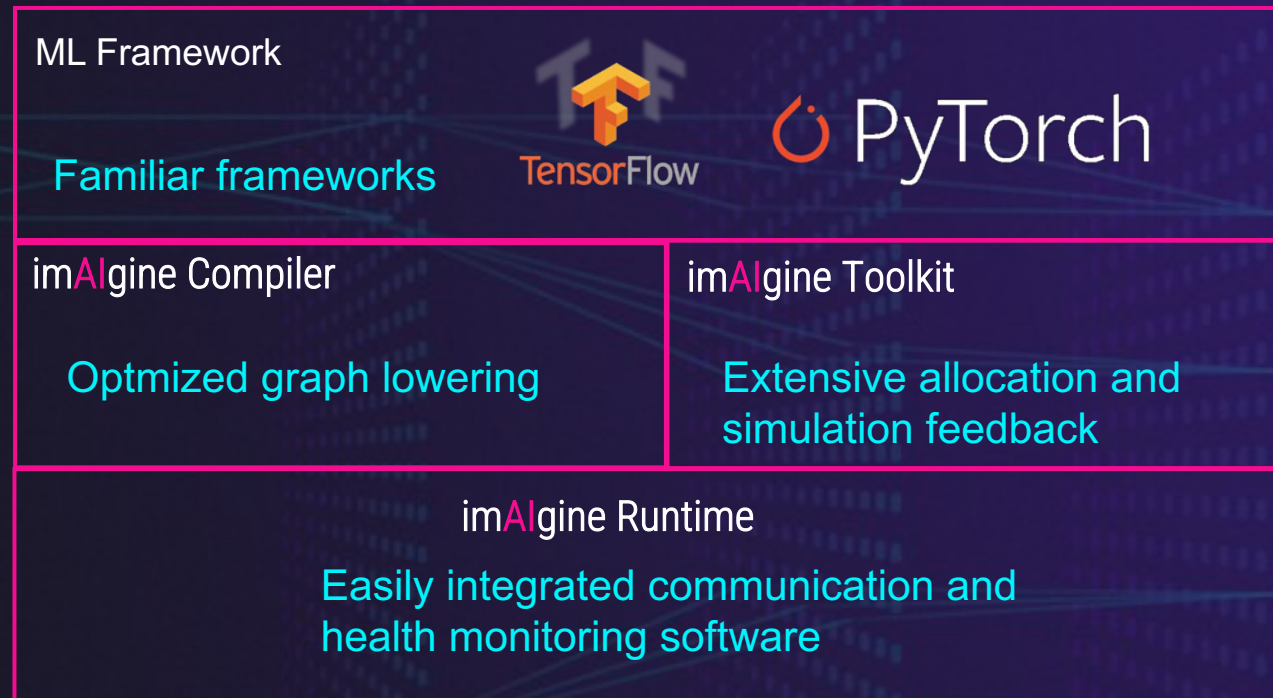
- TensorFlow, PyTorch, and ONNX supported

Robust Toolkit

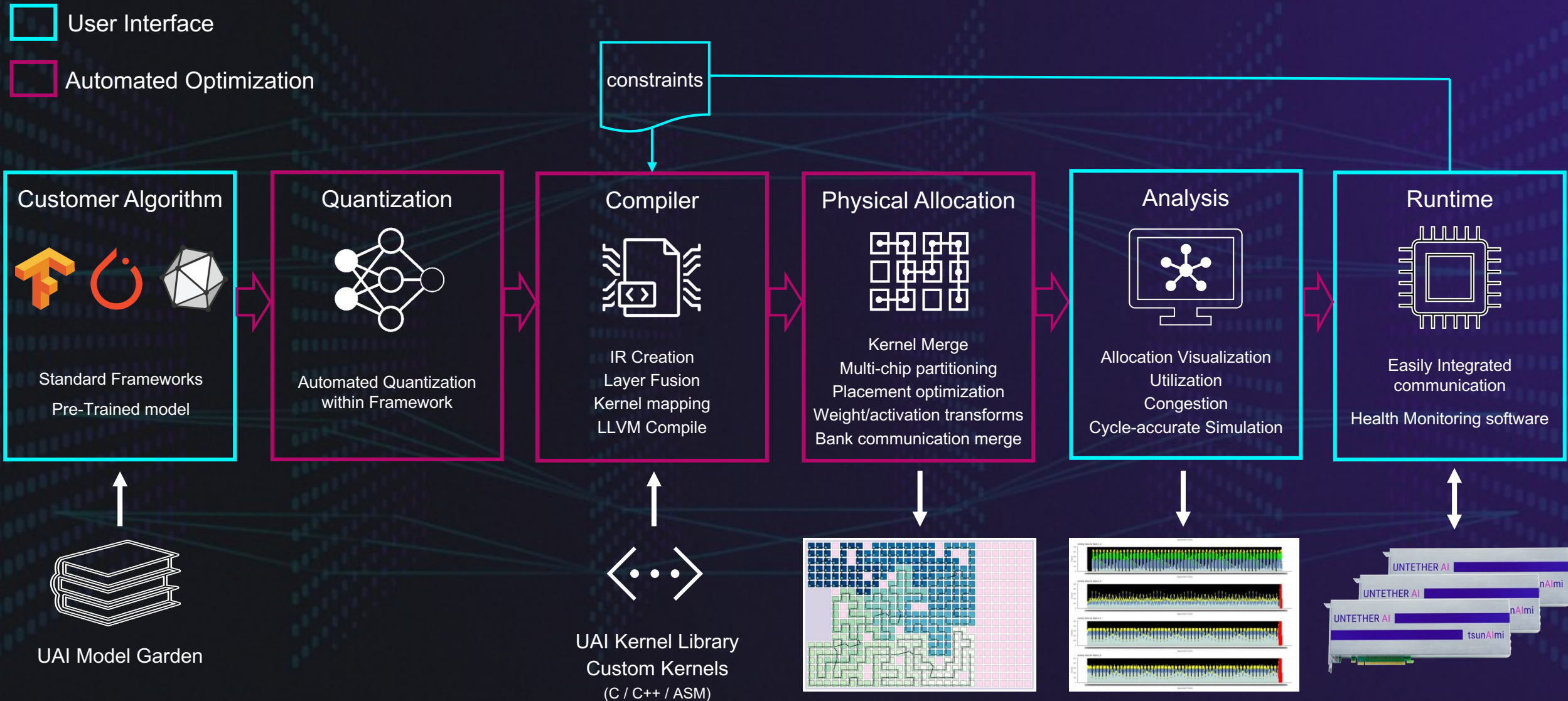
- All the right knobs exposed and tools available to go from trained models to inference ready models

Maintains Accuracy

- Quantizer retains as much accuracy as possible



imAagine SDK Tool Flow



imAlgo SDK: Quantization

User Input



Trained Graph



Automatic ML Framework Optimizations

Automated Quantization
Advanced Quantization (AdaQuant)
Optional Quantization Aware Retraining



Output



TF Quantization Example

```
1 graph = U.read_pb("./resnet50_v1.pb")
2 model = UModel.from_graph_def(graph)
3
4 with U.calibrate(model, U.default_observer):
5     for images, _ in calibration_data:
6         model(images)
7
8 UQuant.quantize(model)
9 model.write_pb('rn50.pb')
10 !imaagine compile 'rn50.pb'
```


imAlaine SDK: Compiler Optimization Options

Optimize for **Efficiency**

Fit in smallest area, least power

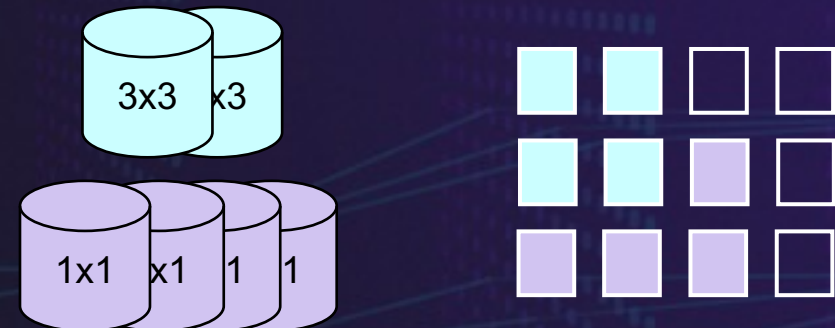


or

Optimize for **Performance**

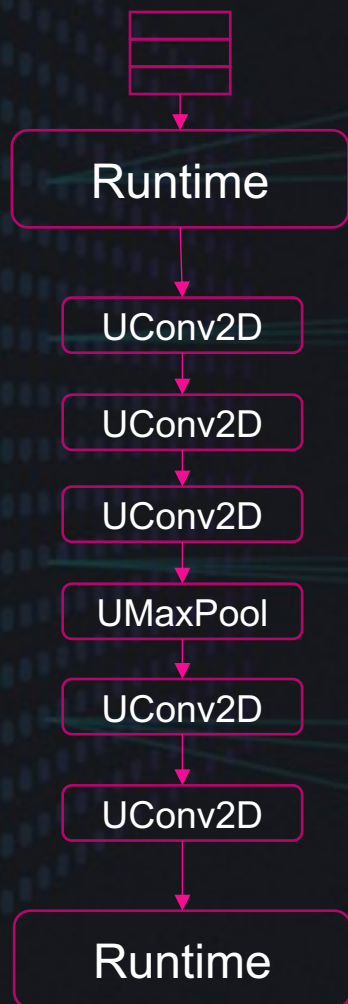
Multiply instances to gain performance

Unique for spatial architectures

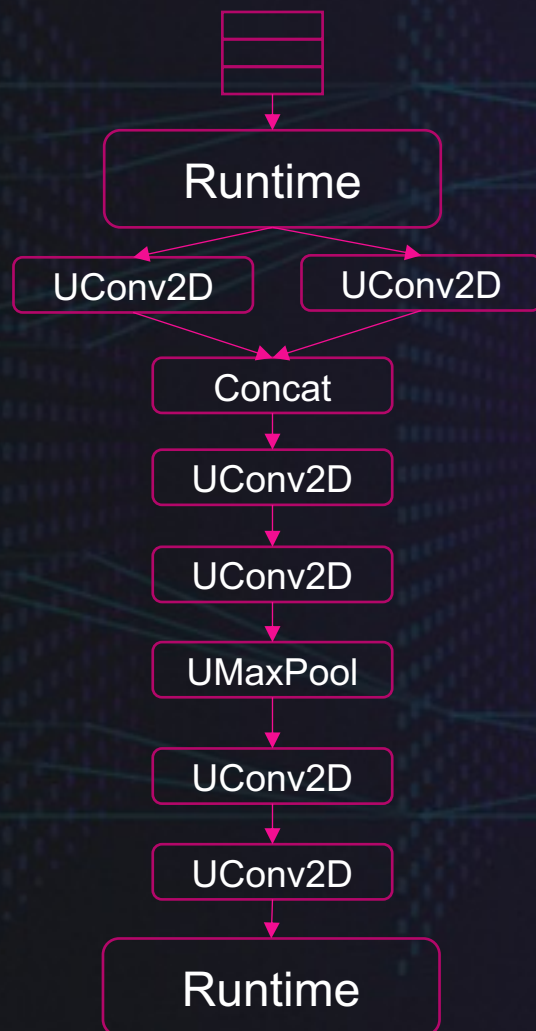


imAIne SDK: Compiler Performance Optimizations

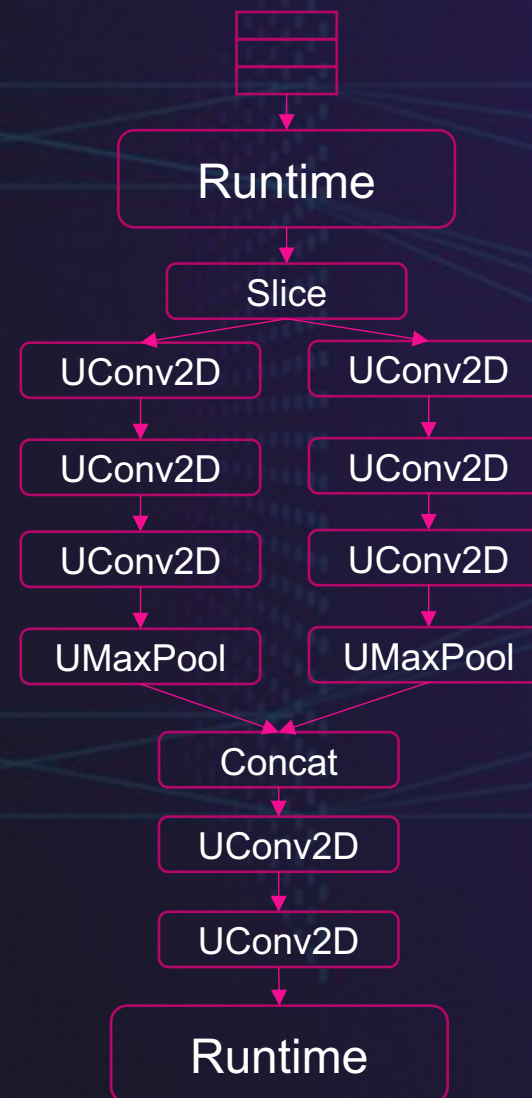
Pipelining



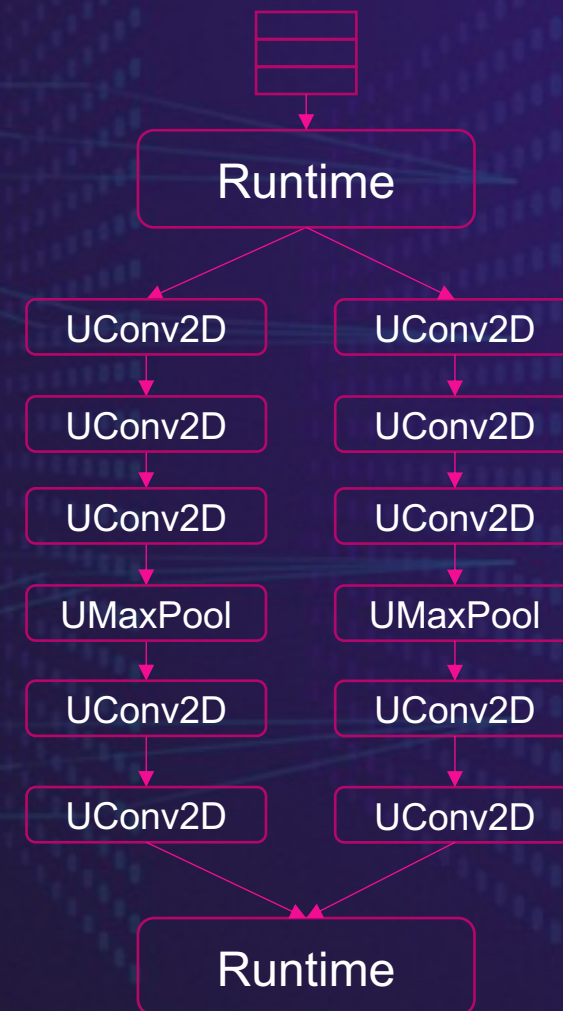
Layer Replication



Sub-Graph Replication



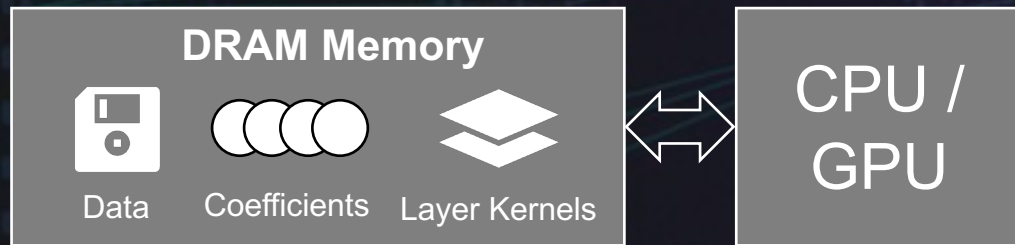
Multi-Instance



imAIne SDK: Physical Allocation

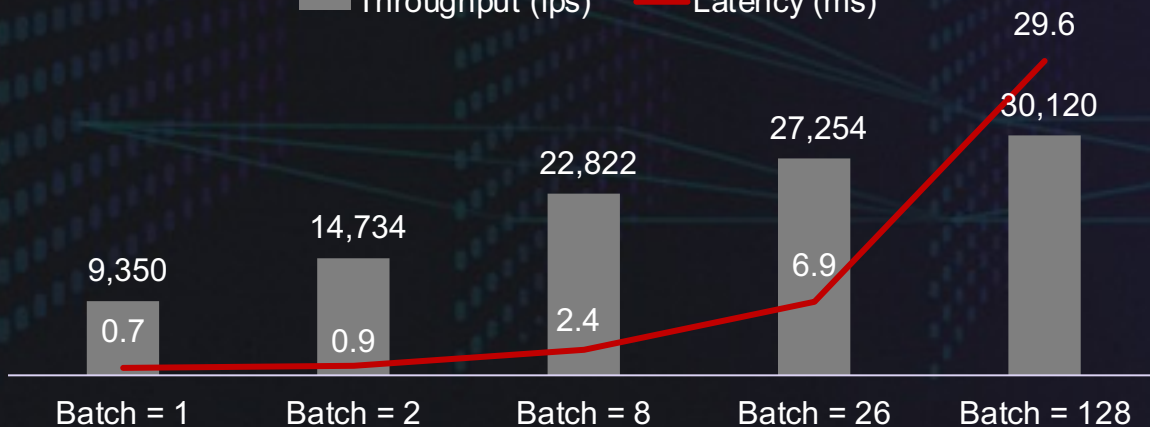
CPU / GPU

Needs to swap layers/coefficients from memory, batching data into larger groups helps with throughput, at the detriment of latency



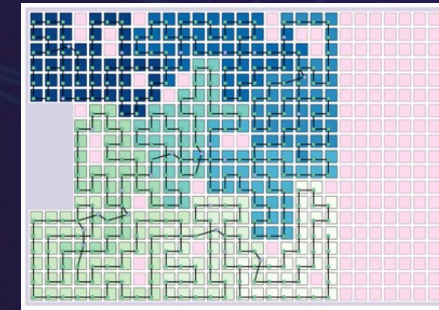
GPU ResNet 50

Throughput (fps) Latency (ms)



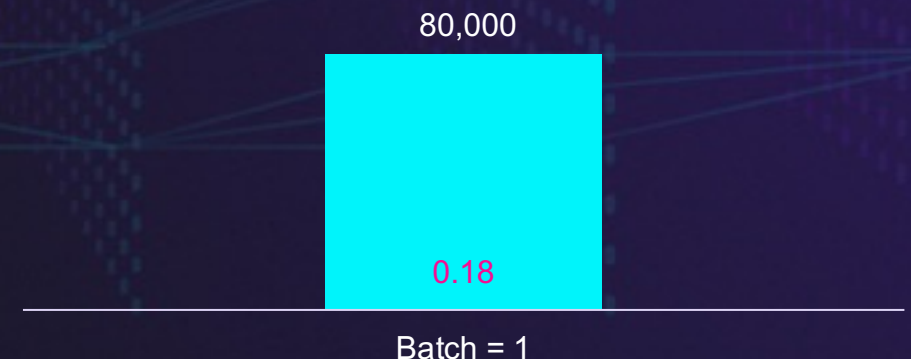
Untether AI

Places the entire network on chip, so is natively batch=1, providing low latency and high throughput



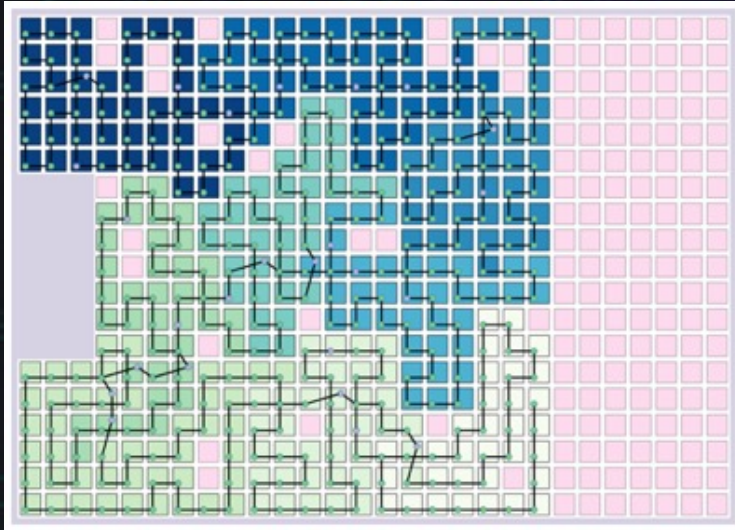
Untether AI tsunAlmi ResNet 50

Throughput (fps) Latency (ms)

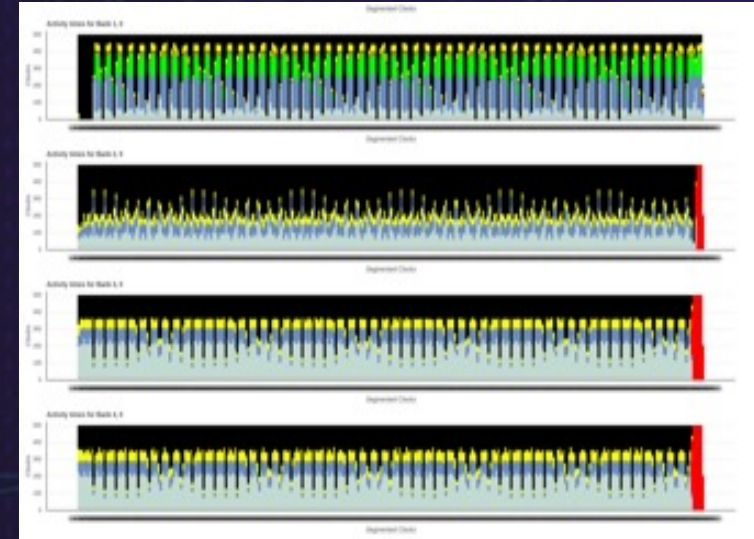


imAIne SDK: Analysis & Cycle Accurate Simulation

Data Flow and Routing Visualization



Simulation

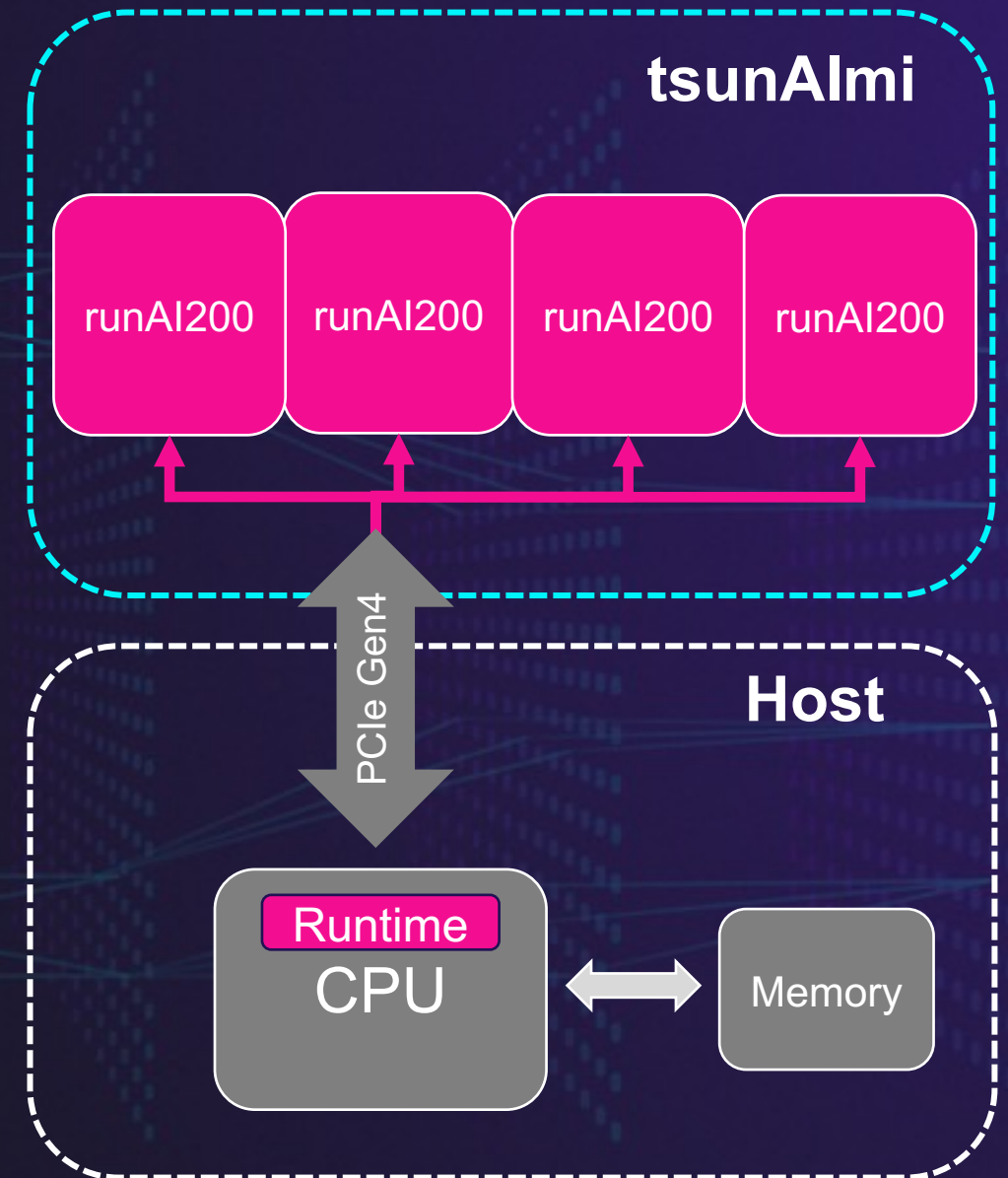


imAlaine SDK: Runtime

- High-Level API in ML framework
- Low-level API to access to various parts of the chip
- System management and debugging tools
- Health monitoring

Runtime API Example

```
1 class ImageRecognizer:
2     def __init__(self, elf_file_path):
3         """Boot the chip with the given ELF file"""
4         self.module = UntetherModule(elf_file_path)
5
6     def process_image(self, image):
7         """Add an image to the processing pipeline, returning an
8         event object used to get status or request the result"""
9         buffers = {
10             'input' : pre_process_image(image),
11             'output' : allocate_output_array(0, RESULTS_SIZE),
12         }
13         event = self.module.enqueue_buffers(buffers)
14         return event
15
16     def get_result(self, event):
17         """Check if an event is ready. If so, return its result."""
18         if self.module.result_ready(event):
19             return self.module.get_result(event)
20         return None
```



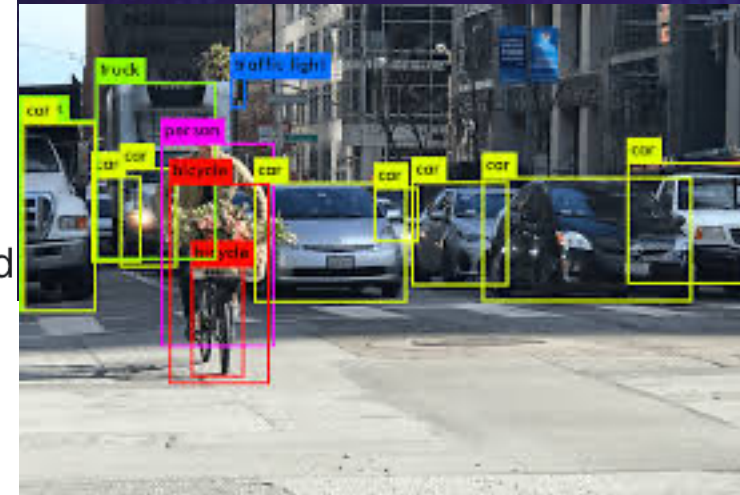
Next Step -> Getting Started

Version: 23.03-23.04

YOLOX-L Inference Demo

The YOLOX-L demo shows the performance of YOLOX-L running on the tsunAlmi card

```
untether-smi --set-profile=SPORT  
make clean  
make USE_COCO_DEMO=1  
USE_COCO_DEMO=1 ./yoloxl_demo -e <path_to_your_elf_file> -d <path_to_coco_dataset_dir>
```





Thank you