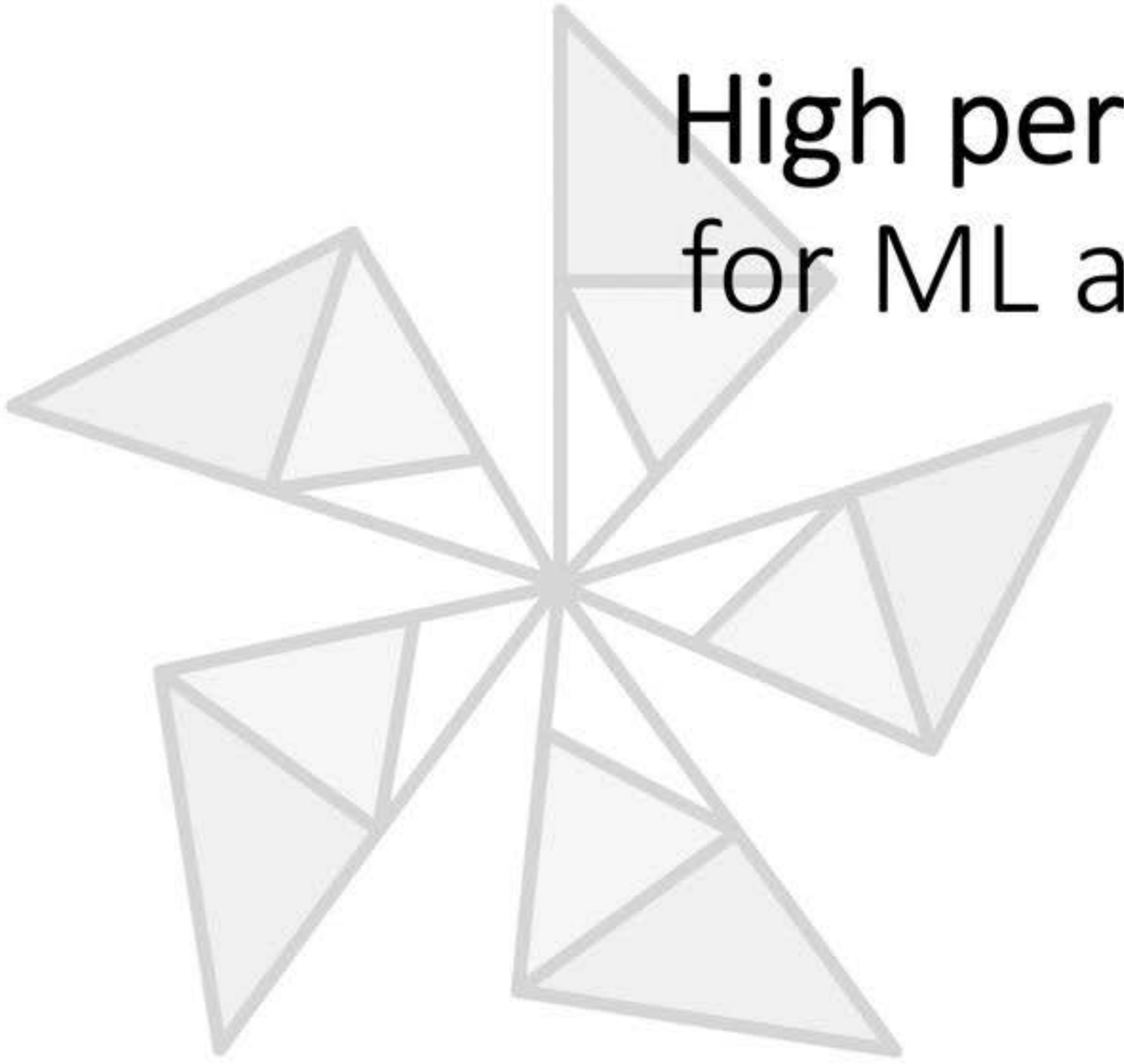
AUGUST 2019

High performance inferencing for ML and DNN models with ONNX Runtime



Agenda

INTRO TO THE ONNX ECOSYSTEM

PRODUCTION USAGE

ONNX TECHNICAL DESIGN

ONNX RUNTIME TECHNICAL DESIGN

Agenda

INTRO TO THE
ONNX ECOSYSTEM

Agenda

INTRO TO THE
ONNX ECOSYSTEM

PRODUCTION
USAGE

Agenda

INTRO TO THE ONNX ECOSYSTEM

PRODUCTION USAGE

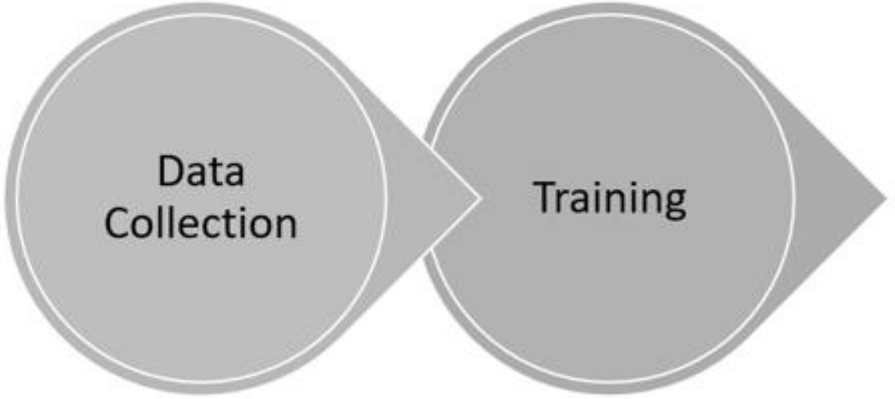
ONNX TECHNICAL DESIGN

Intro to the ONNX Ecosystem

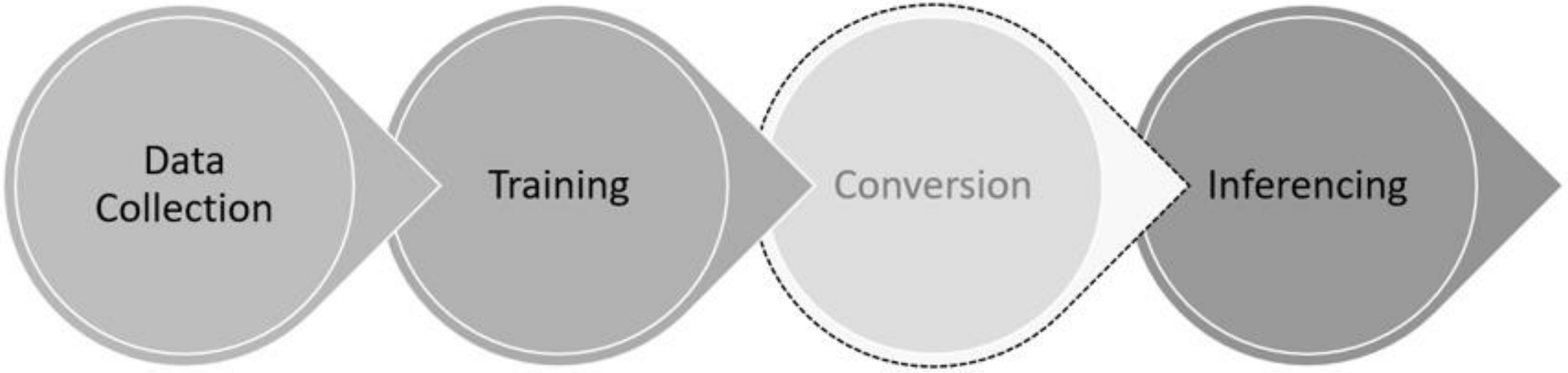
ML Models: Research to Production

ML Models: Research to Production

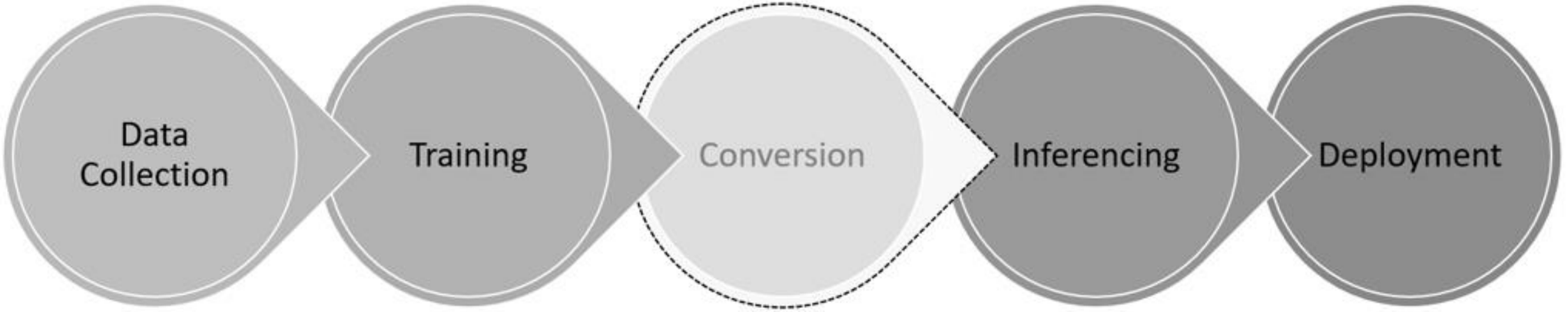
ML Models: Research to Production



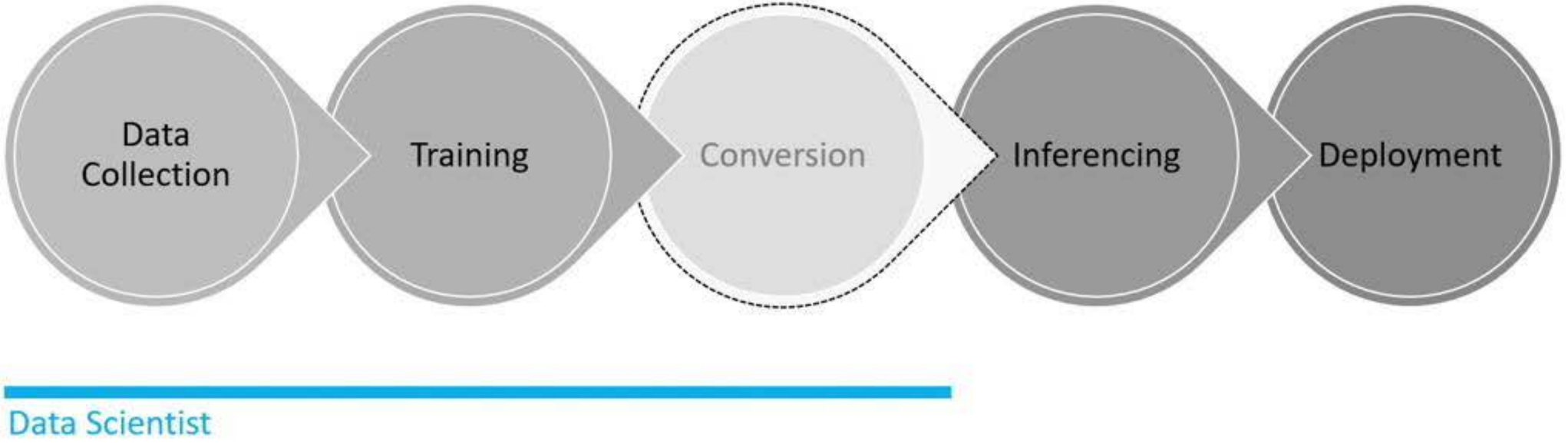
ML Models: Research to Production



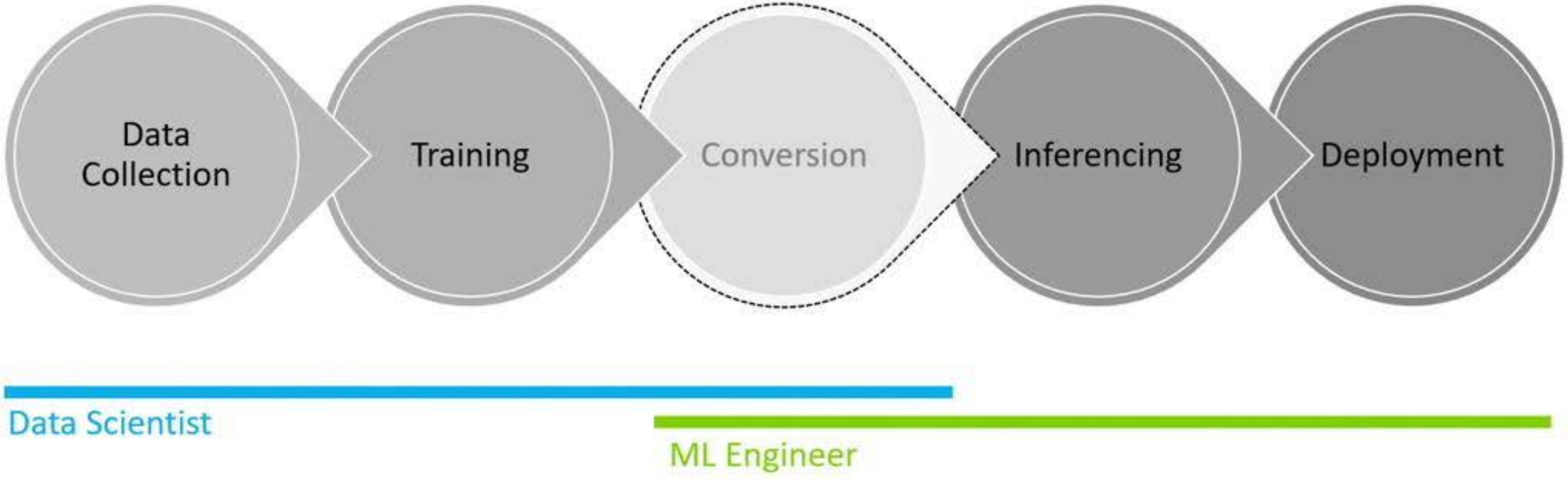
ML Models: Research to Production



ML Models: Research to Production



ML Models: Research to Production



Many product teams want to use ML

Reality

Reality

Training Framework

PyTorch

K Keras

Cognitive Toolkit

ML.NET

Caffe

scikit learn

Reality

Training Framework

PyTorch

Keras

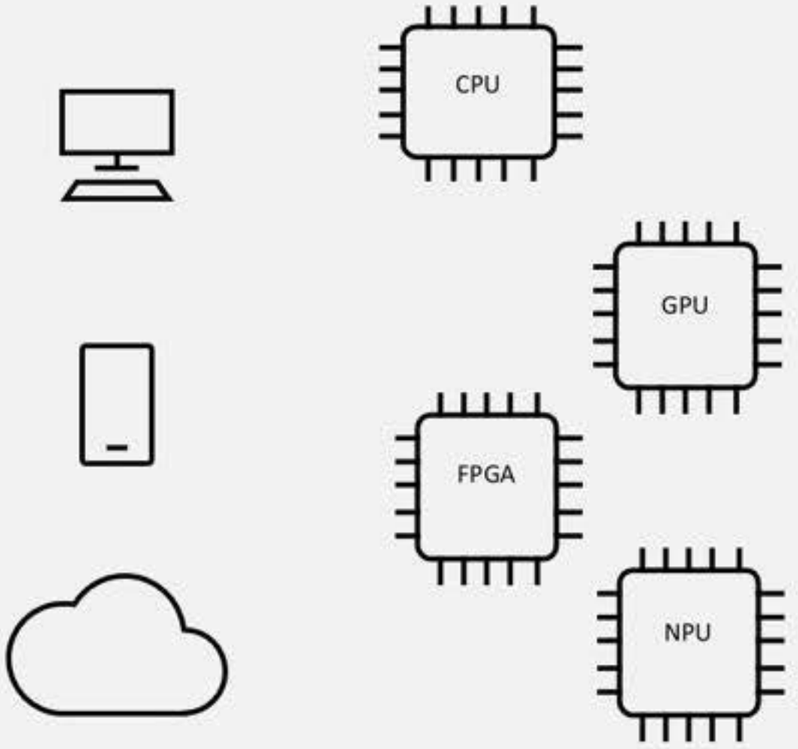
Cognitive Toolkit

ML.NET

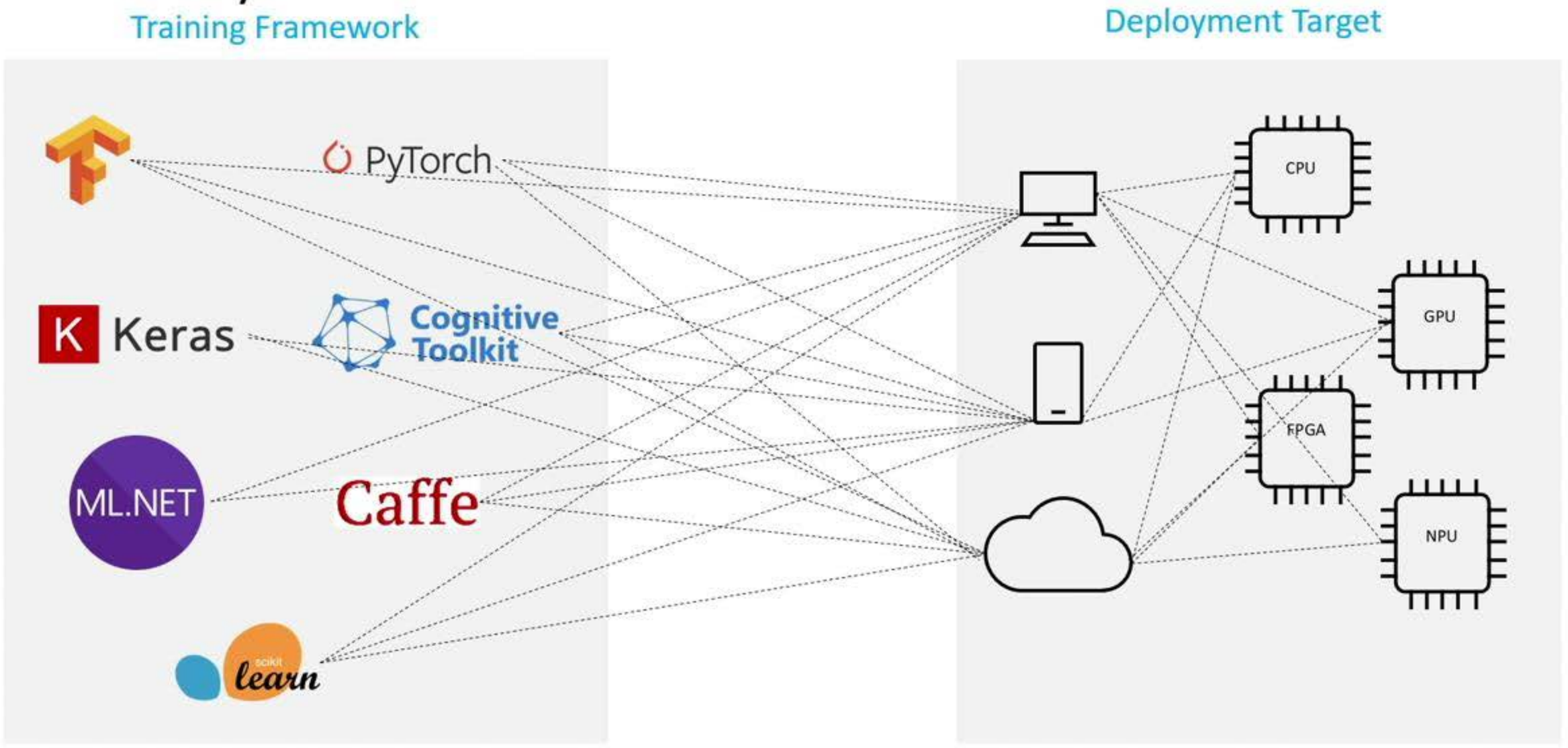
Caffe

scikit learn

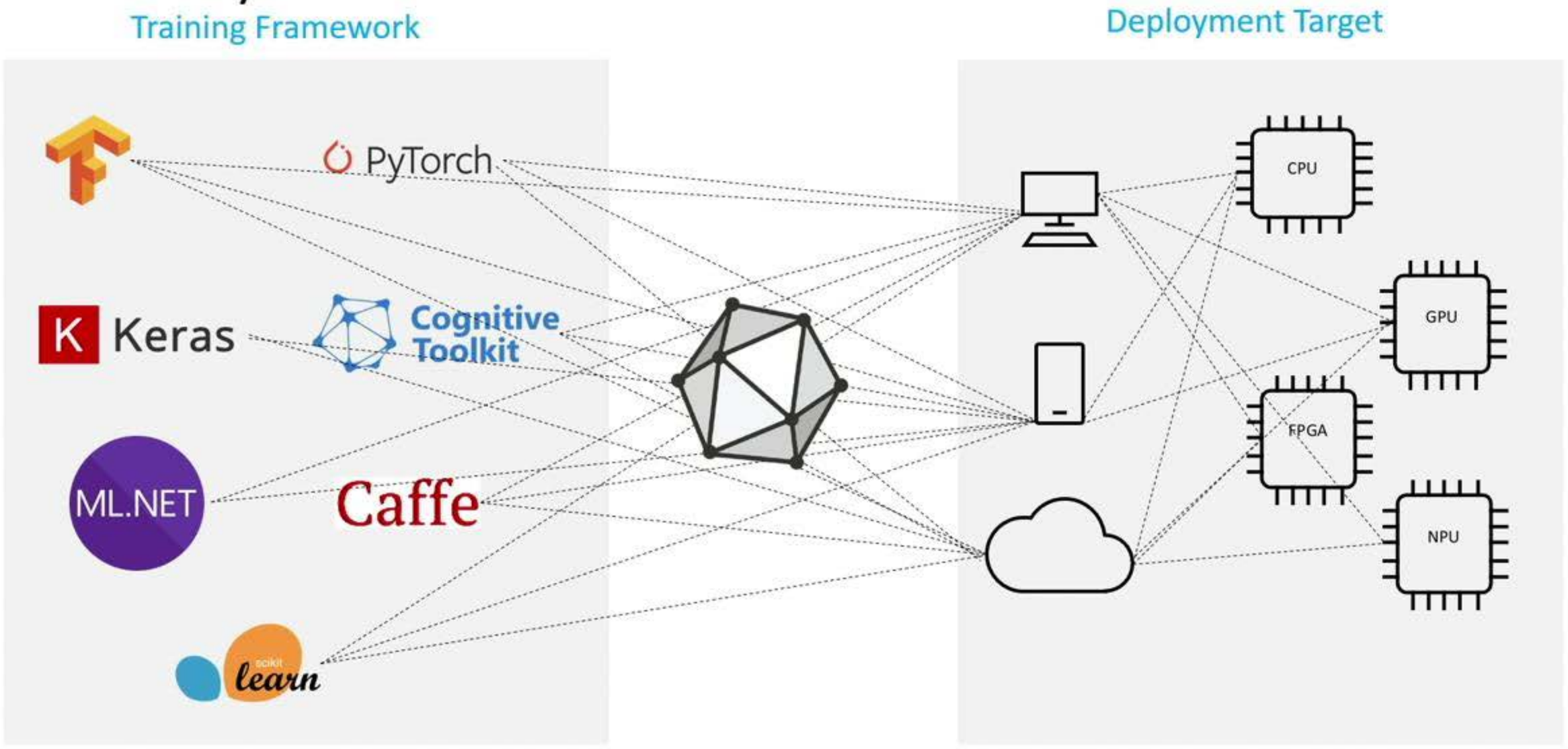
Deployment Target



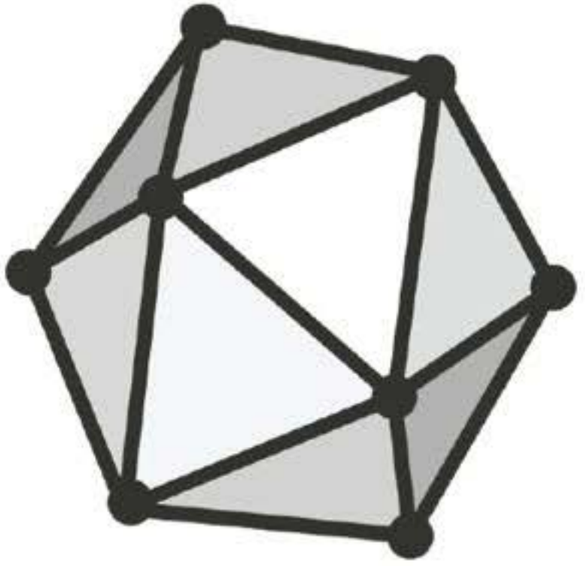
Reality


Training Framework
Deployment Target
PyTorch
K Keras
Cognitive Toolkit
ML.NET
Caffe
scikit learn
CPU
GPU
FPGA
NPU

Reality

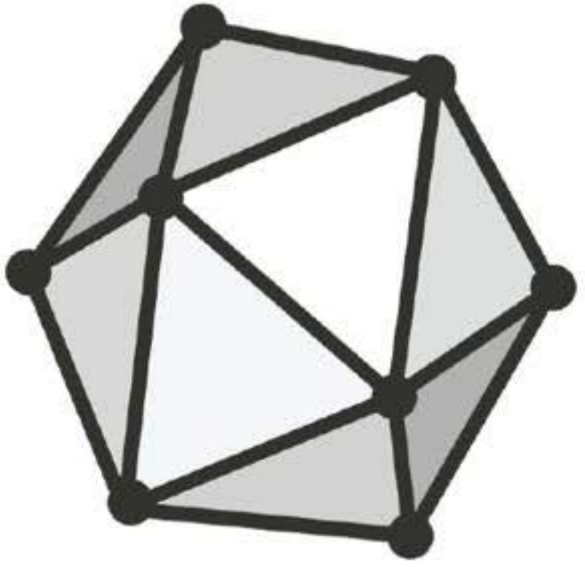

Training Framework
Deployment Target
PyTorch
Keras
Cognitive Toolkit
ML.NET
Caffe
scikit learn
CPU
GPU
FPGA
NPU

ONNX



ONNX

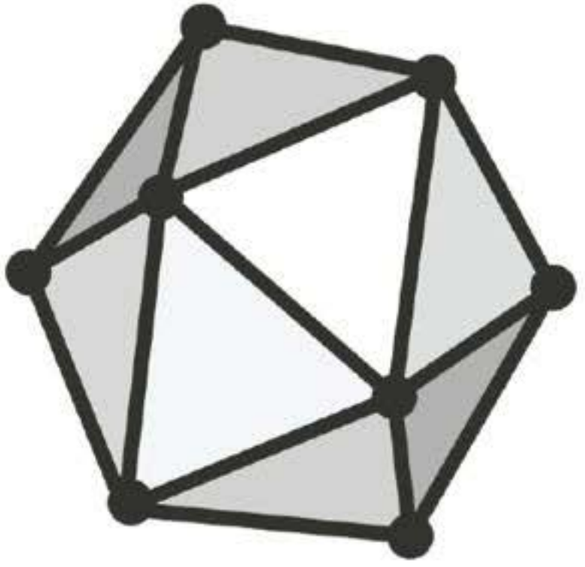
OPEN NEURAL NETWORK EXCHANGE



ONNX

OPEN NEURAL NETWORK EXCHANGE

Interoperable standard format for AI models, consisting of a common Intermediate Representation (IR) + Full operator spec

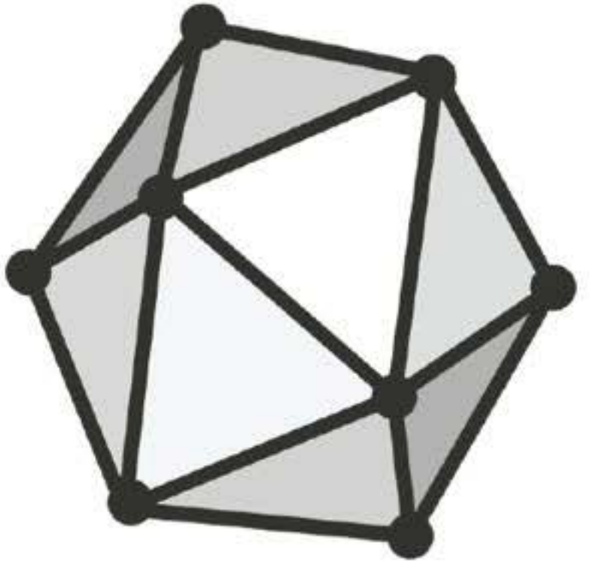


ONNX

OPEN NEURAL NETWORK EXCHANGE

Interoperable standard format for AI models, consisting of a common Intermediate Representation (IR) + Full operator spec

Alibaba Group · AMD · arm · aws · Baidu 百度 · BITMAIN · CEVA · Facebook Open Source · habana · Hewlett Packard Enterprise · HUAWEI · IBM · intel AI · MathWorks · MEDIATEK · Microsoft · Neural Network Libraries · mi · NVIDIA · NXP · Oath: · Preferred Networks · Qualcomm · sas · skymizer · SYNOPSYS · Tencent · unity



ONNX Open Governance

Steering Committee

Prasanth Pulavarthi (Microsoft)

Joe Spisak (Facebook)

Vin Sharma (Amazon)

Harry Kim (Intel)

Dilip Sequeira (Nvidia)

SIG (special interest group)

Architecture/Infrastructure

Lu Fang (Facebook)
and Ke Zhang (Microsoft)

Operators

Michał Karzyński (Intel)
and Emad Barsoum (Microsoft)

Converters

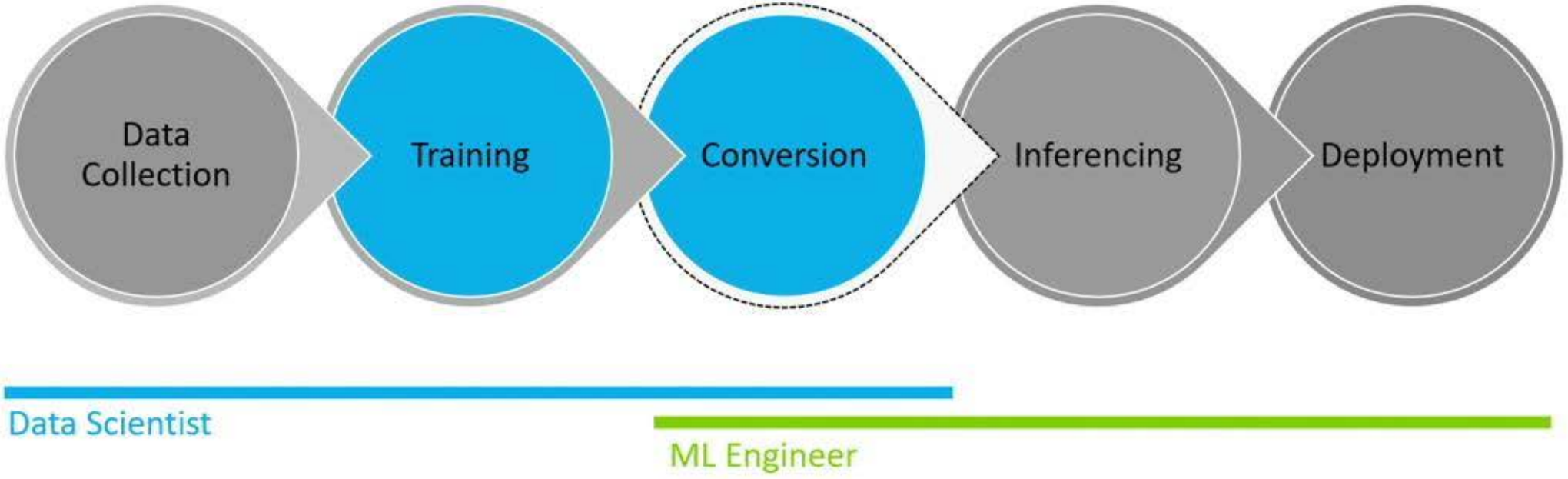
Chin Huang (IBM)
and Guenther Schmuelling (Microsoft)

Working Groups

Training

Edge/Mobile

ML Models: Research to Production



How do I get an ONNX model?

How do I get an ONNX model?

- ONNX Model Zoo

How do I get an ONNX model?

- ONNX Model Zoo
- Model creation services such as Azure Custom Vision and/or AutoML

How do I get an ONNX model?

- ONNX Model Zoo
- Model creation services such as Azure Custom Vision and/or AutoML
- Convert an existing models from another framework

How do I get an ONNX model?

- ONNX Model Zoo
- Model creation services such as Azure Custom Vision and/or AutoML
- Convert an existing models from another framework

- End to End model training via systems such as Azure Machine Learning service

Open Source converters for popular frameworks

- **Tensorflow:** onnx/tensorflow-onnx
- **Keras:** onnx/keras-onnx
- **Scikit-learn:** onnx/sklearn-onnx
- **CoreML:** onnx/onnxmltools
- **LightGBM:** onnx/onnxmltools
- **LibSVM:** onnx/onnxmltools
- **XGBoost:** onnx/onnxmltools
- **SparkML** (alpha): onnx/onnxmltools

Native export

- **Pytorch**
- **CNTK**

LightGBM

LibSVM

dmlc
XGBoost

Spark

PyTorch

Cognitive Toolkit

Examples: Model Conversion

Examples: Model Conversion

PyTorch

```
import torch
import torch.onnx

model = torch.load("model.pt")

sample_input = torch.randn(1, 3, 224, 224)

torch.onnx.export(model, sample_input, "model.onnx")
```

Examples: Model Conversion

PyTorch

```
import torch
import torch.onnx

model = torch.load("model.pt")

sample_input = torch.randn(1, 3, 224,

torch.onnx.export(model, sample_input, "model.onnx")
```

Chainer

```
import numpy as np
import chainer
from chainer import serializers
import onnx_chainer

serializers.load_npz("my.model", model)

sample_input = np.zeros((1, 3, 224, 224), dtype=np.float32)
chainer.config.train = False

onnx_chainer.export(model, sample_input, filename="my.onnx")
```

Examples: Model Conversion

```
from keras.models import load_model
import keras2onnx
import onnx

keras_model = load_model("model.h5")

onnx_model = keras2onnx.convert_keras(keras_model,
keras_model.name)

onnx.save_model(onnx_model, 'model.onnx')
```

```
import torch
import torch.onnx

model = torch.load("model.pt")

sample_input = torch.randn(1, 3, 224,

torch.onnx.export(model, sample_input, "model.onnx")
```

```
serializers

serializers.load_npz("my.model", model)

sample_input = np.zeros((1, 3, 224, 224), dtype=np.float32)
chainer.config.train = False

onnx_chainer.export(model, sample_input, filename="my.onnx")
```

Examples: Model Conversion

```
from keras.models import load_model
import keras2onnx
import onnx

keras_model = load_model("model.h5")

onnx_model = keras2onnx.convert_keras(keras_model,
keras_model.name)

onnx.save_model(onnx_model, 'model.onnx')
```

```
python -m tf2onnx.convert
        --input frozen_model.pb
        --inputs input_batch:0, lengths:0
        --outputs top_k:1
        --fold_const
        --opset 8
        --output deepcc.onnx
```

Chainer

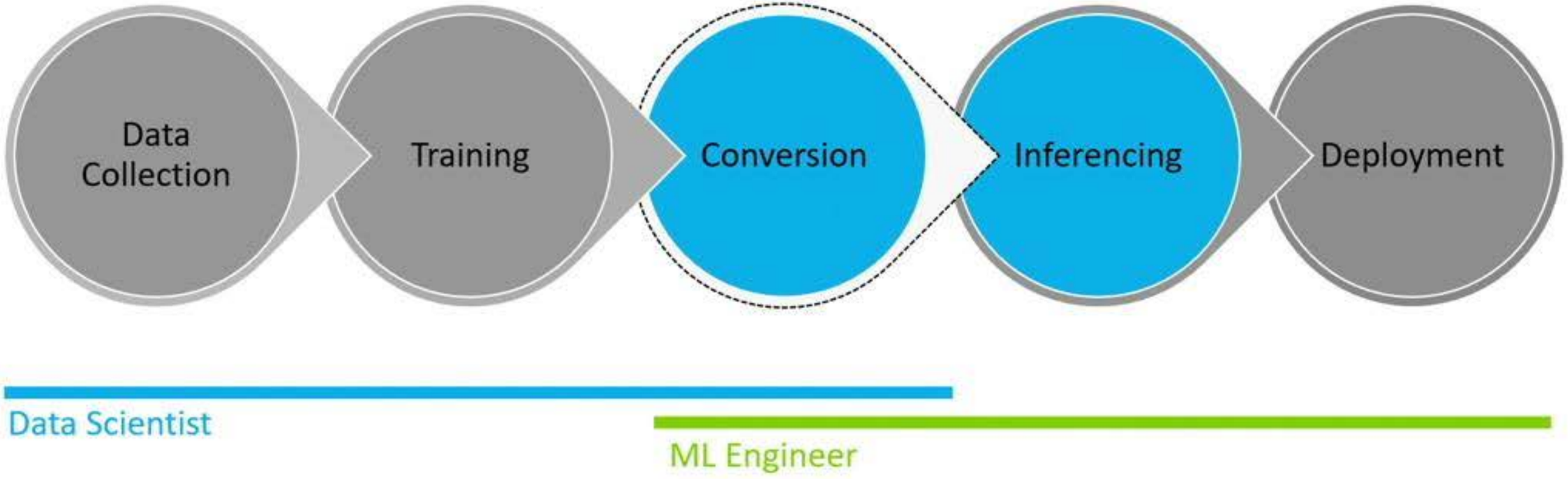
```
serializers
```

```
serializers.load_npz("my.model", model)

sample_input = np.zeros((1, 3, 224, 224), dtype=np.float32)
chainer.config.train = False

onnx_chainer.export(model, sample_input, filename="my.onnx")
```

PyTorch

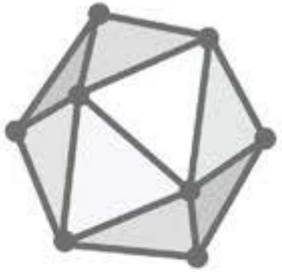
```
import torch
import torch.onnx

model = torch.load("model.pt")

sample_input = torch.randn(1, 3, 224,

torch.onnx.export(model, sample_input, "model.onnx")
```

ML Models: Research to Production



ONNX Runtime

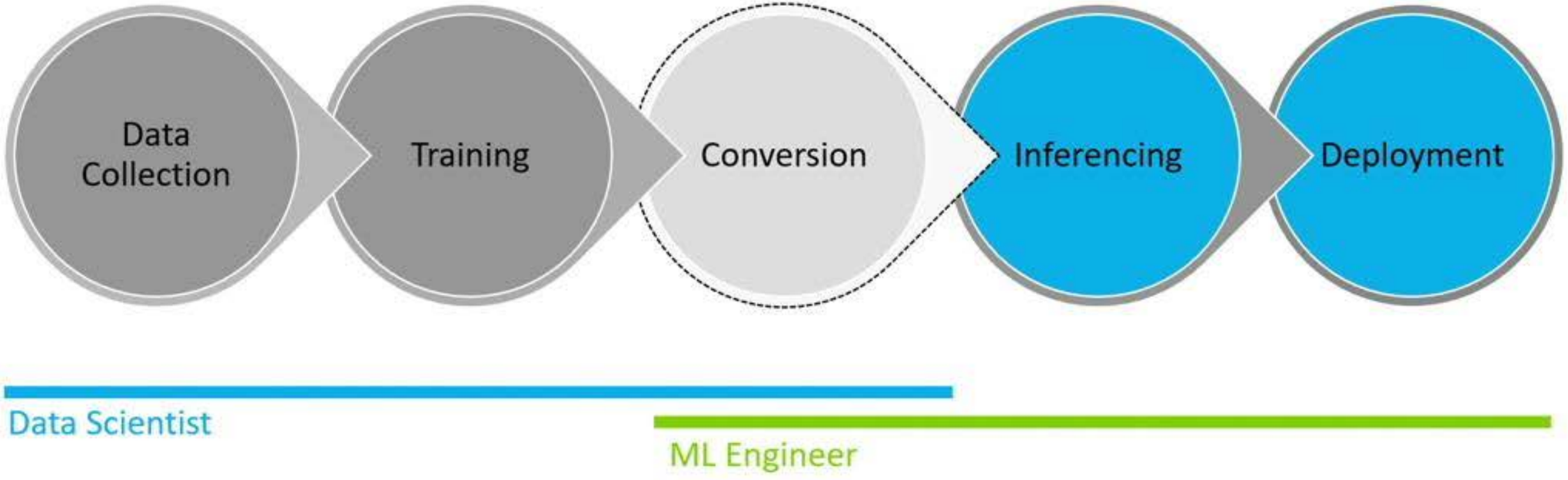
- https://microsoft.github.io/onnxruntime/
- High Performance **Inference Engine** for ONNX models
- Open sourced under MIT license
- Full ONNX spec support (v1.2+)
  - Covers both ONNX and ONNX-ML domain model spec and operators
  - Backwards and forwards compatible
- Extensible and modular framework
  - Clear API for plug-in graph optimizers, operators, and hardware accelerators
- Ships inbox on Windows 10 (RS5+) as WinML



Supported Architectures/Languages/Providers

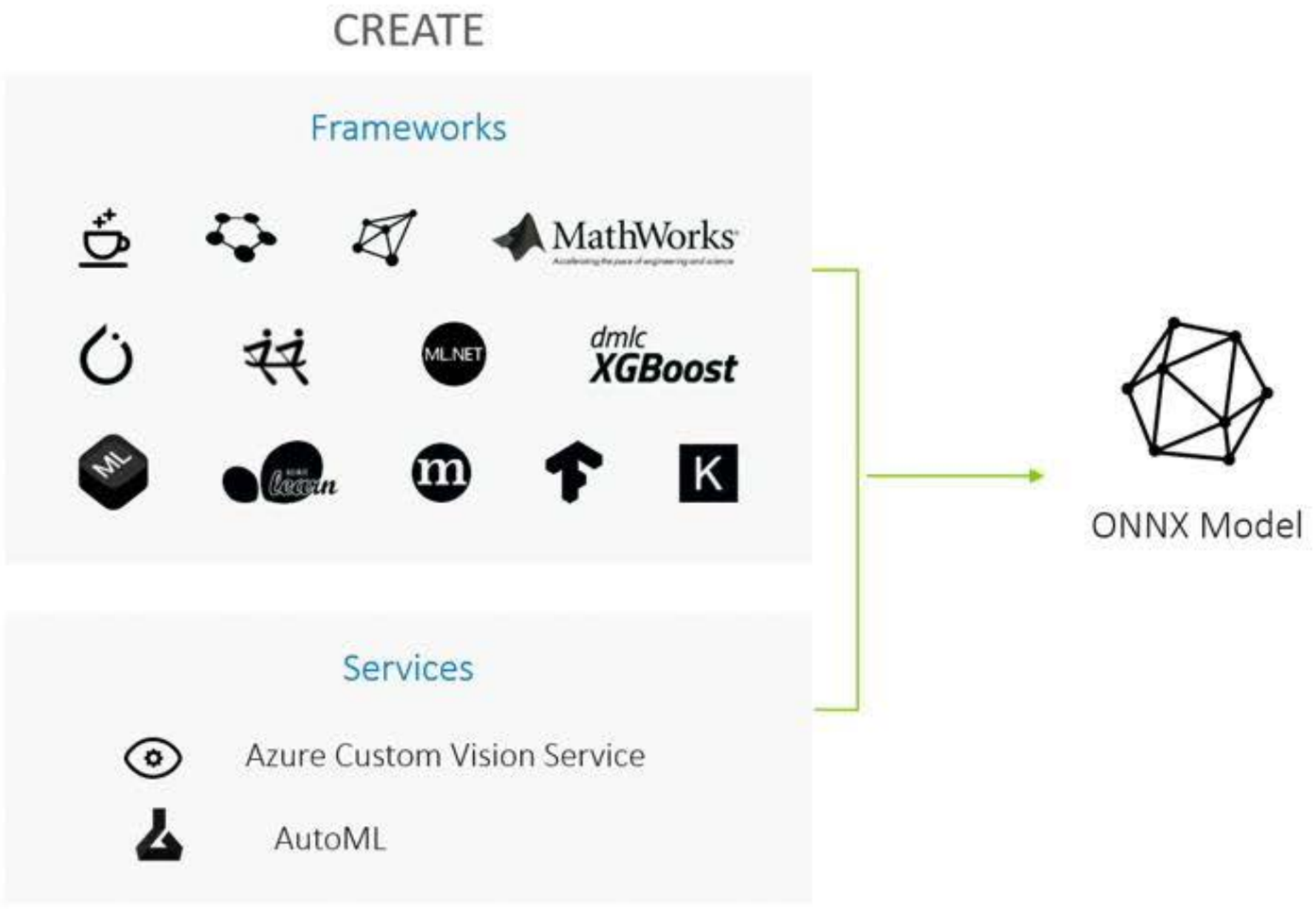
OS	Windows	Linux	Mac		
Language	Python (3.5-3.7)	C++	C#	C	
Architecture	X64	X86	ARM64	ARM32	
Hardware Acceleration	DefaultCPU	CUDA	TensorRT	DirectML	MKL-DNN
	MKL-ML	nGraph	NUPHAR	OpenVINO	
Installation Instructions	pip install onnxruntime				

ML Models: Research to Production

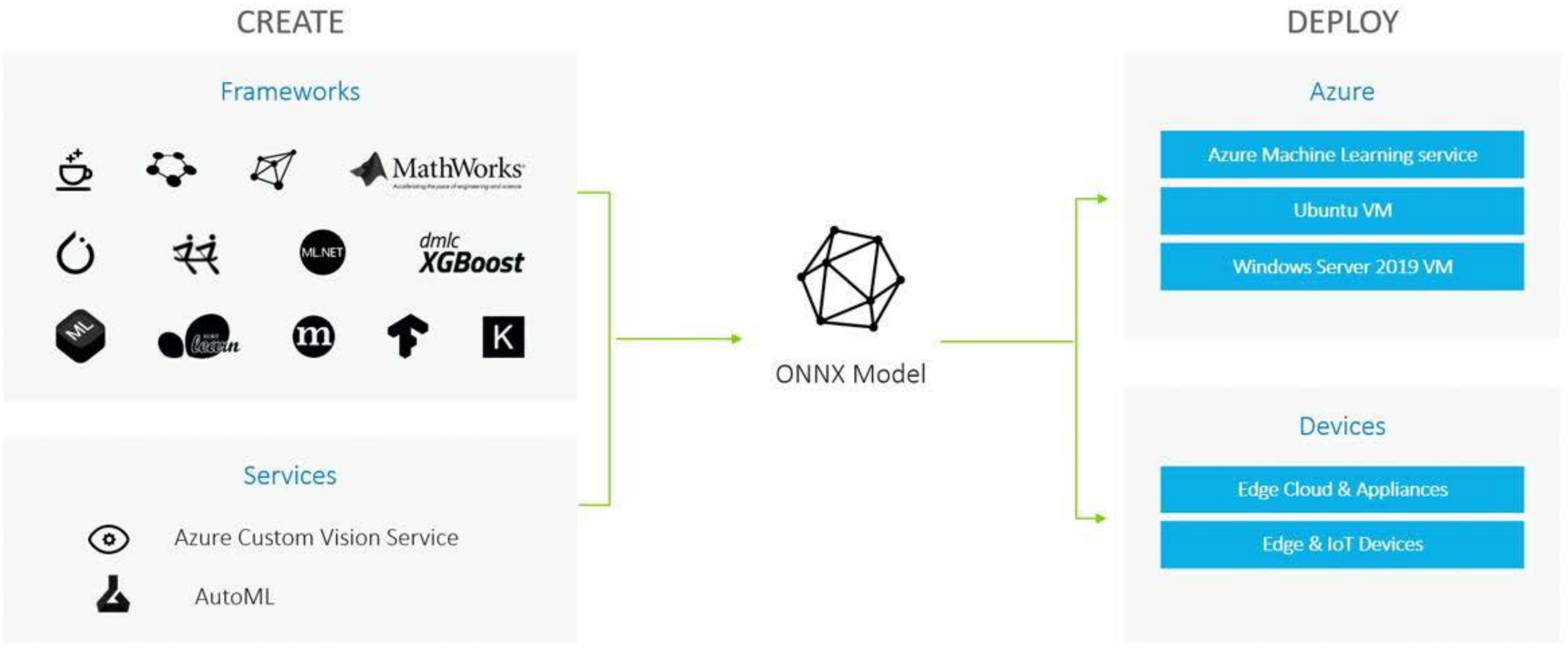


Deployment

Deployment


CREATE
Frameworks
MathWorks
ML.NET
dmlc
XGBoost
ML
Services
Azure Custom Vision Service
AutoML
ONNX Model

Deployment


CREATE
Frameworks
MathWorks
ML.NET
dmlc
XGBoost
Services
Azure Custom Vision Service
AutoML
ONNX Model
DEPLOY
Azure
Azure Machine Learning service
Ubuntu VM
Windows Server 2019 VM
Devices
Edge Cloud & Appliances
Edge & IoT Devices

ONNX @ Microsoft

ONNX @ Microsoft

PLATFORMS

AzureML

WinML

ML.Net

ONNX @ Microsoft

PLATFORMS

AzureML

WinML

ML.Net

PRODUCTS

Power BI

+ more

ONNX @ Microsoft

64+
models in production

PLATFORMS


AzureML


WinML

ML.Net

PRODUCTS

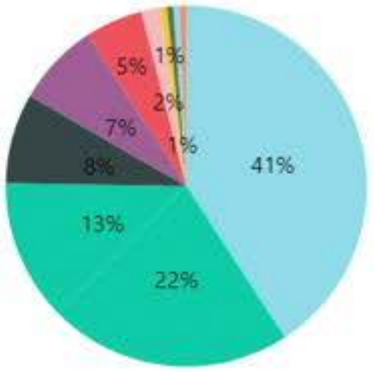

Power BI

+ more

ONNX @ Microsoft

PLATFORMS

AzureML WinML ML.Net



64+ models in production

on average ~3x perf improvement

PRODUCTS

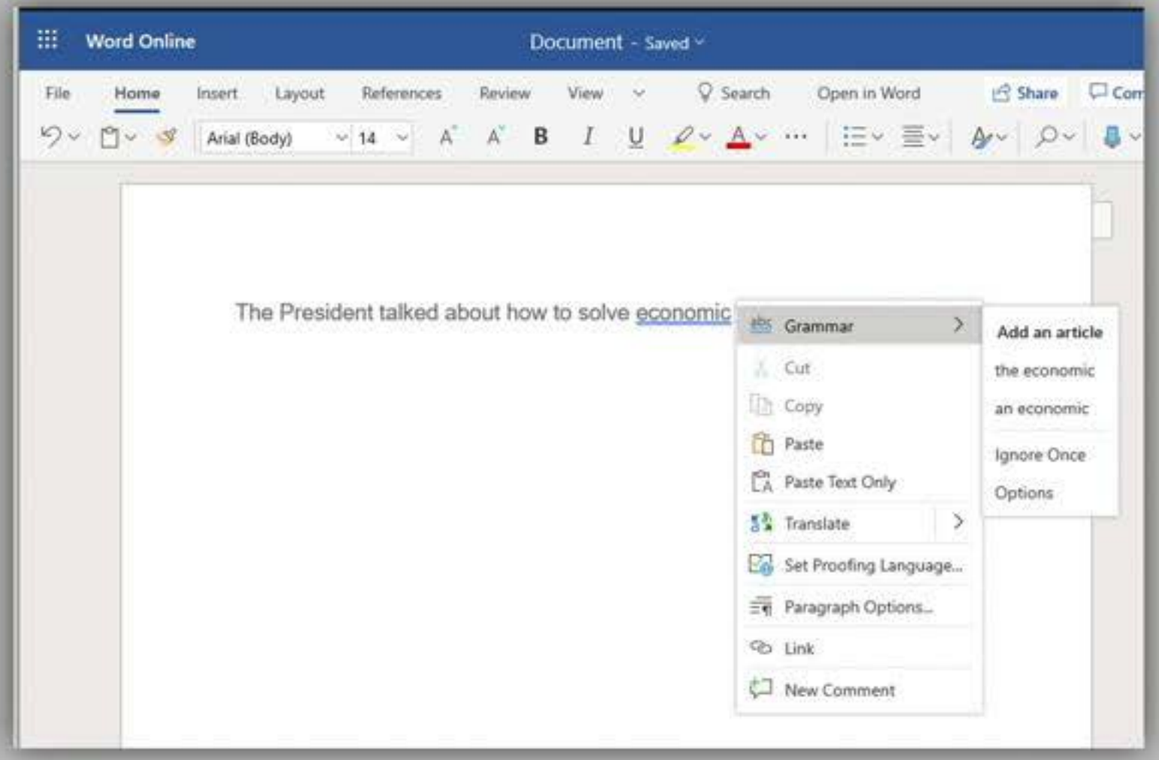
Bing Microsoft Cognitive Services Office 365 ads

Power BI skype

+ more

Office : Missing Determiner

FEATURE OVERVIEW

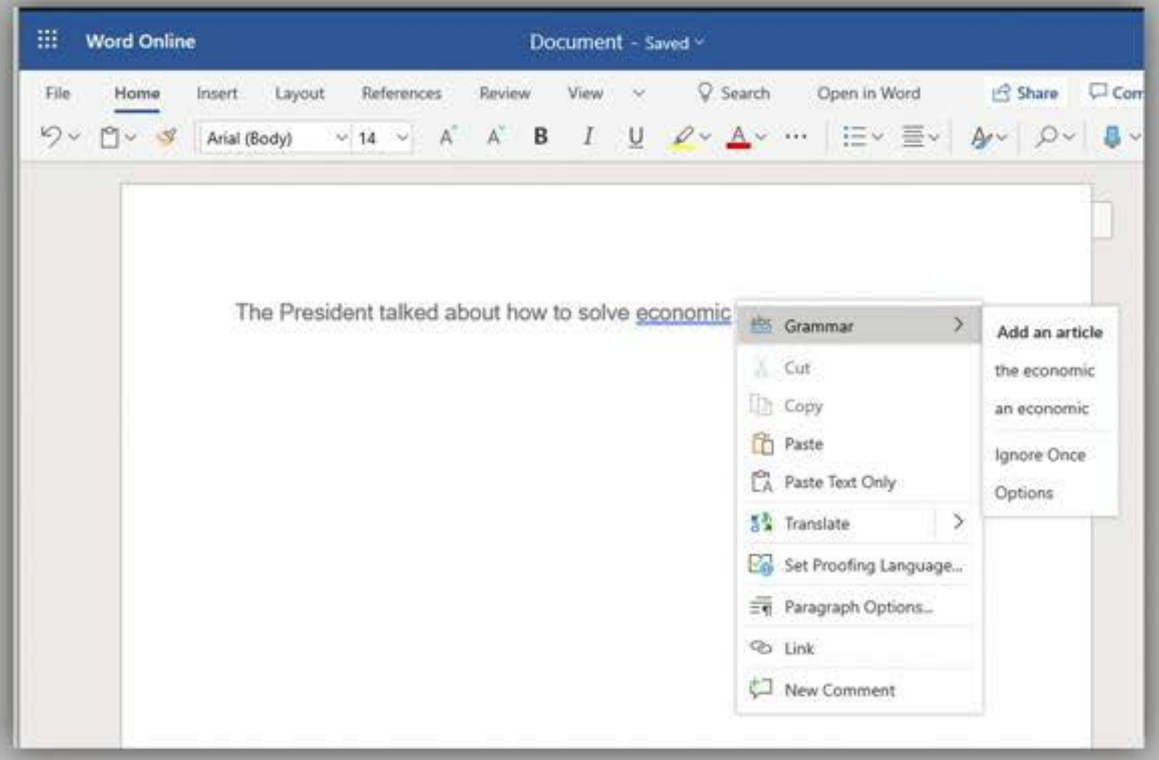


MODEL

Missing Determiner model is used for grammar check and correction for Office Online

Office : Missing Determiner

FEATURE OVERVIEW



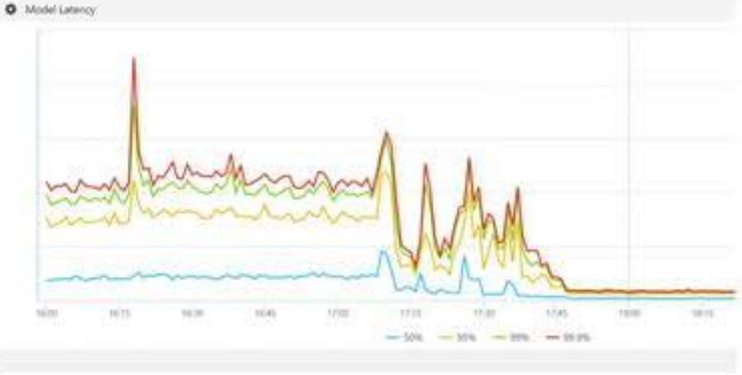
MODEL

Missing Determiner model is used for grammar check and correction for Office Online

PERFORMANCE

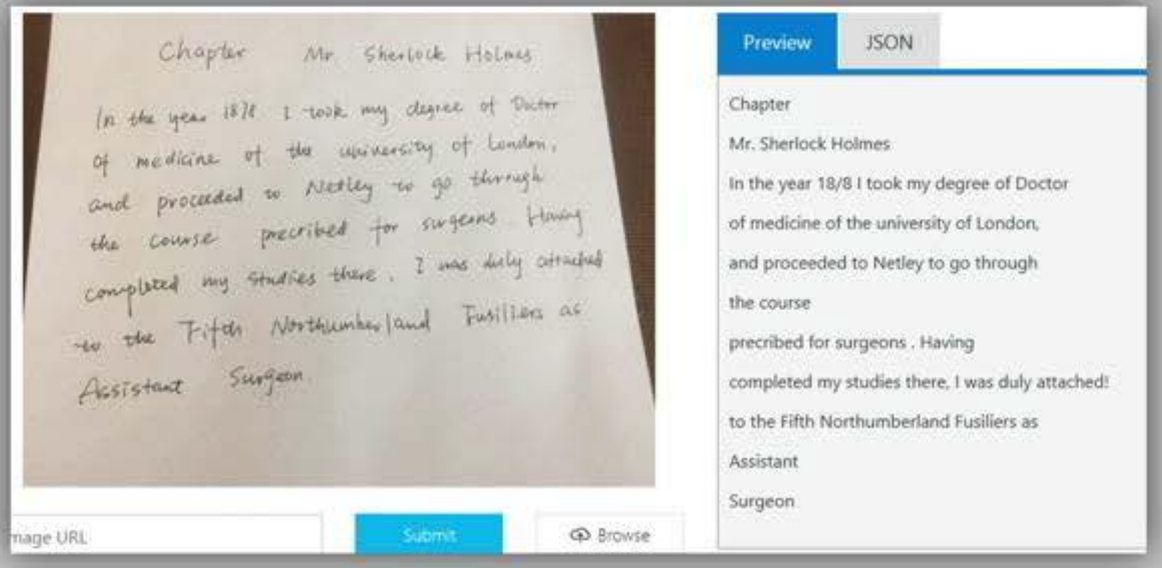
14.6x performance gain with ONNX and ONNX Runtime

ONNX Runtime deployed to production



Cognitive Service : Optical Character Recognition

FEATURE OVERVIEW

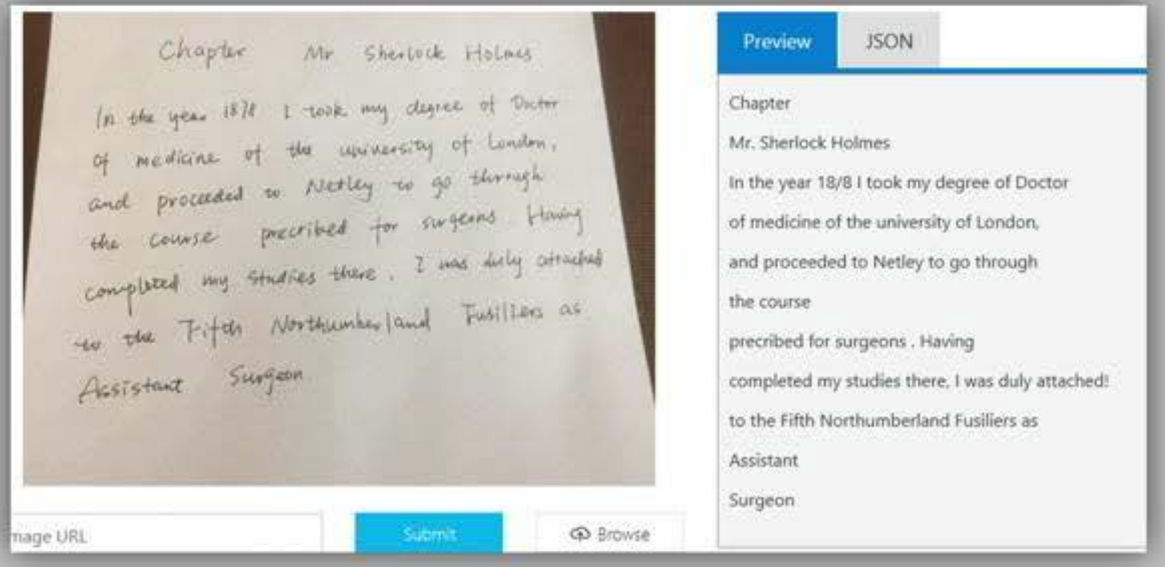


MODEL

OCR model is used to detect text in an image and extract the recognized words into a machine-readable character stream

Cognitive Service : Optical Character Recognition

FEATURE OVERVIEW

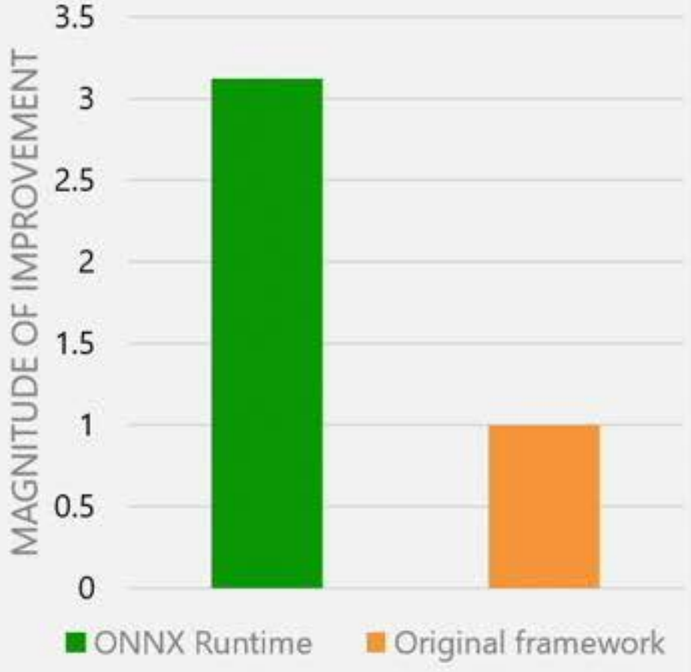


MODEL

OCR model is used to detect text in an image and extract the recognized words into a machine-readable character stream

PERFORMANCE

>3x perf gain by using ONNX and ONNX Runtime



Bing QnA : List and Segment

FEATURE OVERVIEW

QUERY: "empire earth similar games"

MODELS

2 Bing models are used for generating answers from user queries

Bing QnA : List and Segment

FEATURE OVERVIEW

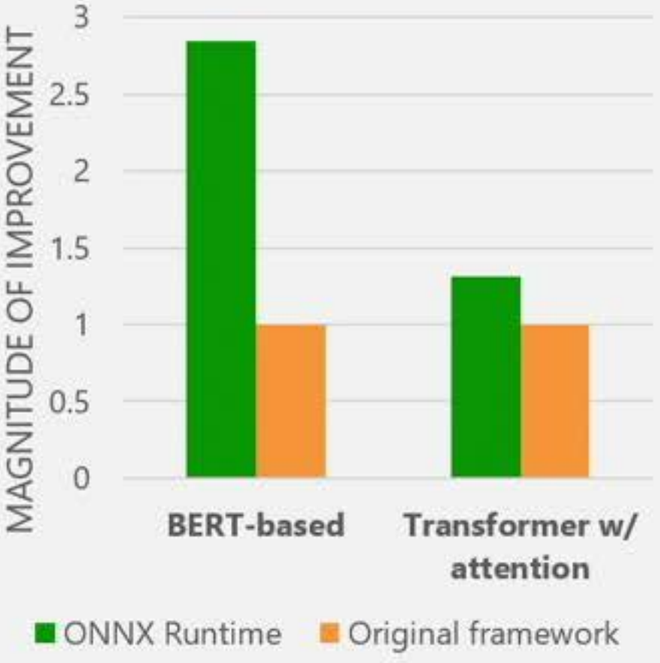
QUERY: "empire earth similar games"

MODELS

2 Bing models are used for generating answers from user queries

PERFORMANCE

Up to **2.8x** perf improvement with ONNX Runtime



Cognitive Service: Computer Vision

FEATURE OVERVIEW

Tags [{ "name": "snow", "confidence": 0.9997839 }, { "name": "sky", "confidence": 0.99880904 }, { "name": "outdoor", "confidence": 0.9985427 }, { "name": "mountain", "confidence": 0.99538064 }, { "name": "nature", "confidence": 0.9336908 }, { "name": "landscape", "confidence": 0.6522721 }, { "name": "cloud", "confidence": 0.6212801 }, { "name": "glacier", "confidence": 0.5570715 }]

MODELS

2 computer vision models are used for enriching images with metadata

Cognitive Service: Computer Vision

FEATURE OVERVIEW

MODELS

2 computer vision models are used for enriching images with metadata

PERFORMANCE

- Latency reduced by 43%
- Throughout increased 1.77x
- API cost reduced by 16%

Technical Design Overviews

ONNX

github.com/onnx/onnx

Technical Design Overview

ONNX – Design Principles

- Support both DNN and traditional ML
- Interoperable
- Backward compatible
- Compact and cross-platform representation for serialization

ONNX – Spec

ONNX is an open specification that consists of the following components:

ONNX – Spec

ONNX is an open specification that consists of the following components:

- A definition of an extensible computation graph model

ONNX – Spec

ONNX is an open specification that consists of the following components:

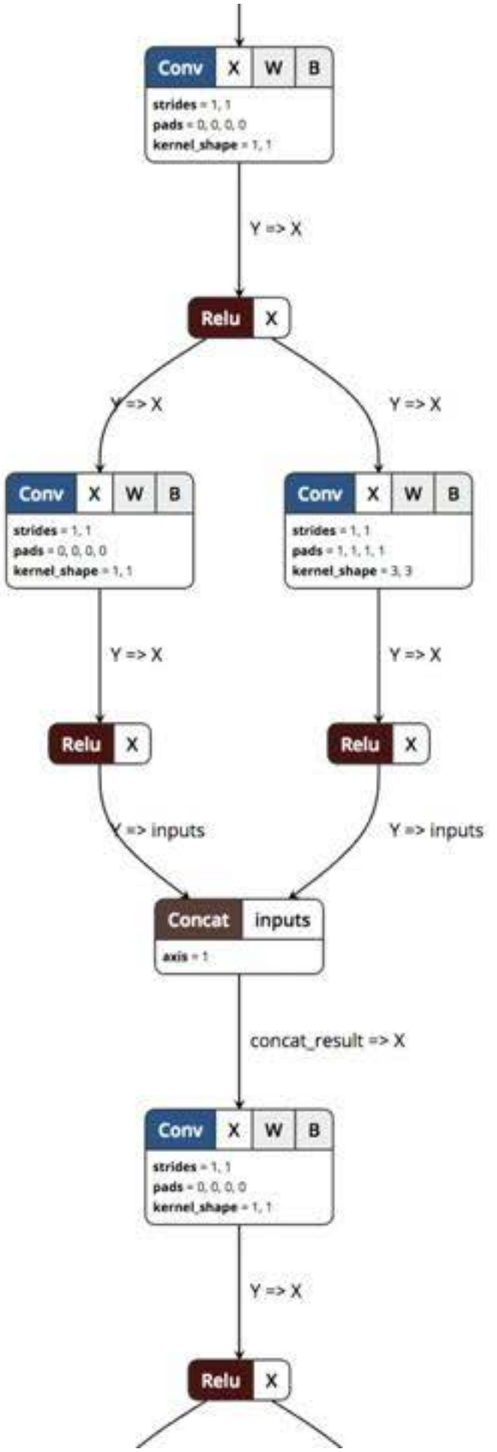
- A definition of an extensible computation graph model
- Definitions of standard data types

ONNX – Spec

ONNX is an open specification that consists of the following components:

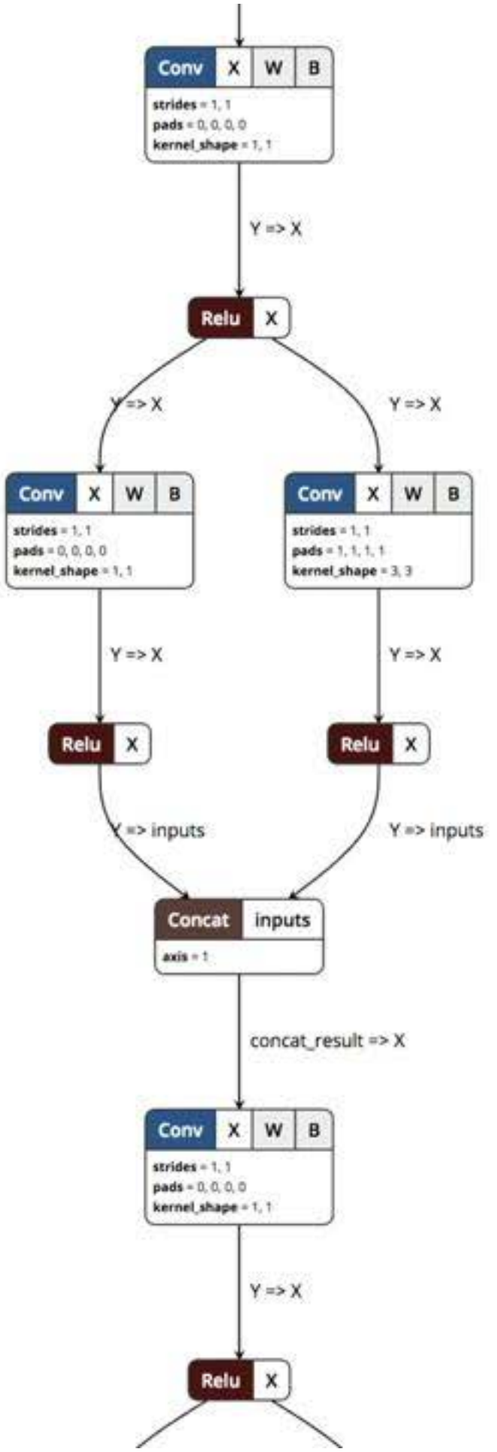
- A definition of an extensible computation graph model
- Definitions of standard data types
- Definitions (schema) of built-in operators (which belong to a versioned operator set)

ONNX – Model File Format



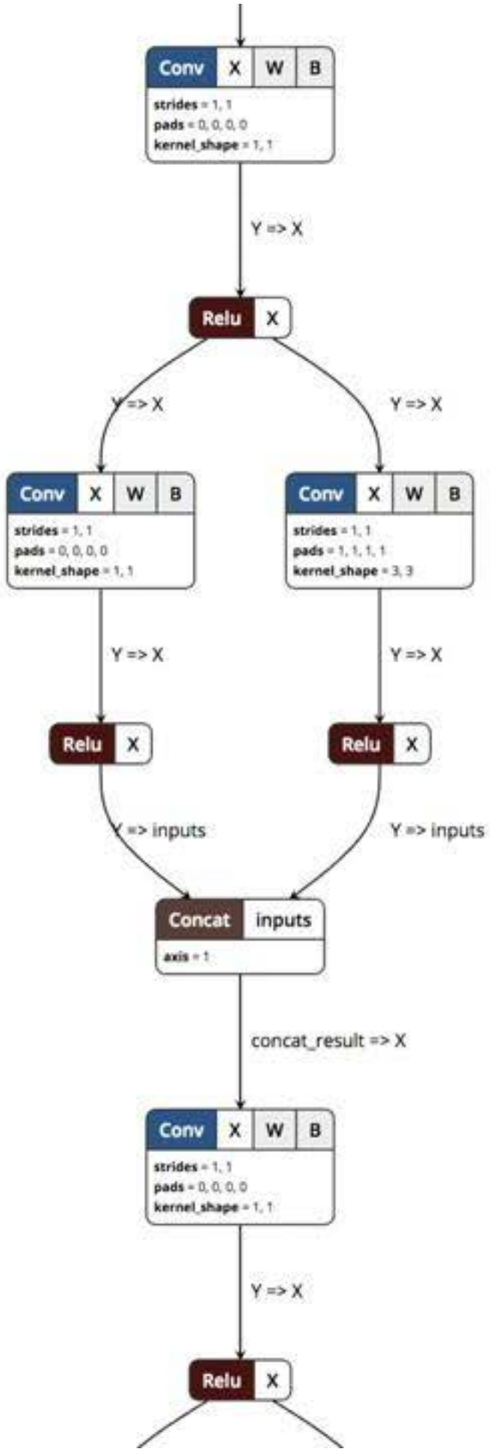
ONNX – Model File Format

- **Model**
  - Version info
  - Metadata
  - Acyclic computation dataflow graph



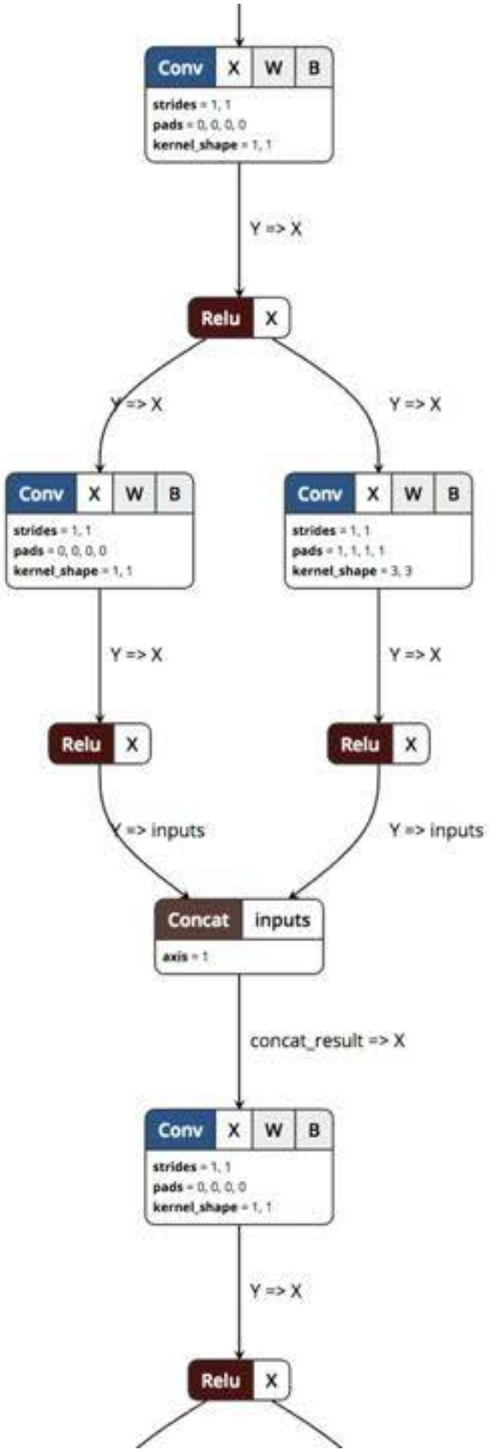
ONNX – Model File Format

- **Model**
  - Version info
  - Metadata
  - Acyclic computation dataflow graph
- **Graph**
  - Inputs and outputs
  - List of computation nodes
  - Graph name



ONNX – Model File Format

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



ONNX – Supported Types

```
message TypeProto {
  message Tensor {
    optional TensorProto.DataType elem_type = 1;
    optional TensorShapeProto shape = 2;
  }
  // repeated T
  message Sequence {
    optional TypeProto elem_type = 1;
  };
  // map<K,V>
  message Map {
    optional TensorProto.DataType key_type = 1;
    optional TypeProto value_type = 2;
  };

  oneof value {
    Tensor tensor_type = 1;
    Sequence sequence_type = 4;
    Map map_type = 5;
  }
}
```

ONNX – Supported Types

- **Tensor type**
  - Element types supported:
  - int8, int16, int32, int64
  - uint8, uint16, uint32, uint64
  - float16, float, double
  - bool
  - string
  - complex64, complex128

```
message TypeProto {
  message Tensor {
    optional TensorProto.DataType elem_type = 1;
    optional TensorShapeProto shape = 2;
  }
  // repeated T
  message Sequence {
    optional TypeProto elem_type = 1;
  };
  // map<K,V>
  message Map {
    optional TensorProto.DataType key_type = 1;
    optional TypeProto value_type = 2;
  };

  oneof value {
    Tensor tensor_type = 1;
    Sequence sequence_type = 4;
    Map map_type = 5;
  }
}
```

ONNX – Supported Types

- **Tensor type**
  - Element types supported:
  - int8, int16, int32, int64
  - uint8, uint16, uint32, uint64
  - float16, float, double
  - bool
  - string
  - complex64, complex128
- **Non-tensor types in ONNX-ML:**
  - Sequence
  - Map

```
message TypeProto {
  message Tensor {
    optional TensorProto.DataType elem_type = 1;
    optional TensorShapeProto shape = 2;
  }
  // repeated T
  message Sequence {
    optional TypeProto elem_type = 1;
  };
  // map<K,V>
  message Map {
    optional TensorProto.DataType key_type = 1;
    optional TypeProto value_type = 2;
  };

  oneof value {
    Tensor tensor_type = 1;
    Sequence sequence_type = 4;
    Map map_type = 5;
  }
}
```

ONNX – Operators

Relu

Relu takes one input data (Tensor) and produces one output data (Tensor) where the rectified linear function, y = max(0, x), is applied to the tensor elementwise.

Version

This version of the operator has been available since version 6 of the default ONNX operator set. Other versions of this operator: Relu-1

Inputs

X : *T*
Input tensor

Outputs

Y : *T*
Output tensor

Type Constraints

T : ***tensor(float16), tensor(float), tensor(double)***
Constrain input and output types to float tensors.

Examples

▼ relu

```
node = onnx.helper.make_node(
    'Relu',
    inputs=['x'],
    outputs=['y'],
)
x = np.random.randn(3, 4, 5).astype(np.float32)
y = np.clip(x, 0, np.inf)

expect(node, inputs=[x], outputs=[y],
       name='test_relu')
```

ONNX – Operators

- An operator is identified by <name, domain, version>

Relu

Relu takes one input data (Tensor) and produces one output data (Tensor) where the rectified linear function, y = max(0, x), is applied to the tensor elementwise.

Version

This version of the operator has been available since version 6 of the default ONNX operator set. Other versions of this operator: Relu-1

Inputs

`X` : *T*
Input tensor

Outputs

`Y` : *T*
Output tensor

Type Constraints

`T` : *tensor(float16), tensor(float), tensor(double)*
Constrain input and output types to float tensors.

Examples

▼ relu

```
node = onnx.helper.make_node(
    'Relu',
    inputs=['x'],
    outputs=['y'],
)
x = np.random.randn(3, 4, 5).astype(np.float32)
y = np.clip(x, 0, np.inf)

expect(node, inputs=[x], outputs=[y],
       name='test_relu')
```

ONNX – Operators

- An operator is identified by <name, domain, version>
- Core ops (ONNX and ONNX-ML)
  - Should be supported by ONNX-compatible products
  - Generally cannot be meaningfully further decomposed
  - Currently 124 ops in ai.onnx domain and 18 in ai.onnx.ml
  - Supports many scenarios/problem areas including image classification, recommendation, natural language processing, etc.

Relu

Relu takes one input data (Tensor) and produces one output data (Tensor) where the rectified linear function, y = max(0, x), is applied to the tensor elementwise.

Version

This version of the operator has been available since version 6 of the default ONNX operator set. Other versions of this operator: Relu-1

Inputs

X : *T*
Input tensor

Outputs

Y : *T*
Output tensor

Type Constraints

T : ***tensor(float16), tensor(float), tensor(double)***
Constrain input and output types to float tensors.

Examples

▼ relu

```
node = onnx.helper.make_node(
    'Relu',
    inputs=['x'],
    outputs=['y'],
)
x = np.random.randn(3, 4, 5).astype(np.float32)
y = np.clip(x, 0, np.inf)

expect(node, inputs=[x], outputs=[y],
       name='test_relu')
```

ONNX – Operators

- An operator is identified by <name, domain, version>
- Core ops (ONNX and ONNX-ML)
  - Should be supported by ONNX-compatible products
  - Generally cannot be meaningfully further decomposed
  - Currently 124 ops in ai.onnx domain and 18 in ai.onnx.ml
  - Supports many scenarios/problem areas including image classification, recommendation, natural language processing, etc.
- Custom ops
  - Ops specific to framework or runtime
  - Indicated by a custom domain name
  - Primarily meant to be a safety-valve

Relu

Relu takes one input data (Tensor) and produces one output data (Tensor) where the rectified linear function, y = max(0, x), is applied to the tensor elementwise.

Version

This version of the operator has been available since version 6 of the default ONNX operator set. Other versions of this operator: Relu-1

Inputs

X : *T*
Input tensor

Outputs

Y : *T*
Output tensor

Type Constraints

T : ***tensor(float16), tensor(float), tensor(double)***
Constrain input and output types to float tensors.

Examples

▼ relu

```
node = onnx.helper.make_node(
    'Relu',
    inputs=['x'],
    outputs=['y'],
)
x = np.random.randn(3, 4, 5).astype(np.float32)
y = np.clip(x, 0, np.inf)

expect(node, inputs=[x], outputs=[y],
       name='test_relu')
```

ONNX – Versioning

Versioning in ONNX is done at 3 levels

ONNX – Operators

- An operator is identified by <name, domain, version>
- Core ops (ONNX and ONNX-ML)
  - Should be supported by ONNX-compatible products
  - Generally cannot be meaningfully further decomposed
  - Currently 124 ops in ai.onnx domain and 18 in ai.onnx.ml
  - Supports many scenarios/problem areas including image classification, recommendation, natural language processing, etc.
- Custom ops
  - Ops specific to framework or runtime
  - Indicated by a custom domain name
  - Primarily meant to be a safety-valve

Relu

Relu takes one input data (Tensor) and produces one output data (Tensor) where the rectified linear function, y = max(0, x), is applied to the tensor elementwise.

Version

This version of the operator has been available since version 6 of the default ONNX operator set. Other versions of this operator: Relu-1

Inputs

X : *T*
Input tensor

Outputs

Y : *T*
Output tensor

Type Constraints

T : ***tensor(float16), tensor(float), tensor(double)***
Constrain input and output types to float tensors.

Examples

▼ relu

```
node = onnx.helper.make_node(
    'Relu',
    inputs=['x'],
    outputs=['y'],
)
x = np.random.randn(3, 4, 5).astype(np.float32)
y = np.clip(x, 0, np.inf)

expect(node, inputs=[x], outputs=[y],
       name='test_relu')
```

ONNX – Versioning

Versioning in ONNX is done at 3 levels

ONNX – Versioning

Versioning in ONNX is done at 3 levels

- IR version (file format): currently at version 5

ONNX – Versioning

Versioning in ONNX is done at 3 levels

- IR version (file format): currently at version 5
- Opset version: ONNX models declare which operator sets they require as a list of two-part operator ids (domain, opset_version)

ONNX – Versioning

Versioning in ONNX is done at 3 levels

- IR version (file format): currently at version 5
- Opset version: ONNX models declare which operator sets they require as a list of two-part operator ids (domain, opset_version)
- Operator version: A given operator is identified by a three-tuple: `(domain, op_type, and op_version)`

ONNX – Operators

- An operator is identified by <name, domain, version>
- Core ops (ONNX and ONNX-ML)
  - Should be supported by ONNX-compatible products
  - Generally cannot be meaningfully further decomposed
  - Currently 124 ops in ai.onnx domain and 18 in ai.onnx.ml
  - Supports many scenarios/problem areas including image classification, recommendation, natural language processing, etc.
- Custom ops
  - Ops specific to framework or runtime
  - Indicated by a custom domain name
  - Primarily meant to be a safety-valve

Relu

Relu takes one input data (Tensor) and produces one output data (Tensor) where the rectified linear function, y = max(0, x), is applied to the tensor elementwise.

Version

This version of the operator has been available since version 6 of the default ONNX operator set. Other versions of this operator: Relu-1

Inputs

X : *T*
Input tensor

Outputs

Y : *T*
Output tensor

Type Constraints

T : ***tensor(float16), tensor(float), tensor(double)***
Constrain input and output types to float tensors.

Examples

▼ relu

```
node = onnx.helper.make_node(
    'Relu',
    inputs=['x'],
    outputs=['y'],
)
x = np.random.randn(3, 4, 5).astype(np.float32)
y = np.clip(x, 0, np.inf)

expect(node, inputs=[x], outputs=[y],
       name='test_relu')
```

ONNX – Versioning

Versioning in ONNX is done at 3 levels

- IR version (file format): currently at version 5
- Opset version: ONNX models declare which operator sets they require as a list of two-part operator ids (domain, opset_version)
- Operator version: A given operator is identified by a three-tuple: `(domain, op_type, and op_version)`

ONNX – Versioning

Versioning in ONNX is done at 3 levels

- IR version (file format): currently at version 5
- Opset version: ONNX models declare which operator sets they require as a list of two-part operator ids (domain, opset_version)
- Operator version: A given operator is identified by a three-tuple: `(domain, op_type, and op_version)`

`Details:` https://github.com/onnx/onnx/blob/master/docs/Versioning.md

ONNX Runtime

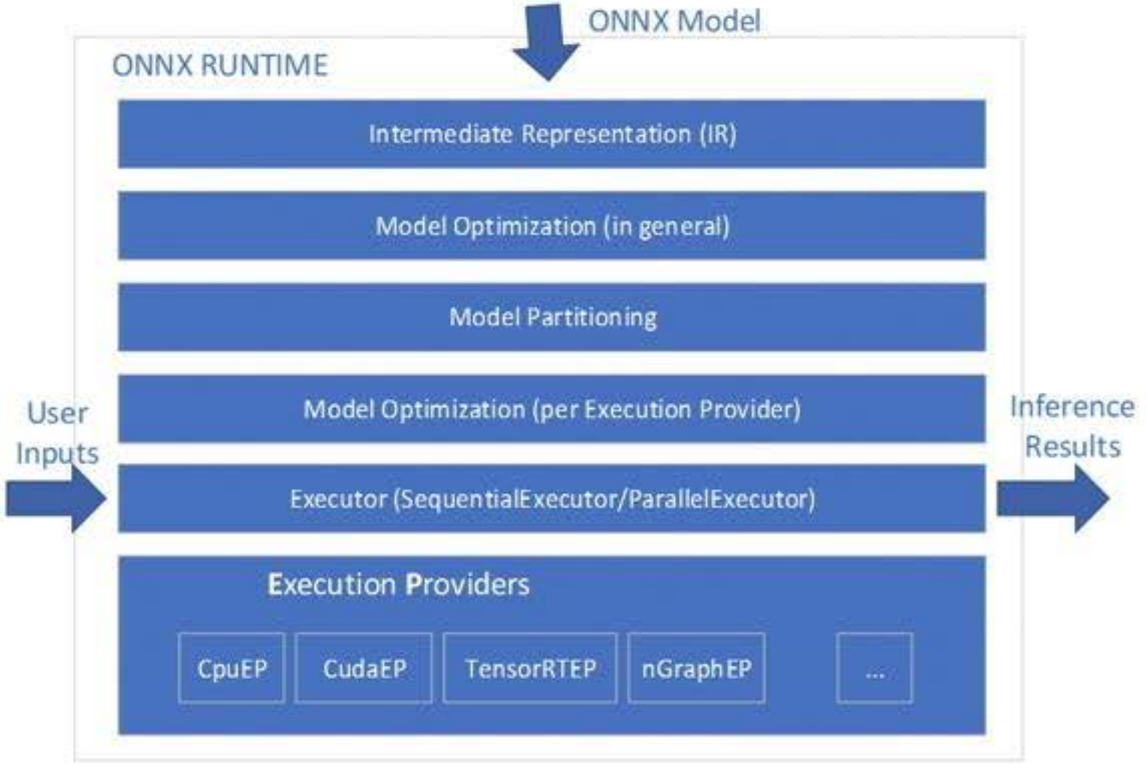
github.com/microsoft/onnxruntime

Technical Design Overview

ONNX Runtime – Design principles

- Provide complete implementation of the ONNX standard – implement all versions of the operators (since opset 7)
- Backward compatibility
- High performance
- Cross platform
- Leverage custom accelerators and runtimes to enable maximum performance (execution providers)
- Support hybrid execution of the models
- Extensible through pluggable modules

ONNX Runtime – Architecture Overview



ONNX Runtime – Architecture Overview

Graph Optimization

- Node elimination (dropout, identity, etc.)
- Node fusion, constant folding, etc.

ONNX Runtime – Architecture Overview

Graph Optimization

- Node elimination (dropout, identity, etc.)
- Node fusion, constant folding, etc.

Model Partitioning

- Graph partitioning based on execution providers' capability
- Greedy algo based on user preferences (Current)
- ML based partitioner? (Next)

ONNX Runtime – Architecture Overview

Graph Optimization

- Node elimination (dropout, identity, etc.)
- Node fusion, constant folding, etc.

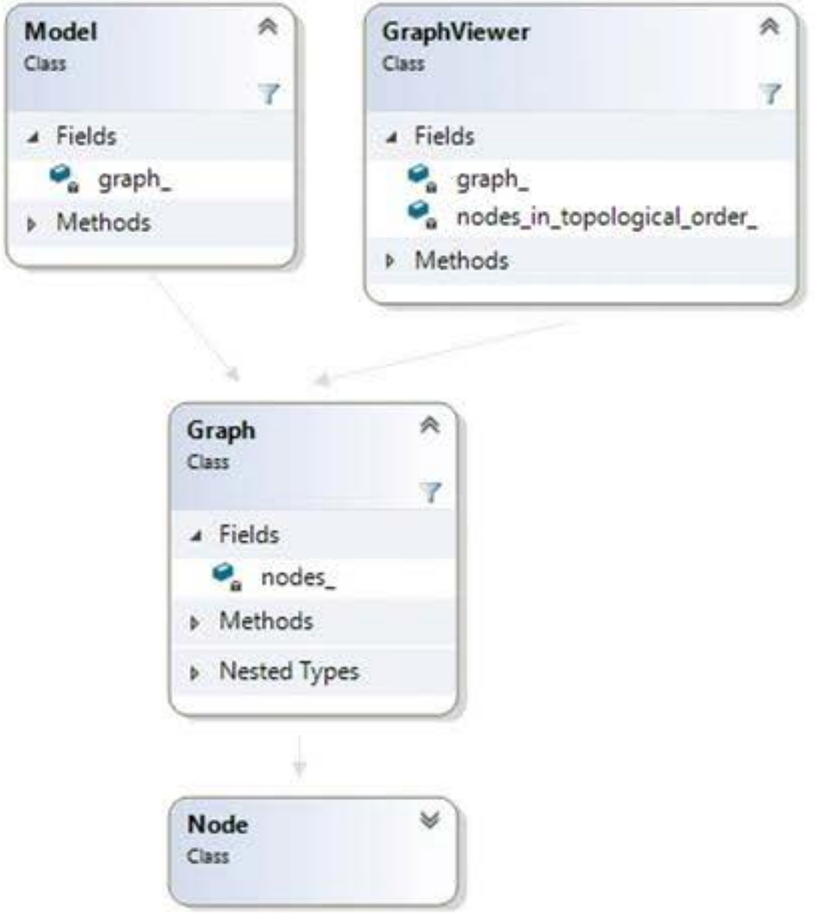
Model Partitioning

- Graph partitioning based on execution providers' capability
- Greedy algo based on user preferences (Current)
- ML based partitioner? (Next)

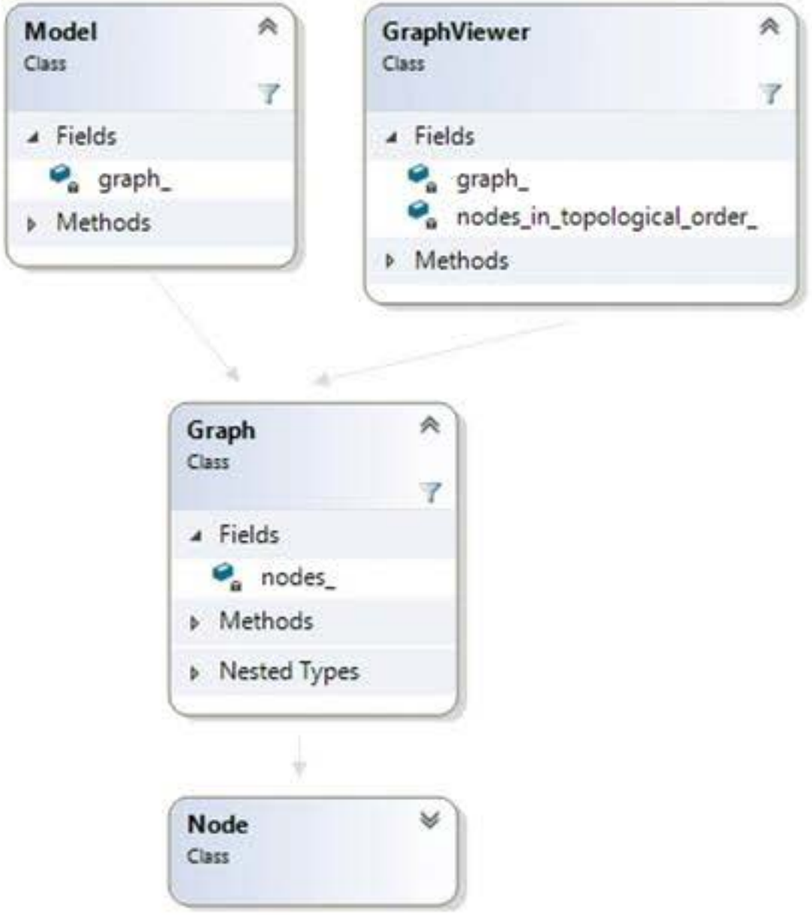
Execution Provider

- Plug-in hardware accelerator
- Key APIs
  - GetCapability – given a graph, return a collection of sub-graphs it can run
  - Compile – given a sub-graph (node), return function pointers to run the sub-graph

ONNX Runtime – IR



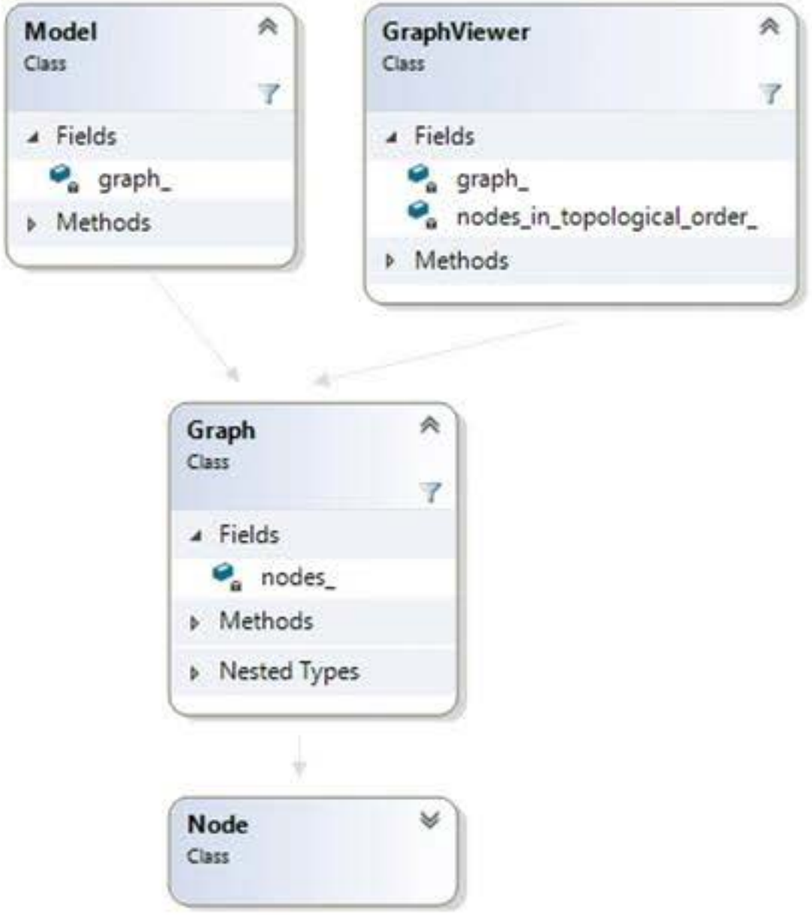
ONNX Runtime – IR



Model/Graph/Node

In-memory object mapping to ONNX model file format design.
Offering APIs to read/write a computational graph.

ONNX Runtime – IR



Model/Graph/Node

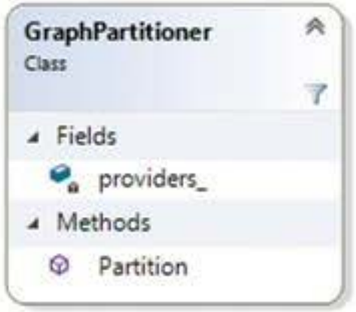
In-memory object mapping to ONNX model file format design. Offering APIs to read/write a computational graph.

GraphViewer

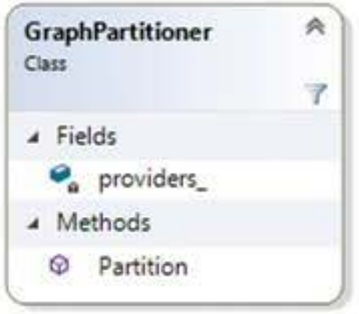
Read-only view of a computational graph. Used in:

1. IExecutionProvider (API between Runtime and hardware accelerator)
2. Model evaluation (after model optimization and partitioning)

ONNXRUNTIME – Graph Partitioning



ONNXRUNTIME – Graph Partitioning



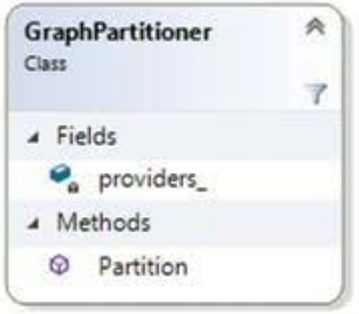
GraphPartitioner

Given a mutable graph, graph partitioner assigns graph nodes to each execution provider per their capability and idea goal is to reach best performance in a heterogeneous environment.

ONNX RUNTIME uses a “greedy” node assignment mechanism

- Users specify a preferred execution provider list in order
- ONNX RUNTIME will go thru the list in order to check each provider’s capability and assign nodes to it if it can run the nodes.

ONNXRUNTIME – Graph Partitioning



GraphPartitioner

Given a mutable graph, graph partitioner assigns graph nodes to each execution provider per their capability and idea goal is to reach best performance in a heterogeneous environment.

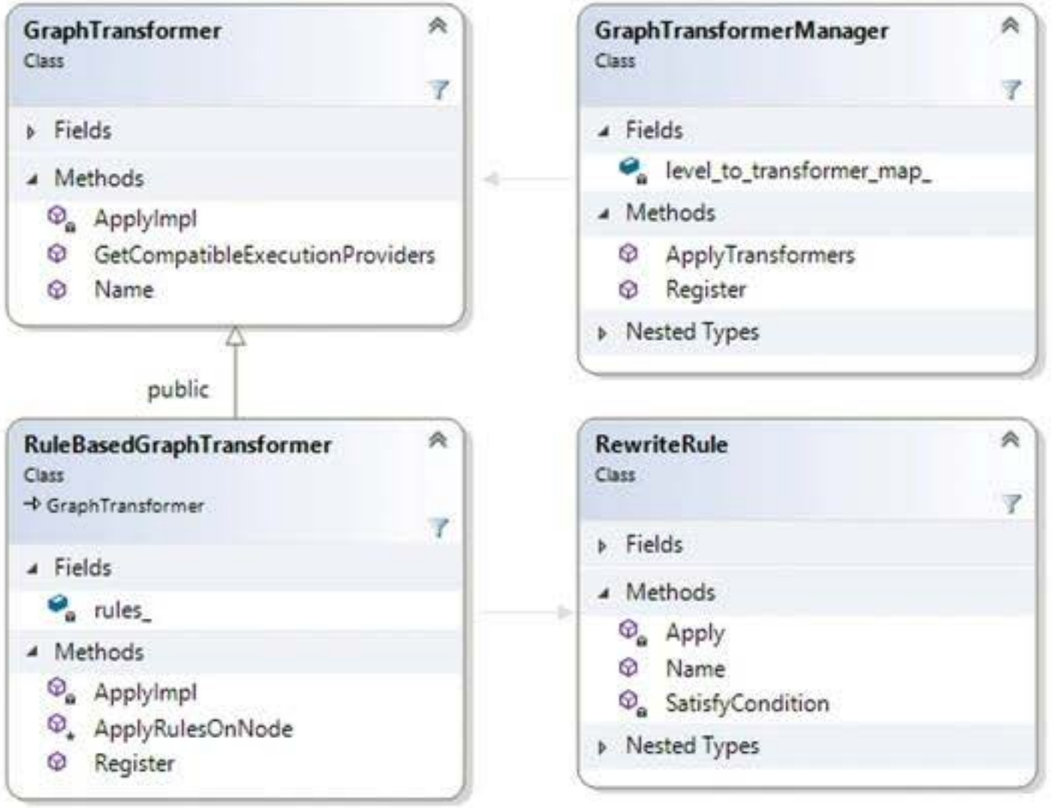
ONNX RUNTIME uses a “greedy” node assignment mechanism

- Users specify a preferred execution provider list in order
- ONNX RUNTIME will go thru the list in order to check each provider’s capability and assign nodes to it if it can run the nodes.

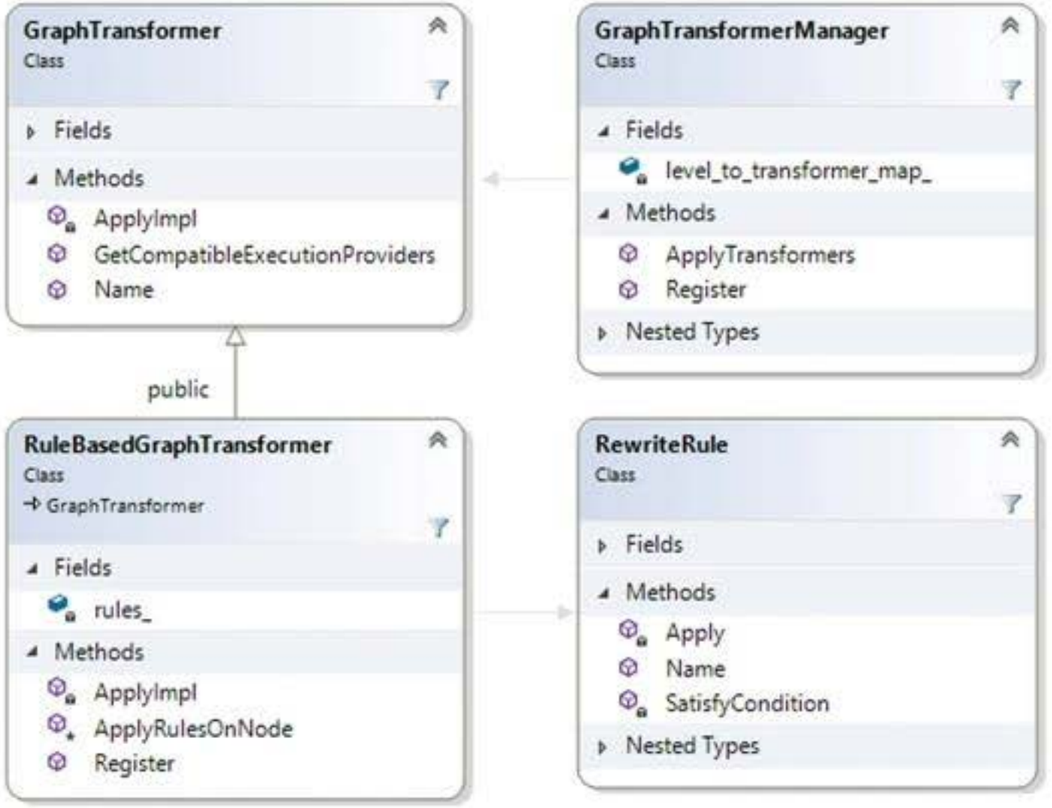
FUTURE:

- Manual tuned partitioning
- ML based partitioning

ONNX Runtime – Graph Optimization



ONNX Runtime – Graph Optimization



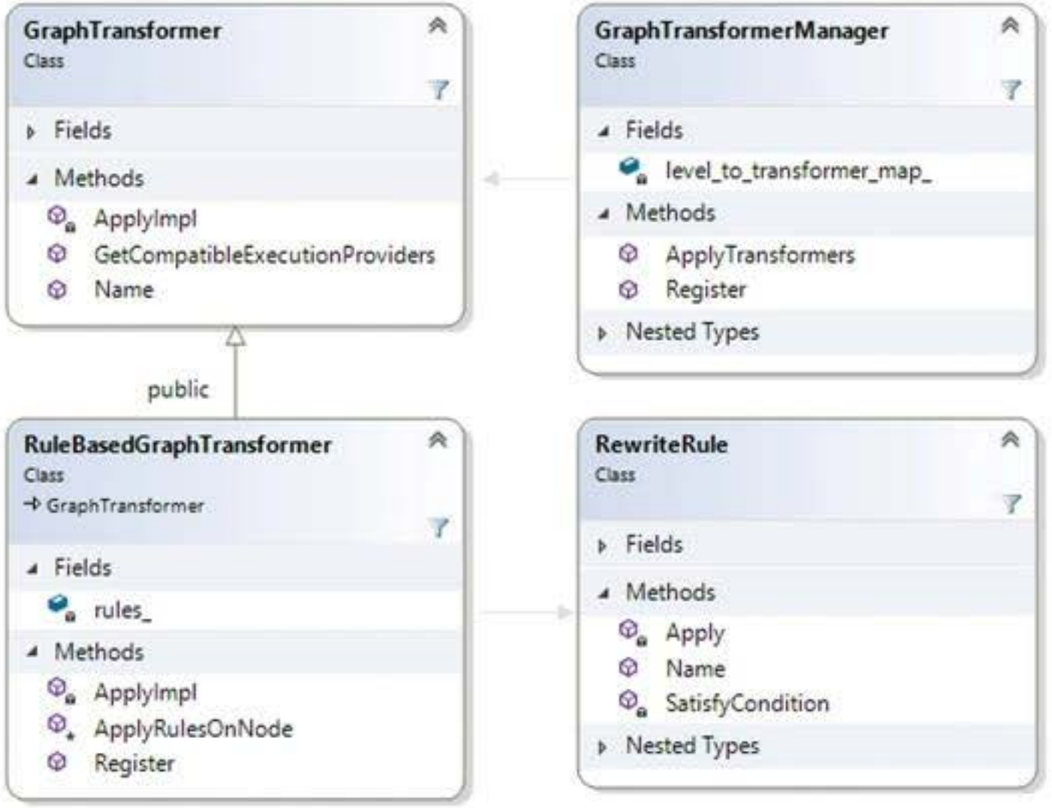
RewriteRule

An interface created for finding patterns (with specific nodes) and applying rewriting rules against a sub-graph.

GraphTransformer

An interface created for applying graph transformation with full graph editing capability.

ONNX Runtime – Graph Optimization



RewriteRule

An interface created for finding patterns (with specific nodes) and applying rewriting rules against a sub-graph.

GraphTransformer

An interface created for applying graph transformation with full graph editing capability.

TransformerLevel

Level 0: Transformers anyway will be applied after graph partitioning (e.g. cast insertion, mem copy insertion)

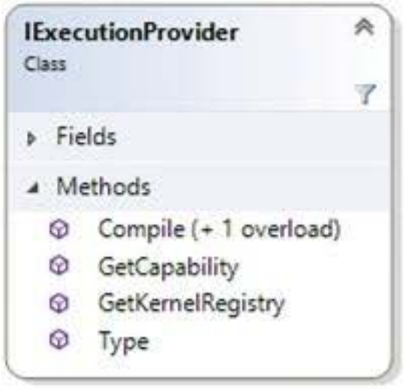
Level 1: General transformers not specific to any specific execution provider (e.g. drop out elimination)

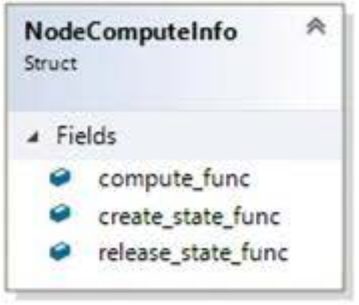
Level 2: Execution provider specific transformers (e.g. transpose insertion for FPGA)

Graph Optimizations

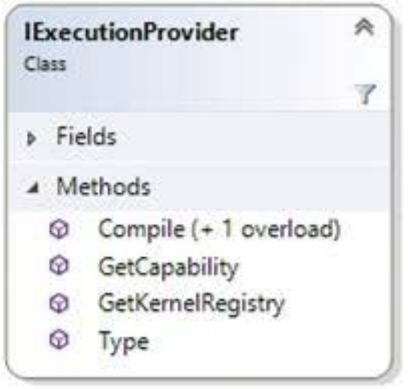
- Level 0
  - Cast
  - MemCopy
- Level 1
  - EliminateIdentity
  - EliminateSlice
  - UnsqueezeElimination
  - EliminateDropout
  - FuseReluClip
  - ShapeToInitializer
  - ConvAddFusion
  - ConvMulFusion
  - ConvBNFusion
- Level 2

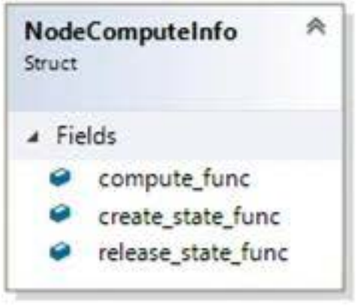
ONNX Runtime – Execution Provider





ONNX Runtime – Execution Provider





IExecutionProvider

A hardware accelerator interface to query its capability and get corresponding executables.

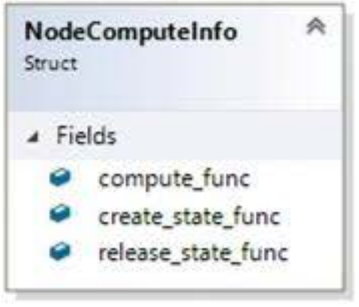
1) Kernel based execution providers

   These execution providers provides implementations of operators defined in ONNX (e.g. CPUExecutionProvider, CudaExecutionProvider, MKLDNNExecutionProvider, etc.)

2) Runtime based execution providers

   These execution providers may not have implementations with the granularity of ONNX ops, but it can run whole or partial ONNX graph. Say, it can run several ONNX ops (a sub-graph) together with one function it has (e.g. TensorRTExecutionProvider, nGraphExecutionProvider, etc.)

ONNX Runtime – Execution Provider



IExecutionProvider

A hardware accelerator interface to query its capability and get corresponding executables.

1) Kernel based execution providers

   These execution providers provides implementations of operators defined in ONNX (e.g. CPUExecutionProvider, CudaExecutionProvider, MKLDNNExecutionProvider, etc.)

2) Runtime based execution providers

   These execution providers may not have implementations with the granularity of ONNX ops, but it can run whole or partial ONNX graph. Say, it can run several ONNX ops (a sub-graph) together with one function it has (e.g. TensorRTExecutionProvider, nGraphExecutionProvider, etc.)

NodeComputeInfo

A data structure carries executables returned by runtime-based execution providers.

ONNX Runtime – Execution Provider

Improve performance by leveraging hardware accelerators

CPU: Xeon-E5-1650v4
RAM: 32G
OS: Ubuntu 16.04

Model	CPU (ms)	nGraph (ms)
bvlc_googlenet	14.29	8.99 (42% ↓)
inception_v2	21.57	16.29 (24% ↓)
resnet50	29.48	21.33 (28% ↓)

Extending ONNX Runtime

Extending ONNX Runtime

- Execution providers
  - Implement the IExecutionProvider interface
  - Examples: TensorRT, OpenVino, NGraph, Android NNAPI, etc

Extending ONNX Runtime

- Execution providers
  - Implement the IExecutionProvider interface
  - Examples: TensorRT, OpenVino, NGraph, Android NNAPI, etc
- Custom operators
  - Support operators outside the ONNX standard
  - Support for writing custom ops in both C/C++ and Python

Extending ONNX Runtime

- Execution providers
  - Implement the IExecutionProvider interface
  - Examples: TensorRT, OpenVino, NGraph, Android NNAPI, etc
- Custom operators
  - Support operators outside the ONNX standard
  - Support for writing custom ops in both C/C++ and Python
- Graph optimizers
  - Implement the GraphTransformer interface

ONNX Runtime – Multi-language API (Python)

Python

```
import onnxruntime as rt
import numpy
from onnxruntime.datasets import get_example
sess = rt.InferenceSession("sigmoid.onnx")
input_name = sess.get_inputs()[0].name
import numpy.random
x = numpy.random.random((3,4,5))
x = x.astype(numpy.float32)
res = sess.run([output_name], {input_name: x})
```

ONNX Runtime – Multi-language API (C)

```
const OrtApi* g_ort = OrtGetApiBase()->GetApi(ORT_API_VERSION);

OrtEnv* env;
g_ort->CreateEnv(ORT_LOGGING_LEVEL_WARNING, "test", &env);

OrtSession* session;
g_ort->CreateSession(env, model_path, session_options, &session);

g_ort->Run(session, NULL, input_names, (const OrtValue* const*)&input_tensor, 1, output_names,
1, &output_tensor);
```

Example reference:
https://github.com/microsoft/onnxruntime/blob/master/csharp/test/Microsoft.ML.OnnxRuntime.EndToEndTests.Capi/C_Api_Sample.cpp

ONNX Runtime – Multi-language API (C#)

C#

```
SessionOptions options = new SessionOptions();
var session = new InferenceSession(“model.onnx”, options))
var inputMeta = session.InputMetadata;
var container = new List<NamedOnnxValue>();
// Prepare input data – container.
// Run the inference
var results = session.Run(container);
```

Example reference:
https://github.com/microsoft/onnxruntime/blob/master/csharp/sample/Microsoft.ML.OnnxRuntime.InferenceSample/Program.cs

ONNX Runtime – Custom Operators

Custom Operators allow for registration of operators not officially supported by ONNX

ONNX Runtime – Custom Operators

Custom Operators allow for registration of operators not officially supported by ONNX

OrtCustomOp

```
void*(ORT_API_CALL* CreateKernel)(_In_ struct OrtCustomOp* op, _In_ const
OrtCustomOpApi* api, _In_ const OrtKernelInfo* info);

void(ORT_API_CALL* KernelCompute)(_In_ void* op_kernel, _In_
OrtKernelContext* context);

void(ORT_API_CALL* KernelDestroy)(_In_ void* op_kernel);
```

ONNX Runtime 1.0

- C APIs are ABI compatible and follows Semantic Versioning. Programs linked with the current version of ORT library will continue to work with subsequent releases without the need to update any client code or re-link.
- ONNX 1.6 compatibility - operator support for all opset11 ops, including Sequence ops.
- Support for CentOS 6
- Preview Execution Providers: NUPHAR, DirectML, ARM Compute Library
- Availability of ONNX Go Live tool, which automates the process of shipping ONNX models by combining model conversion, correctness tests, and performance tuning into a single pipeline as a series of Docker images.

ONNX Go Live (OLive)

- Automates the process of ONNX model shipping
- Integrates
  - model conversion
  - correctness test
  - performance tuning

into a single pipeline and outputs a production ready ONNX model with ONNX Runtime configurations (execution provider + optimization options)

https://github.com/microsoft/OLive

ONNX Runtime – Upcoming

- Additional execution providers
  - NN API
  - NXP (Qualcom)
- Java API
- Model online training support
- Continued performance optimizations

References

- ONNX: https://github.com/onnx/onnx
- ONNX Runtime: https://github.com/microsoft/onnxruntime
- ONNX Runtime Tutorials: https://github.com/microsoft/onnxruntime#examples-and-tutorials
- ONNX Tutorials: https://github.com/onnx/tutorials
- ONNX Converters: https://github.com/onnx/onnxmltools/tree/master/onnxmltools
- ONNX Ecosystem Docker Image: https://github.com/onnx/onnx-docker/tree/master/onnx-ecosystem
- Perf Tuning: https://github.com/microsoft/onnxruntime/blob/master/docs/ONNX_Runtime_Perf_Tuning.md
- ONNX Runtime in AzureML: https://aka.ms/onnxnotebooks

Q&A