Gérald Rocher
Université Côte d'Azur, CNRS/INRIA Kairos
Mineure IOT-CPS 2025-2026

Challenges With Deep Learning

Fragmented tools and frameworks

- Multiple deep learning frameworks exist, each optimized for specific use cases
  - **PyTorch** flexible and research-friendly, widely used in computer vision and NLP.
  - **TensorFlow** production-ready, supports cloud, edge, browser, and even microcontrollers.
  - **Keras** high-level, user-friendly API for rapid prototyping, now tightly integrated with TensorFlow.
  - **Caffe2** lightweight and efficient, optimized for mobile and embedded deployment.
  - **ML.NET** designed for .NET ecosystem, ideal for enterprise applications and classical ML integration.

Lack of interoperability

- Deep learning models rarely transfer seamlessly across frameworks,
- Switching frameworks often means starting model development from scratch.

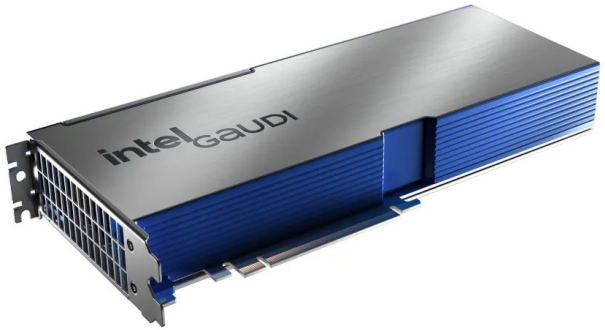
Today's AI models face the same interoperability challenges as programming languages did before standardized bytecode, Common Intermediate Language (CIL) and runtimes like JVM and CLR.

https://www.youtube.com/watch?v=cK5AyawZSUI

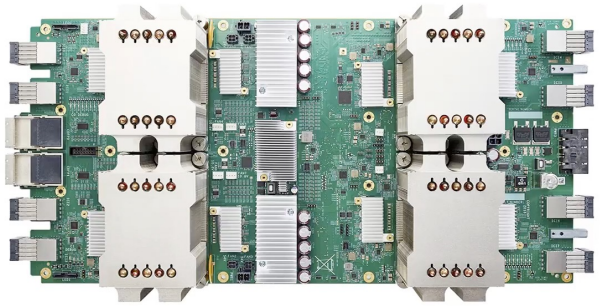
Challenges With Deep Learning

From the model learning perspective in Cloud or enterprise servers, different hardware accelerators compete…

Nvidia GPUs



Intel FPGAs (Agilex), GPUs (Arc), CPUs (Xeon) & ASICs (Gaudi)



Google Cloud TPUs
(TensorFlow, Accelerated Linear Algebra, XLA)

NVIDIA CUDA cuDNN

intel oneAPI HPC TOOLKIT

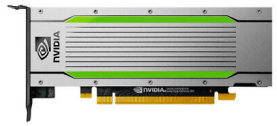
UXL Unified Acceleration Foundation

oneDNN

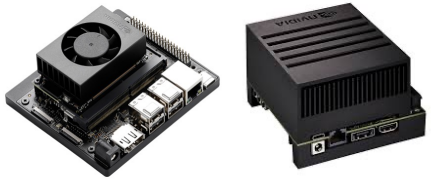
OpenXLA

Challenges With Deep Learning

From the inference perspective in the edge or in smart devices, different lower-end hardware accelerators compete…
A pipeline for models' simplification and optimization is needed!



Nvidia GPUs



Nvidia Edge/AIoT

Intel

Google Edge TPU

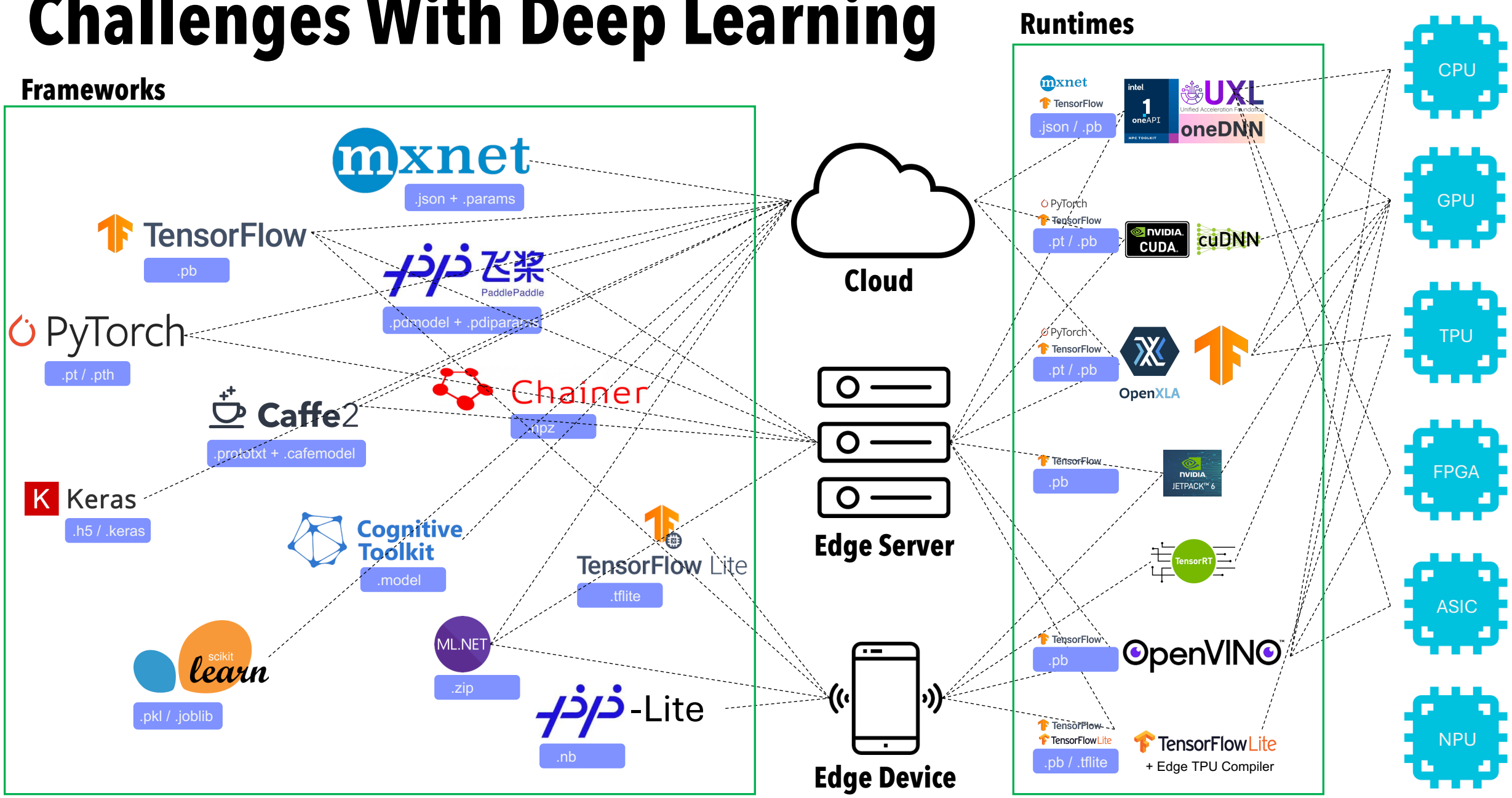
TensorRT

NVIDIA JETPACK™ 6

OpenVINO™

TensorFlow Lite
+ Edge TPU Compiler

Challenges With Deep Learning



Open Neural Network eXchange (ONNX)

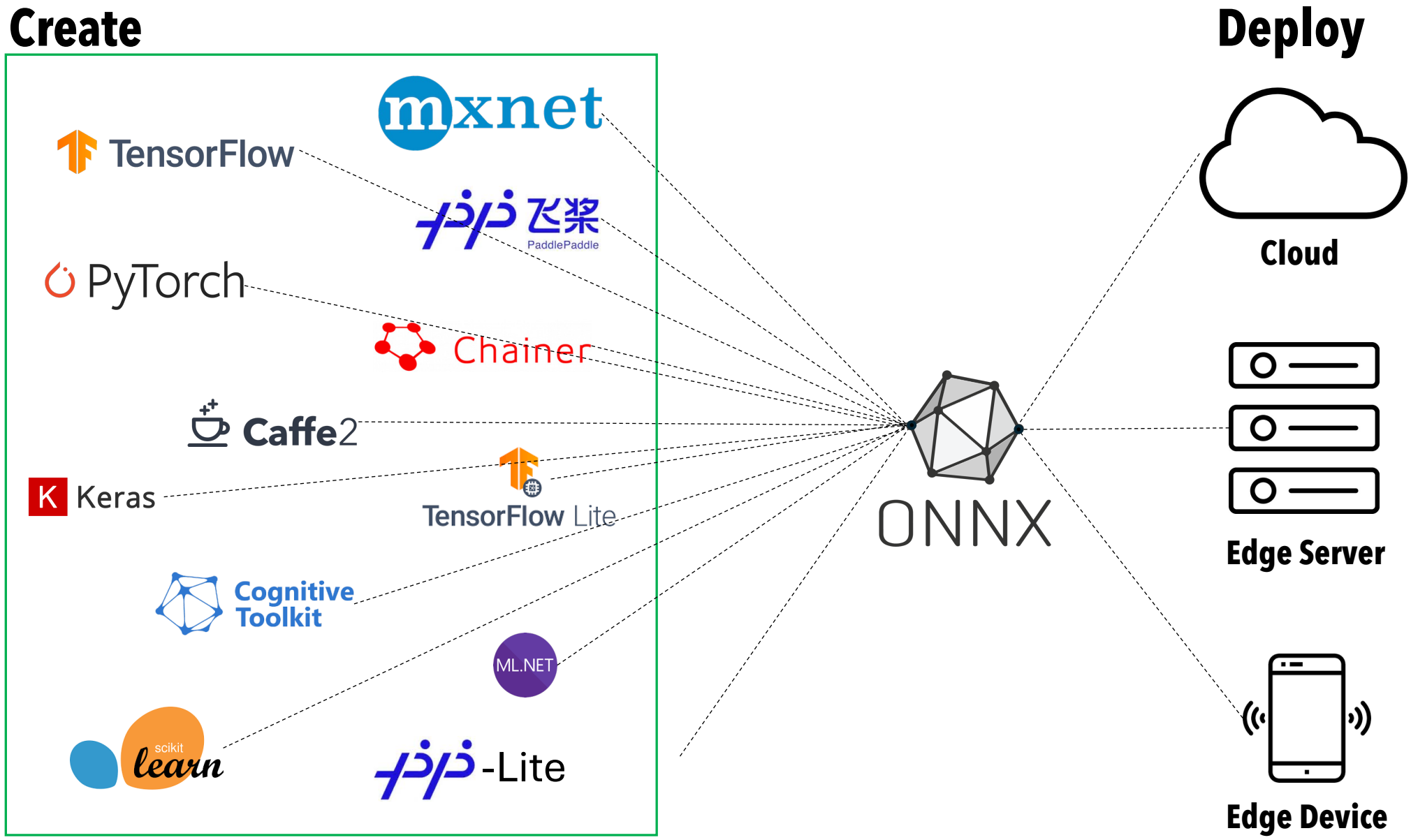
- **ONNX is an open standard** format for AI models, enabling transfer and execution across different frameworks and hardware.
- Supports both deep learning and classical ML models.

- Originated in 2017 by AWS, Microsoft, and Facebook.

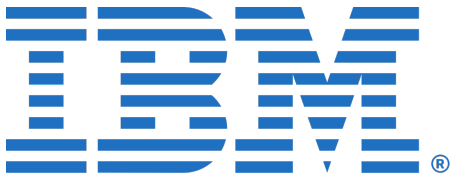
- **ONNX specification includes**
  - A file format (.onnx),
  - A standardized set of operators.

- **ONNX Runtime** is the **high-performance execution engine** for running models in the ONNX format, optimized for different hardware (CPU, GPU, FPGA, mobile, etc.) and supporting multiple acceleration backends like CUDA, TensorRT, oneDNN, and OpenVINO.

ONNX Vision


Create
mxnet
TensorFlow
飞桨
PaddlePaddle
PyTorch
Chainer
Caffe2
Keras
TensorFlow Lite
Cognitive Toolkit
ML.NET
scikit learn
-Lite
ONNX
Deploy
Cloud
Edge Server
Edge Device

ONNX Ecosystem



ONNX Design Principles

Supports both deep learning and traditional machine learning models (Linear models, Tree-based models, Support Vector Machines, Clustering algorithms, etc.),

Flexible and **extensible** to keep pace with rapid AI advancements,

Compact, cross-platform format for model serialization,

- To deep learning what the CIL (Common Intermediate Language) or Java bytecode are to programming languages.

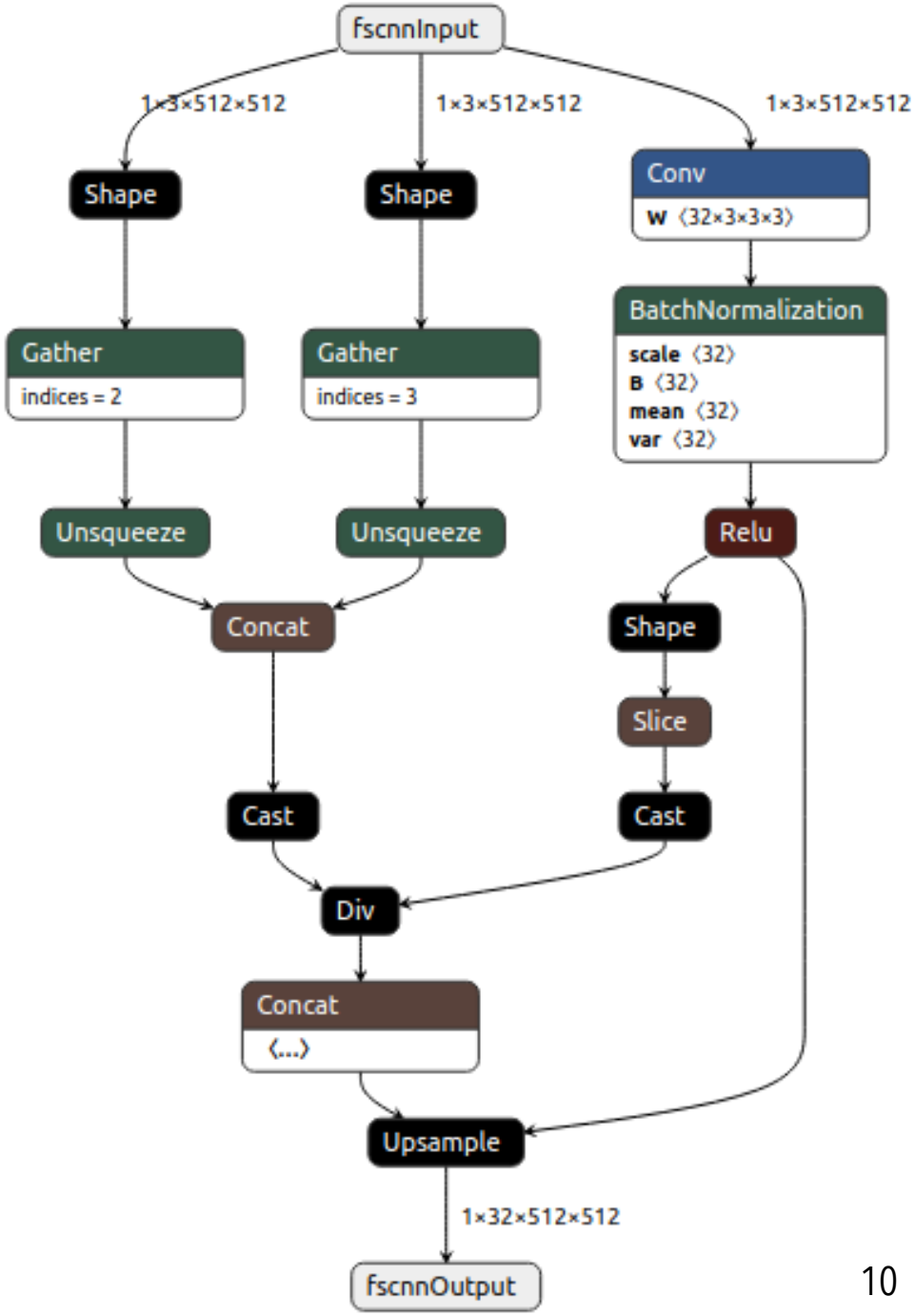
Standardized set of well-defined operators, informed by real-world usage.

Now part of Linux Foundation (OLF AI https://lfai.foundation)

ONNX File Format

A PORTABLE INDEPENDENT FORMAT TO DEFINE A MODEL GRAPH

- **Model**
  - Version info,
  - Metadata,
  - **Acyclic computation dataflow graph**.
- **Graph**
  - Graph name,
  - Inputs/Outputs,
  - List of computation nodes.
- **Computation Node**
  - Zero or more inputs of defined types,
  - One or more outputs of defined types,
  - Operator,
  - Operator parameters.



ONNX Data Types

Tensor Type

- **Element types**
  - `Int8, int16,int32,int64`
  - `uint8,uint16,uint32,uint64`
  - `Float16,float,double`
  - `Bool,`
  - `String`
  - `Complex64,complex128`

Non-tensor types in ONNX-ML

- **Maps** – key–value collections
  - `map<string, float>`
  - `map<int64, float>`
- **Sequences** – ordered lists of elements
  - `sequence<T>` where T can be a tensor or a map.
- **Strings** – used for categorical features or labels in classical ML models.

These types allow ONNX-ML operators (like `LabelEncoder`, `OneHotEncoder`, `StringNormalizer`, `TreeEnsembleClassifier`, etc.) to work directly with non-numeric, structured data.

ONNX Operators

An Operator is identified by `<name,domain,version>`

- **Core Ops (both ONNX and ONNX-ML)**
  - https://onnx.ai/onnx/operators/ ,
  - Currently (v1.20) 124 operators in `ai.onnx` domain and 19 in `ai.onnx.ml`
  - Should be supported by ONNX-compatible frameworks,
  - Support many scenarios/problem area including image classification, recommendation, NLP, etc.

- **Custom Ops**
  - Specific to framework or runtime,
  - Indicated by a custom domain name,
  - Primarily meant to be a safety-valve.

Ways to Get ONNX Models

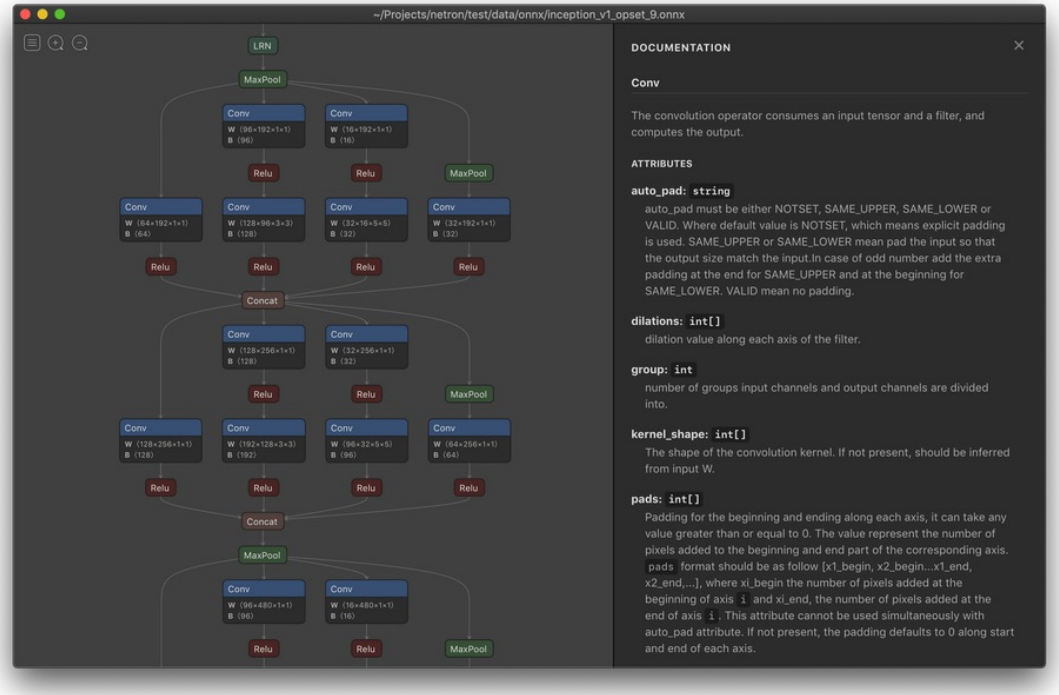
- **ONNX Model Zoo**
  - http://github.com/onnx/models

- **Convert Existing Models**

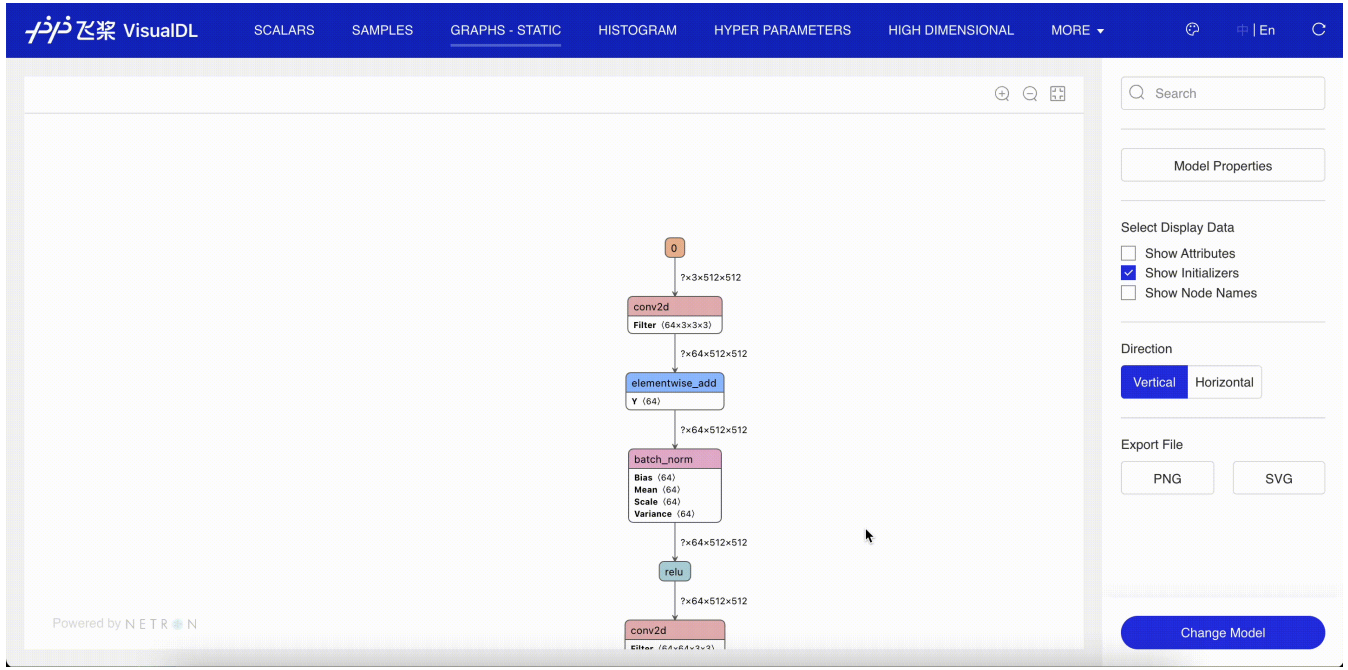
- **Train models from ground up**
  - Azure Machine Learning (Microsoft),
  - AWS SageMaker,
  - Etc.

ONNX Tools

NETRON



Visual DL



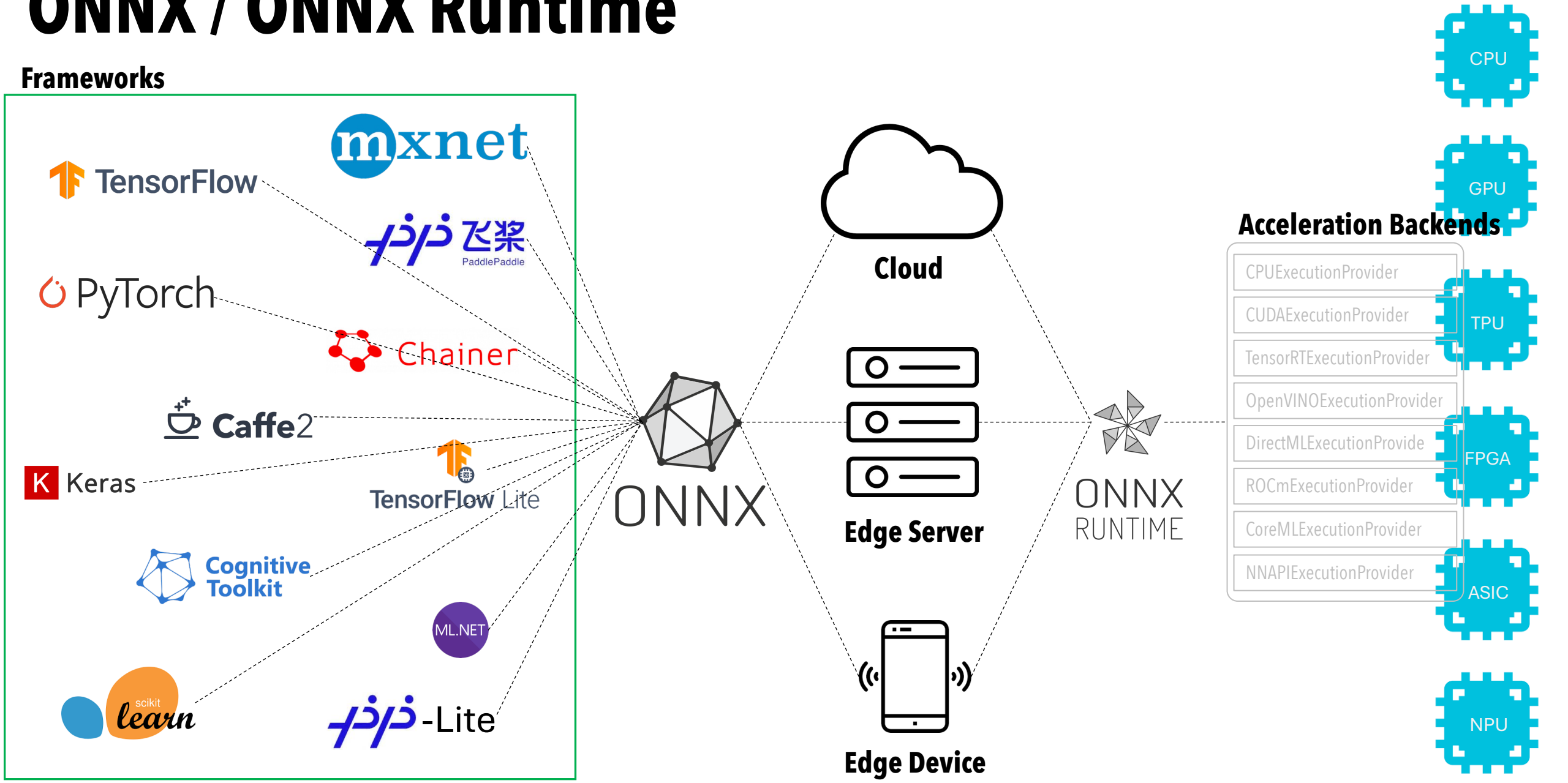
ONNX Runtime

- **High Performance runtime for executing ONNX models**
  - Supports inference and training acceleration,
  - Cross platform, cross architecture, and cross language
    - Python, C/C++, C#, Java, JavaScript, etc.
  - Orchestrates ONNX graph execution and maps to optimized code implementations (through ONNX Execution Providers),
- **Open source (MIT license), founded by Microsoft,**
- **Full ONNX specification support (v1.2+)**

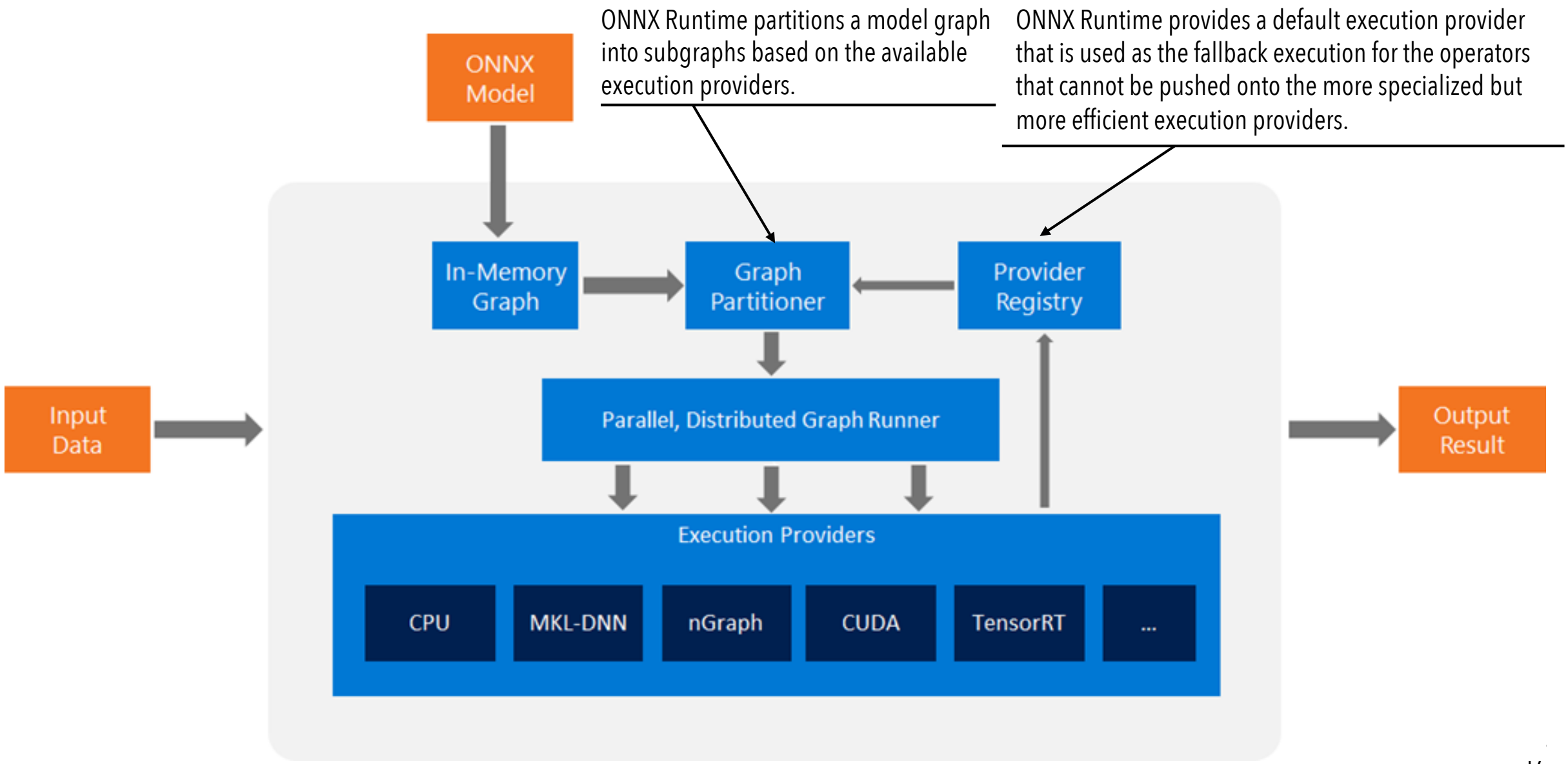
- ONNX Runtime is not part of Linux Foundation (separate from ONNX governance).

- **ONNX Runtime is to ONNX models what CLR/JVM are to compiled programming languages.**

ONNX / ONNX Runtime

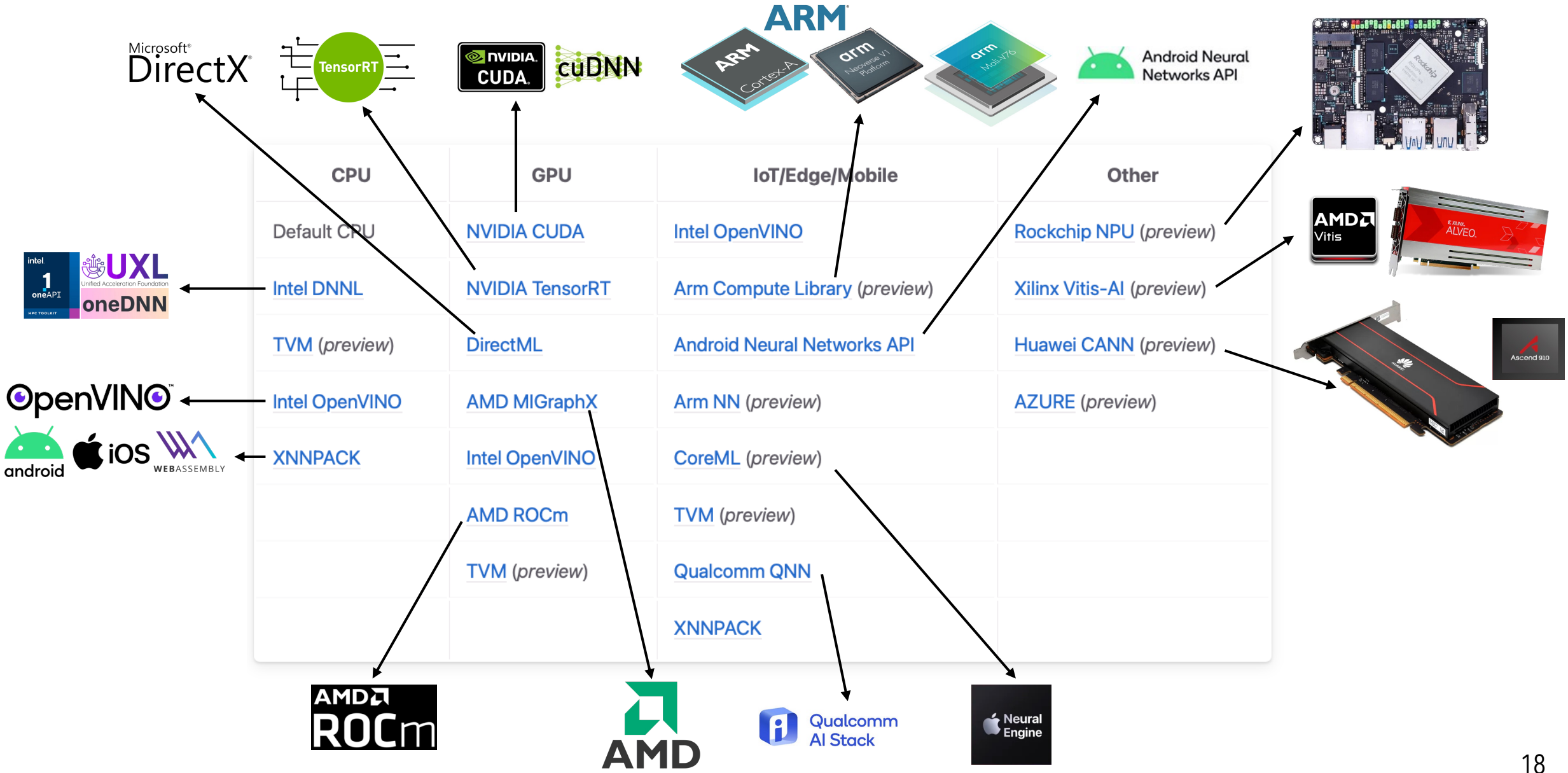


ONNX Runtime Architecture



https://onnxruntime.ai/docs/reference/high-level-design.html#high-level-system-architecture

ONNX Runtime Supported Execution Providers



https://onnxruntime.ai/docs/execution-providers/

ONNX Runtime Graph Optimizations

Graph optimizations are divided into three levels (enabled by default)

- **Basic**
  - Semantics-preserving graph rewrites which remove redundant nodes and redundant computation,
  - Run before graph partitioning and thus apply to all the execution providers
- **Extended**
  - These optimizations include complex node fusions,
  - They are run after graph partitioning and are only applied to the nodes assigned to the CPU or CUDA or AMD ROCm execution providers.
- **Layout Optimizations**
  - These optimizations change the data layout for applicable nodes to achieve higher performance improvements,
  - They are run after graph partitioning and are only applied to nodes assigned to CPU execution provider.

https://onnxruntime.ai/docs/performance/model-optimizations/graph-optimizations.html

ONNX Runtime Extensions

- A library of custom operators for common pre/post processing
- https://onnxruntime.ai/docs/extensions/
- Useful for audio, vision, text and language models.

ONNX Runtime Adoption