GISC 2025

Global ICT Standards Conference 2025

Physical AI

KHRONOS GROUP

Current Status of AI-related Standardization in the Khronos Group

Markus Tavenrath
Principal Engineer Developer Technology, NVIDIA
ML Council Chair, Khronos

ICT Standards and Intellectual Property:
AI for All

Hosted by Ministry of Science and ICT, Ministry of Intellectual Property
Organised by National Radio Research Agency, TTA, IITP, KEA, kista, ETRI

Abstract

This presentation examines the current status and future direction of AI-related standardisation within Khronos.

Physical AI

This session introduces the ongoing efforts to standardise machine learning acceleration within Khronos. As machine learning technologies evolve, the industry faces challenges such as fragmented solutions, duplicated development effort, and slow adoption of new hardware features. The presentation sets the stage for a discussion on how open standards can help address these issues, enabling more efficient, interoperable, and future-proof deployment of machine learning across diverse platforms.

Khronos Connects Software to Silicon

Founded in 2000
~ 150 Members | ~ 40% US, 30% Europe, 30% Asia
ISO/IEC JTC 1 PAS Submitter

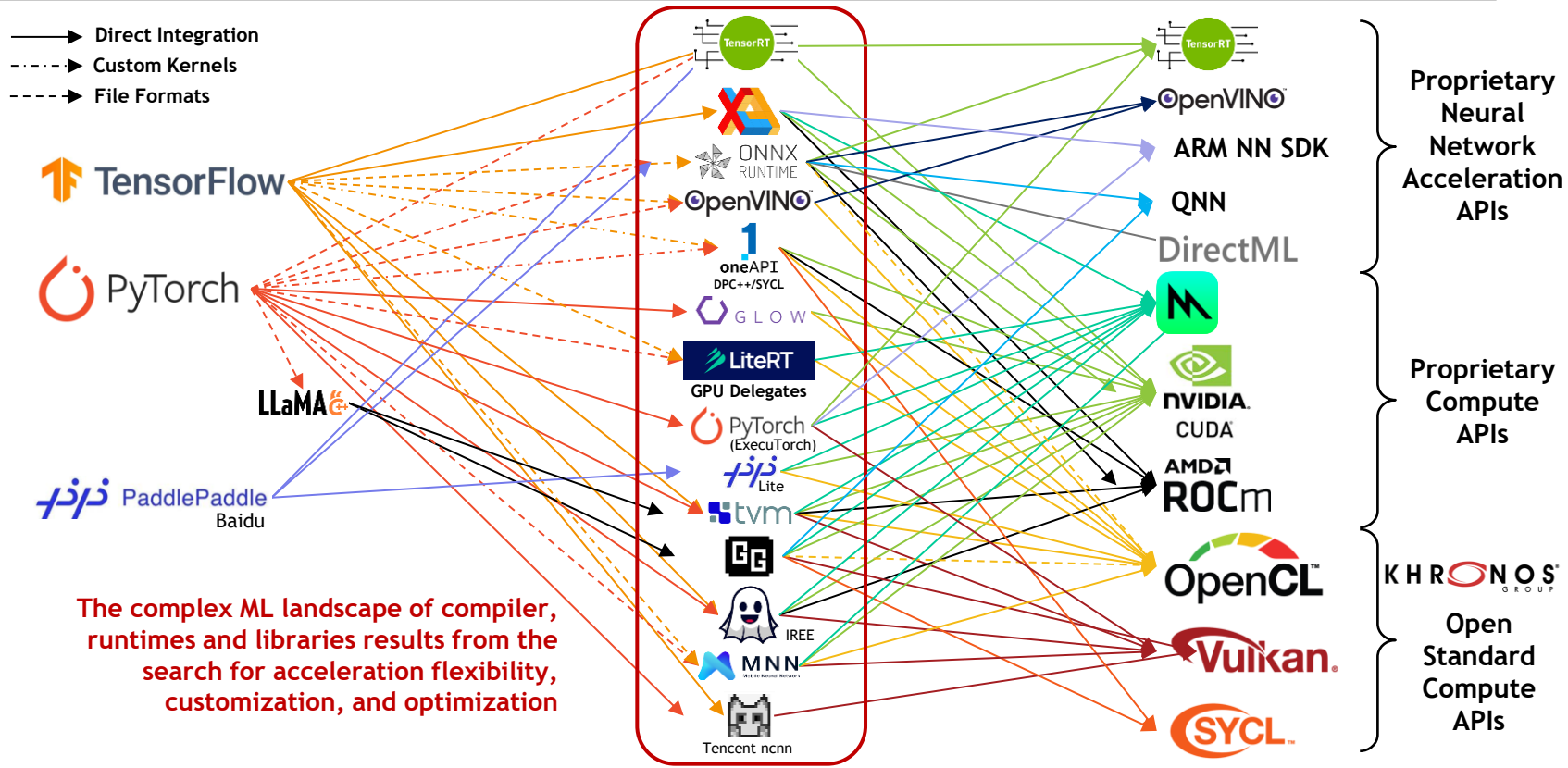
KHRONOS GROUP 25 YEARS

Non-profit Standards Consortium creating open, royalty-free standards

Focused on acceleration APIs and file formats for 3D, XR, AI, vision, and parallel compute

Membership open to any organization

Machine Learning Acceleration Complexity



The complex ML landscape of compiler, runtimes and libraries results from the search for acceleration flexibility, customization, and optimization

Khronos interviewed key decision-makers 1-1 across the entire ML stack

AI Model Experts | Platforms | Hardware Vendors | AI Compilers | Tools Vendors | Standards

Goal was to understand

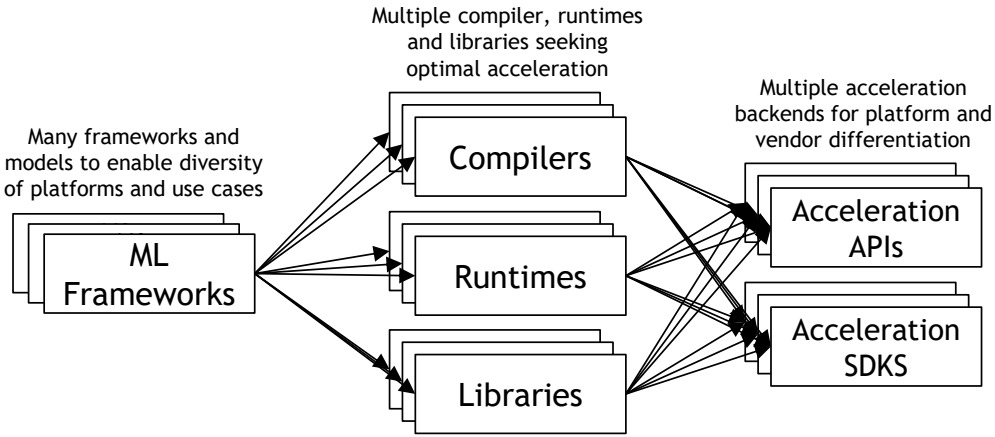
1) Real-world ML acceleration pain points facing hardware and software vendors
2) How can standards bodies, like Khronos, help reduce ML ecosystem complexity

Key result - one common problem

Launching AI products & features requires too much manual expert work!

Up to 85% of ML deployment efforts is the manual expert work needed to develop and optimize backend ML acceleration

ML Ecosystem Acceleration Inefficiency



Multiplied porting and support costs

IHVs must integrate with every framework
Frameworks × # Accelerators × # IHVs

ISVs must support multiple backends

- Increased software complexity and support burden
- Slow adoption of new hardware features
- Fragmented platform support

Lack of Performance Portability

Speed-of-light performance requires expert interventions for each backend

- Model porting & optimization
- Custom operators
- Graph rewriting & scheduling

ML acceleration pain points are slowing innovation!

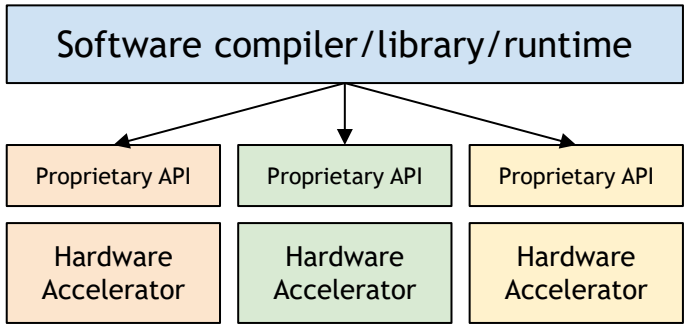
- Expertise bottleneck
- Duplicated effort

= Slow adoption of new acceleration opportunities …

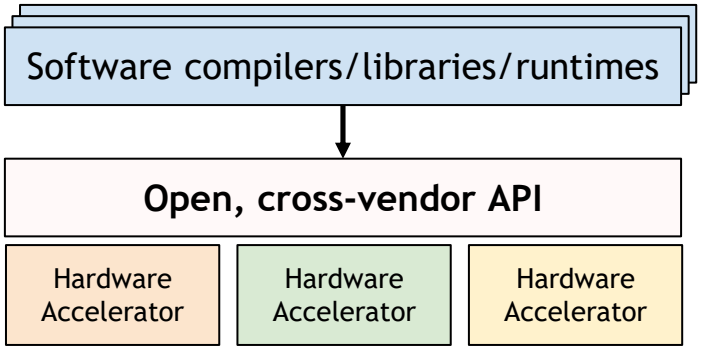
… end users can't run the models they need on the hardware they want!

Can Open Interoperability Standards Help?

- **Open cross-vendor APIs benefit software AND hardware vendors**
  - Software vendors have reduced costs and broder market reach
  - Hardware vendors have access to a broader software ecosystem
  - Robust API design and support with input and investment from multiple companies
- **BUT acceleration standards must be designed with care**
  - Not too soon - only standardize interoperability that is **proven** and **stable**
  - **Extensible** to accommodate technology advances



Custom API per hardware vendor increases software porting and support costs



Cross-vendor API streamlines software development and widens market reach

Hardware Vendors Could Do More for ML?

Sophisticated lowering ML compilers are the most promising route to performance portability - but there are two backend issues that need solving

1. Many compiler stacks define custom operators to try to gain access to efficient operations

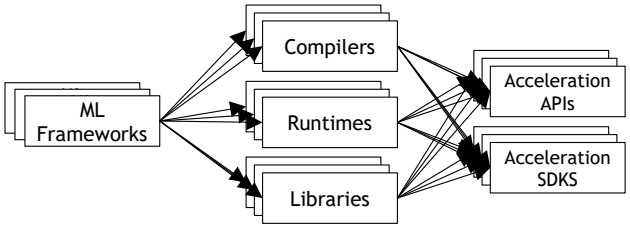
BUT

Resulting operator proliferation means that overwhelmed hardware vendors can't deploy effective operator acceleration fast enough

2. Open-Standard Intermediate Representations (IRs) don't carry operator (graph) information

WHICH MEANS

- **Device driver optimization opportunities are limited**
- **Can't lower GPU-centric IR code to NPUs**



An effective cross-platform standard for ML acceleration could reduce the number of needed backends

Can we create cross-platform standards that address the operator, graph-in-IR and NPU deployment pain points?

‘Default’ ML acceleration on GPUs is to run compute shaders

ML Workloads dominated by Matrix-Matrix Multiply, Matrix Vector Multiply and Convolutions (which can also be expressed as matrix multiplies)

Performance Bottlenecks	**Memory Capacity** Memory size Bandwidth per memory pin	**Compute Throughput** Instructions per clock are limited Max clock is limited
Solutions	**Quantization** Reduce bit precision per weight	**Parallel Operations** Do more work per instruction
Vulkan. **Extensions**	**ML Data Types** Int8 | Int16 Float8 | float16	**Cooperative Matrix Operations** Thread group cooperates on highly parallel matrix multiplies
	2-4x Less Memory Load	**4x+ Speed Increase**

Vulkan (and OpenCL) are continuing to expand data types and intrinsic operations to speed ML operations

ML Adoption of Khronos Open Standards

- OpenCL has been traditionally widely used in mobile and embedded markets
- Vulkan is becoming increasingly used across all markets

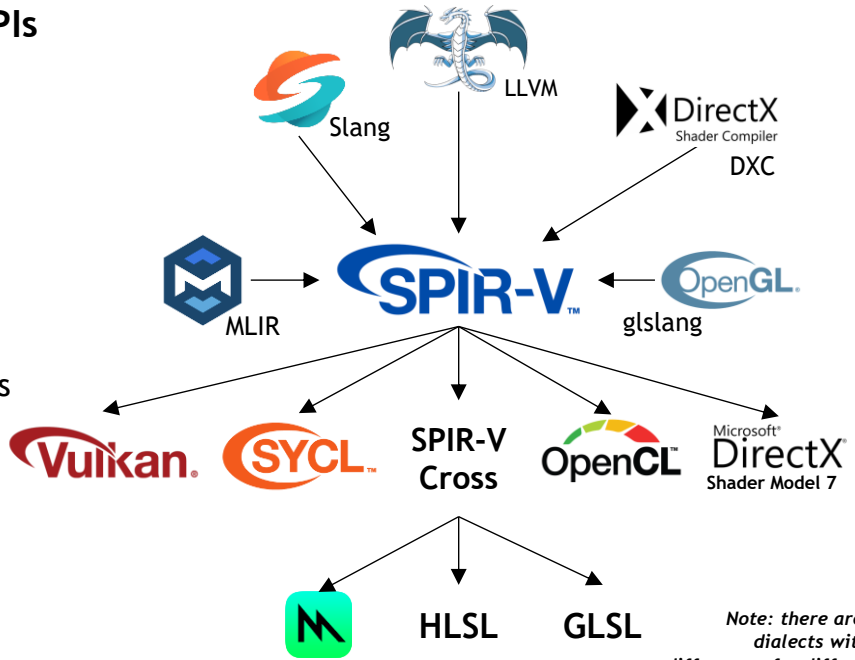
General Frameworks	Type	Khronos APIs
Tinygrad	Framework	OpenCL
Intel OpenVINO	Runtime	OpenCL
Apache TVM	Compiler	OpenCL, Vulkan
Meta Glow	Compiler	OpenCL
SYCL / DPC++ / OneAPI	Compiler	OpenCL
GGML	Runtime	OpenCL, Vulkan
IREE	Compiler/Runtime	Vulkan
AI Inc. Ailia SDK	SDK	Vulkan

Mobile / Embedded Frameworks	Type	Khronos APIs
Cadence Xtensa Neural Network Compiler (XNNC)	Compiler	OpenCL
CEVA Deep Neural Network compiler (CDNN)	Compiler	OpenCL
Synopsys MetaWare EV	Runtime	OpenCL
VeriSilicon Acuity	Runtime	OpenCL
Texas Instruments DL Library (TIDL)	Library	OpenCL
Arm Compute Library	Library	OpenCL, Vulkan
Xilinx Vitis AI	SDK	OpenCL
QNN: Qualcomm Neural Network SDK	SDK	OpenCL
Google LiteRT	Runtime	OpenCL
Alibaba MNN	Runtime	OpenCL, Vulkan
Xiaomi Mace	Runtime	OpenCL, Vulkan
Baidu Paddle Paddle Lite	Runtime	OpenCL
ExecuTorch	Compiler/Runtime	Vulkan

Note: frameworks may have additional supported backends

Khronos SPIR-V Intermediate Representation

- **SPIR-V is the IR for Vulkan, OpenCL and SYCL**
  - Designed to support graphics and compute use cases
  - Simple binary format, easy and fast to parse and analyze
  - Decouples compute and shading language from the API
- **Industry support is widening beyond Khronos APIs**
  - Official LLVM backend since LLVM 20
  - MLIR dialect available
  - Adopted for DirectX Shader Model 7
- **Easily extendible to add new features**
  - Expose innovative features in your hardware
  - Enable the industry to widely adopt them
- **Enables forward compatibility**
  - New tools generate bytecode readable by older drivers
  - Compiler update does need a driver update
- **Extensive SPIR-V open-source tooling**
  - Read, write, validate and transform SPIR-V



Note: there are SPIR-V dialects with minor differences for different APIs

- **Arm vendor extensions add graphs and tensors to SPIR-V**
  - Arm's TOSA is the initial operator set
  - Extensible to any other operator set
  - But no custom kernel support
- **Complete neural network is sent to the driver as one SPIR-V blob**
  - Driver can run a full graph optimizer on the network to optimize execution
- **This solves multiple identified pain points!**

Should Khronos pursue making SPIR-V Graphs a cross-vendor open standard?

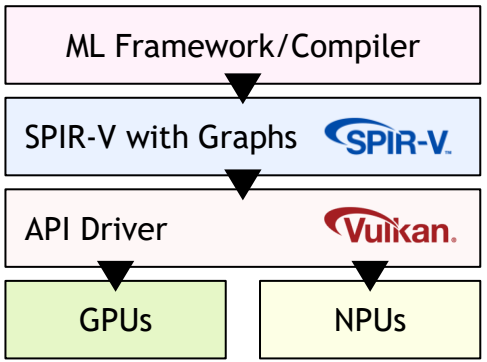
Drivers have full graph information for optimization

Offload operators to special hardware, e.g. NPUs

Choose optimal Kernels

Choose optimal tiling for best cache efficiency

Drivers have the most detailed architectural knowledge of available processor resources

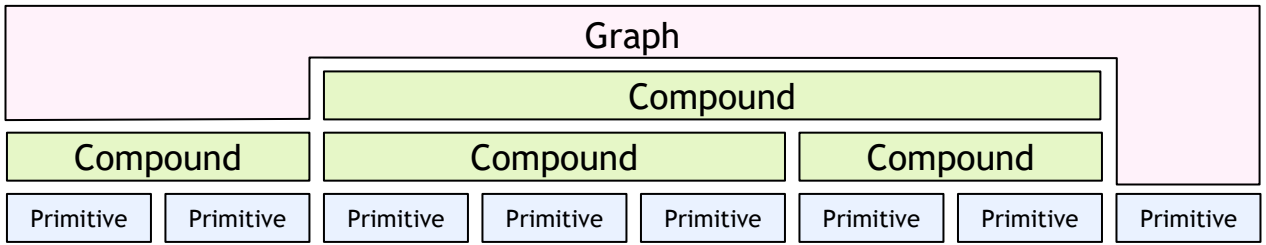


Reducing Need for Custom Operators

- **How to handle custom operators in the SPIR-V graph?**
  - Custom operators usually are defined through compute kernels
  - The Arm SPIR-V graph extension does NOT support compute kernels as graph nodes
- **We are seeking industry feedback - what are your custom operator requirements?**
  - Our research says 20-30% of typical pipelines are custom operators
  - Custom nodes distributed equally in the graph?
  - Just used for preprocessing and postprocessing?
  - Large single big operator in the middle of a graph?
- **If it's preprocessing or post processing the current graph extension works fine**
  - Unless data transfer or data layout conversion is required
- **For everything else**
  - Do you consider synchronization as problem?
  - Do you think it's valuable to have compute kernels in the graph?
  - What kind of programming model do you expect?

NNEF™

- **NNEF is a Khronos open standard file format for describing neural networks**
  - Two key aspects: neural structure and network data
  - Primary focus today is neural structure
- **NNEF 2.0 defines the SkriptND domain specific language**
  - Shape inference
  - Runtime computation of tensor operators called “primitives”
  - Compounds of primitives - and other compounds
  - Control flow constructs (branching / looping at the graph level) for dynamic models
  - Full neural network graphs consisting of primitives and compounds
- **Networks defined with SkriptND are fully self-contained**
  - New operators can be defined by exporter tools or the developer as needed



SkriptND primitive and compound operators

Advantages of SkriptND Flexibility

- **SkriptND enables flexible graph construction**
  - "Mixing and matching" optimization levels
- **Handling Tensor Primitives (e.g., conv, add)**
  - Optimized Path: Map the primitive to a specialized hardware unit or compute kernel
  - Fallback Path: Directly compile the provided SkriptND definition into vanilla compute shaders
- **Handling Compounds (e.g., conv_bn_relu)**
  - Optimized Path: Support the compound directly as a single "fused" operation
  - Fallback Path: "Lower" the compound, breaking it down into its constituent primitives
- **Implementers can choose what Primitives and Compounds to optimize**
  - Using custom hardware or compute kernels
  - The graph will always execute even if everything is compiled into standard computer shaders
- **Guaranteed Forward Compatibility**
  - New primitives always have a compiler path to compute shaders on programmable hardware
  - The model will always run, even if operation are not yet fully optimized

SkriptND Operator and Graph Definition Example

```
operator matmul {
        @input {
                A: real[m,k];
                B: real[k,n];
        }
        @output {
                C: real[m,n];
        }
        @lower {
                C[i,j] = 0.0,
                        i < m, j < n;
                C[i,j] += A[i,l] * B[l,j],
                        i < m, j < n, l < k;
        }
}
```

- *Einsum-like notation independent of HW details*
- *Does not require low level coding expertise to write*
- *Allows optimization to target HW*

```
graph AlexNet {
    @input {
        input: real[1,3,224,224];
    }
    @output {
        output: real[1,1000];
    }
    @variable {
        kernel1: real[64, 3, 11, 11];
        bias1: real[64];
        kernel2: real[192, 64, 5, 5];
        bias2: real[192];
        kernel3: real[384, 192, 3, 3];
        bias3: real[384];
        ...
        kernel7: real[4096, 4096];
        bias7: real[4096];
        kernel8: real[classes, 4096];
        bias8: real[classes];
    }
    @compose {
        conv1 = nn.conv{padding=0, stride=4}(input, kernel1, bias1);
        relu1 = nn.relu(conv1);
        pool1 = nn.max_pool{padding=0, size=3, stride=2}(relu1);
        conv2 = nn.conv{padding=2}(pool1, kernel2, bias2);
        relu2 = nn.relu(conv2);
        pool2 = nn.max_pool{padding=0, size=3, stride=2}(relu2);
        conv3 = nn.conv{padding=1}(pool2, kernel3, bias3);
        relu3 = nn.relu(conv3);
        ...
        relu6 = nn.relu(conv6);
        flat1 = layout.flatten{axis=1}(relu6);
        conv7 = nn.linear(flat1, kernel7, bias7);
        relu7 = nn.relu(conv7);
        conv8 = nn.linear(relu7, kernel8, bias8);
        output = nn.softmax(conv8);
    }
}
```

NNEF 2.0 and SkriptND Tooling

- **NNEF 2.0 specification preview is available on GitHub**
- **NNEF 2.0 Tools is a tooling ecosystem around NNEF 2.0 to get you started**
- **Network Conversion**
  - Importers from ONNX, TensorFlow, and TFLite
  - Exporters to ONNX and TensorFlow
- **Network Execution**
  - C++ code generator with parser, frontend and runtime
  - TVM based sample compiler which can compile NNEF to Vulkan, CUDA and CPU code.
- **Give it a try!**

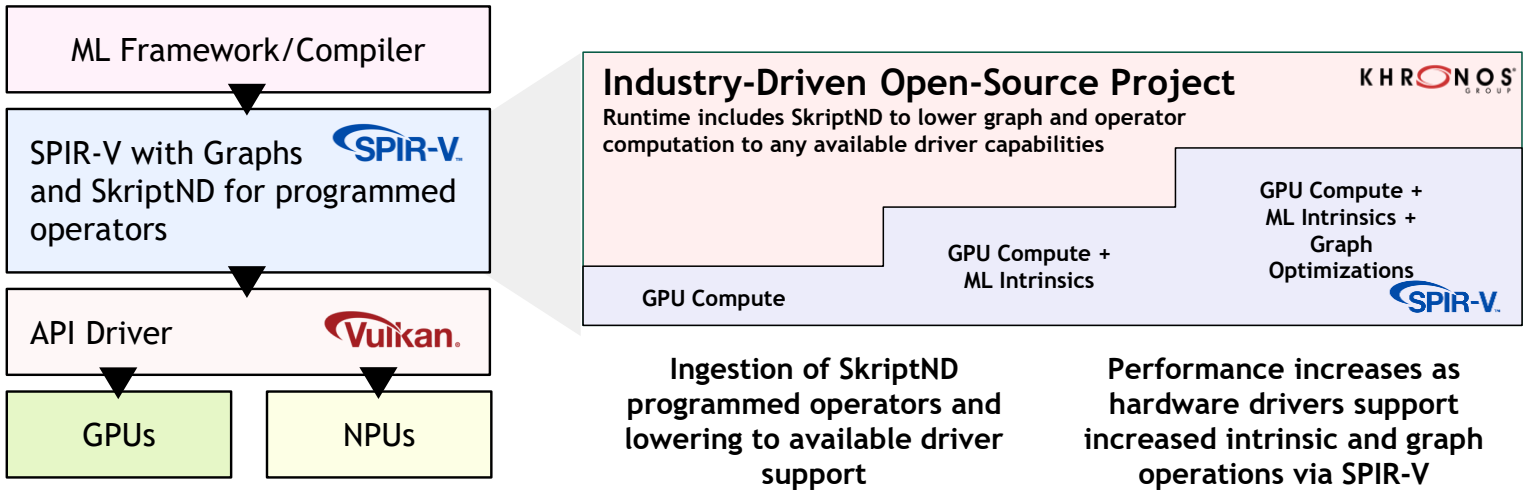
NNEF 2.0 Specification

NNEF 2.0 Tools

Should SkriptND be embedded into SPIR-V?

- **SkriptND could be embedded into SPIR-V graphs**
  - Custom operators become part of the graph
  - Drivers execute hardware targeted compilation for efficiency
- **Incremental support to build ecosystem**
  - Implement SkriptND as a Layered Vulkan Extension
  - IHVs can incrementally increase levels of optimization support over time

Should Khronos initiate an open source project to enable incremental industry rollout of SkriptND-based operators?



Ingestion of SkriptND programmed operators and lowering to available driver support

Performance increases as hardware drivers support increased intrinsic and graph operations via SPIR-V

Khronos proposes simplifying the industry's backend acceleration of ML stacks
We welcome your feedback on whether this meets real-world industry needs!

KHRONOS GROUP

Simplifying Cross-Platform Machine Learning Acceleration

SPIR-V Graphs
- Communicate graph information to driver
- Enables driver-level graph optimizations
- Unified driver with GPU and NPU support

SkriptND
- Operator programming
- Lowering to optimized hardware and shaders
- Forward compatibility for incremental rollout

Compute Shaders
- Add ML-needed data types and matrix/Tensor intrinsic operations to HAL acceleration APIs
- Open-source run-time enables incremental rollout