Microsoft

Reactor

一起学人工智能系列
- 模型部署与应用

2021-11-19

Map


Stockholm
COMING SOON
London
Redmond
Toronto
New York
San Francisco
Tel Aviv
Shanghai
Abu Dhabi
Bangalore
Sao Paulo
Sydney

个人介绍

Kinfey Lo – (卢建晖)

Microsoft Cloud Advocate

前微软MVP、Xamarin MVP和微软RD，拥有超过10年的云原生、人工智能和移动应用经验，为教育、金融和医疗提供应用解决方案。 Microsoft Iginte，Teched 会议讲师，Microsoft AI 黑客马拉松教练，目前在微软，为技术人员和不同行业宣讲技术和相关应用场景。

爱编程(Python， C#，TypeScript，Swift，Rust，Go ）

专注于人工智能，云原生，跨平台移动开发

Github : https://github.com/kinfey

Email : kinfeylo@microsoft.com **Blog :** https://blog.csdn.net/kinfey

Twitter : @Ljh8304

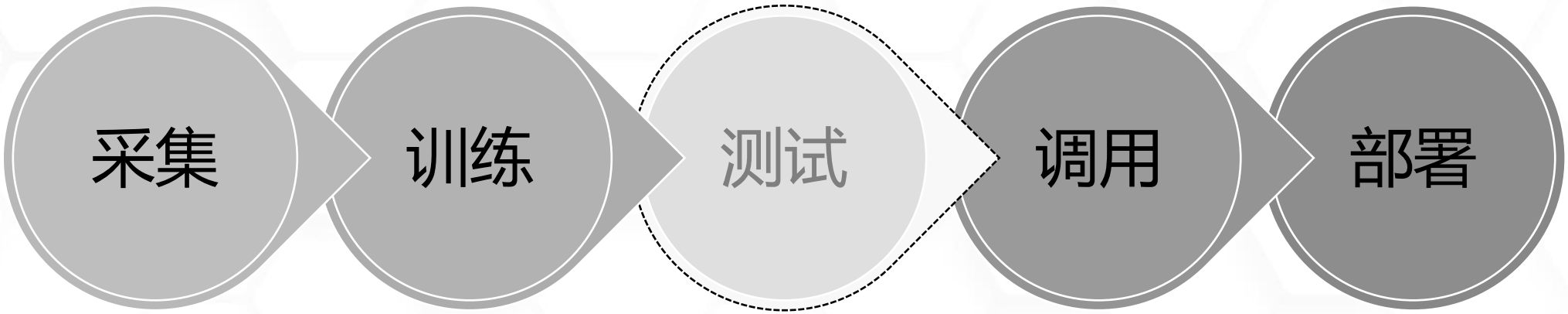
模型介绍

机器学习无处不在

Microsoft 365

如何在桌面端或手机端应用中使用模型

模型文件中不仅包含了所有的权重矩阵，还记录了该神经网络的数据流图。这样不同平台、不同语言都可以根据模型文件里的信息构建网络、加载权重矩阵、执行前向计算。



为什么需要模型文件

如今人工智能发展的越来越快，在图像分析、自然语言处理、语音识别等领域都有着令人惊喜的效果。而模型，可以想象为一个“黑盒”，输入是你需要处理的一张图像，输出是一个它的类别信息或是一些特征，模型文件也因此保存了能完成这一过程的所有重要信息，并且还能用来再次训练、推理等，方便了模型的传播与发展。

模型文件描述的是什么

首先我们需要了解，目前绝大部分的深度学习框架都将整个AI模型的计算过程抽象成数据流图（Data Flow Graphs），用户写的模型构建代码都由框架组建出一个数据流图（也可以简单理解为神经网络的结构），而当程序开始运行时，框架的执行器会根据调度策略依次执行数据流图，完成整个计算。

当有了这个背景知识后，我们很容易想到，为了方便地重用AI模型的计算过程，我们需要将它运行的数据流图、相应的运行参数（Parameters）和训练出来的权重（Weights）保存下来，这就是AI模型文件主要描述的内容。

AI模型的作用是什么?

以视觉处理为例，人通过眼睛捕获光线，传递给大脑处理，返回图像的一些信息，比如，这是花，是动物。AI模型的作用就相当于大脑的处理，能根据输入的数据给予一定的判断。使用封装好的AI模型，那么设计者只需要考虑把输入的数据处理成合适的格式（类似于感光细胞的作用），然后传递给AI模型（大脑），之后就可以得到一个想要的输出。

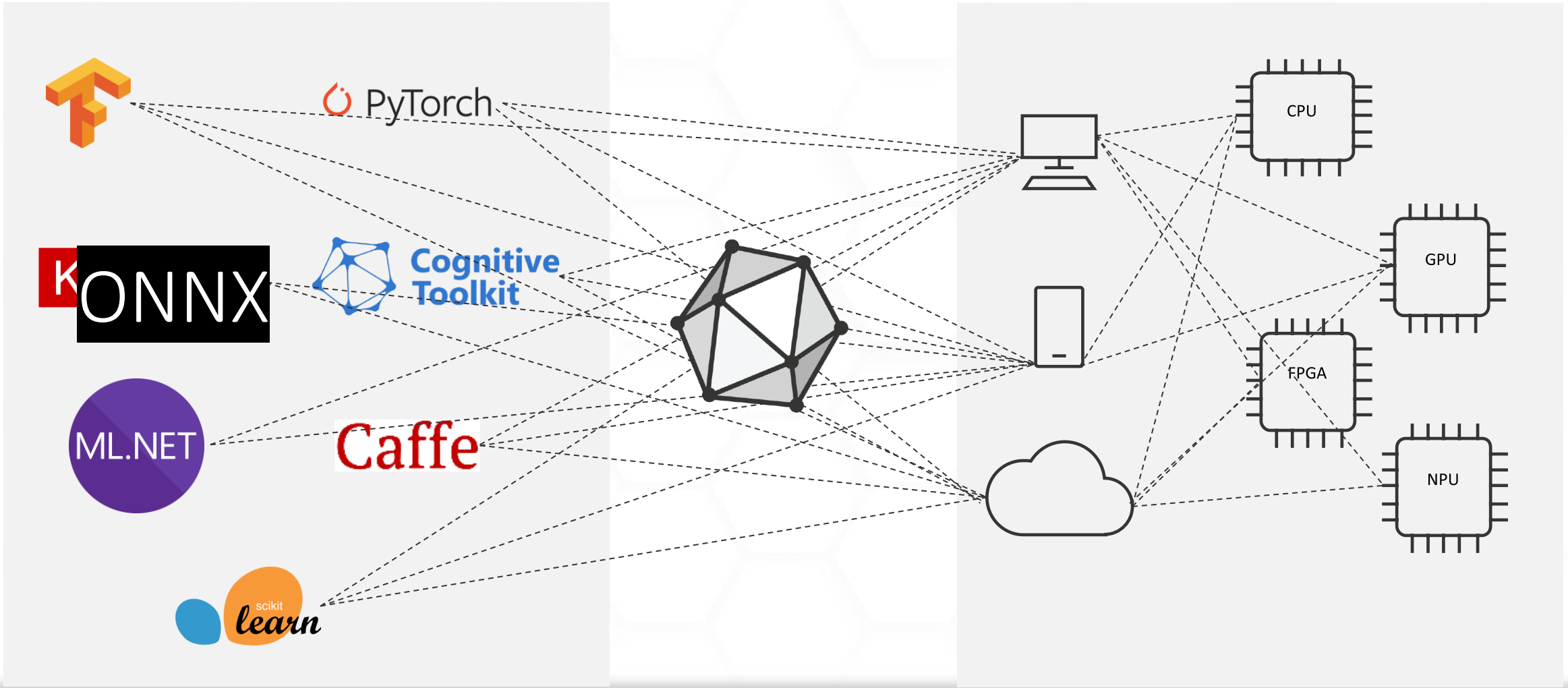
模型文件有哪些类型，TensorFlow和其他框架的区别

由于每个深度学习框架都有自己的设计理念和工具链，对数据流图的定义和粒度都不一样，所以每家的AI模型文件都有些区别，几乎不能通用。例如，TensorFlow的Checkpoint Files用Protobuf去保存数据流图，用SSTable去保存权重；Keras用Json表述数据流图而用h5py去保存权重；PyTorch由于是主要聚焦于动态图计算，模型文件甚至只用pickle保存了权重而没有完整的数据流图。

TensorFlow在设计之初，就考虑了从训练、预测、部署等复杂的需求，所以它的数据流图几乎涵盖了整个过程可能涉及到操作，例如初始化、后向求导及优化算法、设备部署（Device Placement）和分布式化、量化压缩等，所以只需要通过TensorFlow的模型文件就能够获取模型完整的运行逻辑，所以很容易迁移到各种平台使用。

ONNX

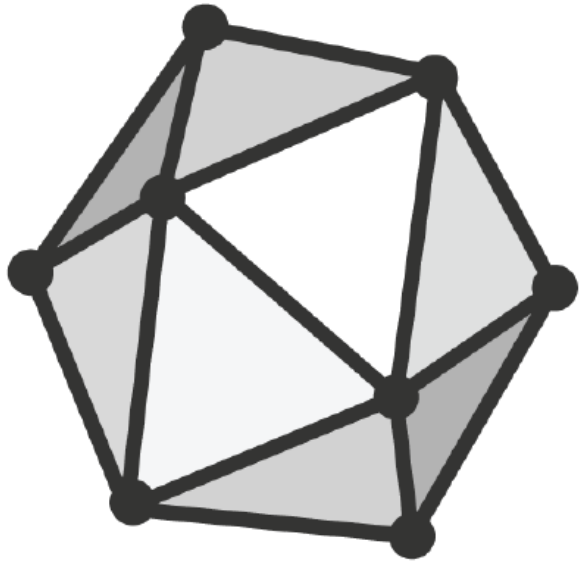
现实场景


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

ONNX

OPEN NEURAL NETWORK EXCHANGE

ONNX是一个开放式的规范，定义了可扩展的计算图模型、标准数据类型以及内置的运算符。该文件在存储结构上可以理解为是一种层级的结构



如何使用ONNX模型

- ONNX Model Zoo
- Azure Custom Vision 和 AutoML 自动生成
- 通过转换

- Azure 机器学习服务

相关转换支持

- **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

原生支持

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



LightGBM
LibSVM
dmlc
XGBoost
Spark
PyTorch

Cognitive
Toolkit

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
```

```
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")
```

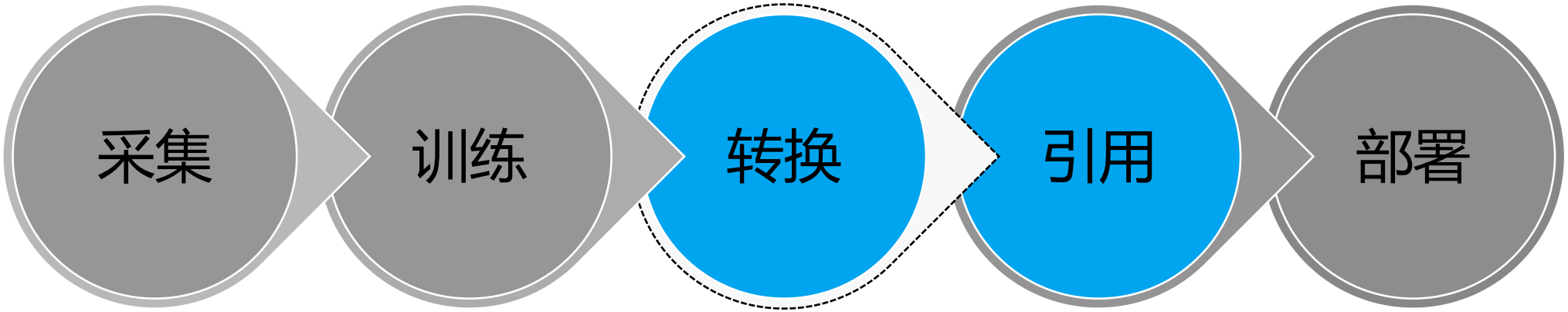
PyTorch

```
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")
```

ML 模型：研究到生产



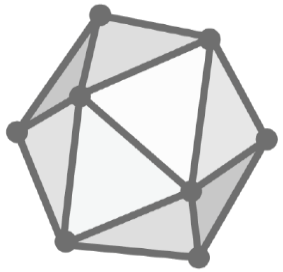
数据科学家

机器学习工程师

ONNX Runtime

- https://microsoft.github.io/onnxruntime/
- ONNX 模型的高性能推理引擎
- 在 MIT 许可下开源
- 完整的 ONNX 规范支持(v1.7+)
  - 涵盖 ONNX 和 ONNX-ML 域模型规范和运算符
  - 向后和向前兼容

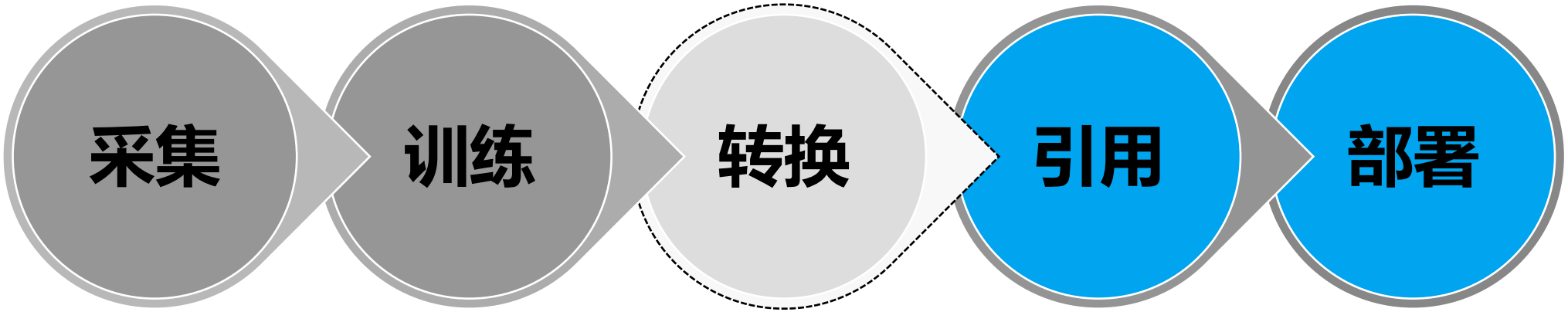
  可扩展的模块化框架
  - 图优化器、运算符和硬件加速器的API
- Windows WinML



支持的架构/语言/提供者

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 模型：研究到生产



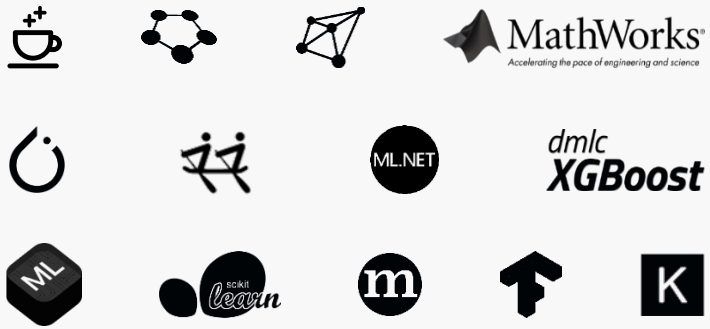
数据科学家

机器学习工程师

部署

CREATE

Frameworks



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

PLATFORMS

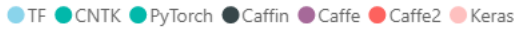
AzureML WinML ML.Net

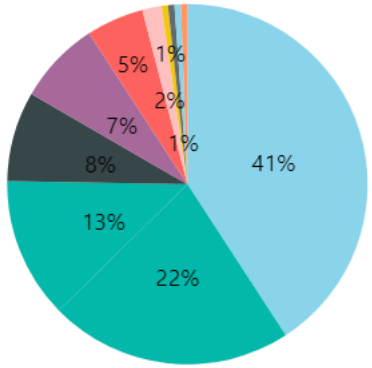
PRODUCTS

Bing Microsoft Cognitive Services Office 365 ads Power BI skype

+ more

Original Model Framework

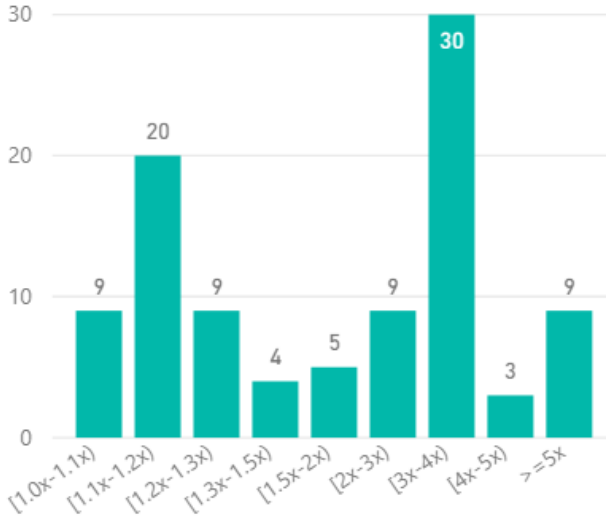




70+ models in production

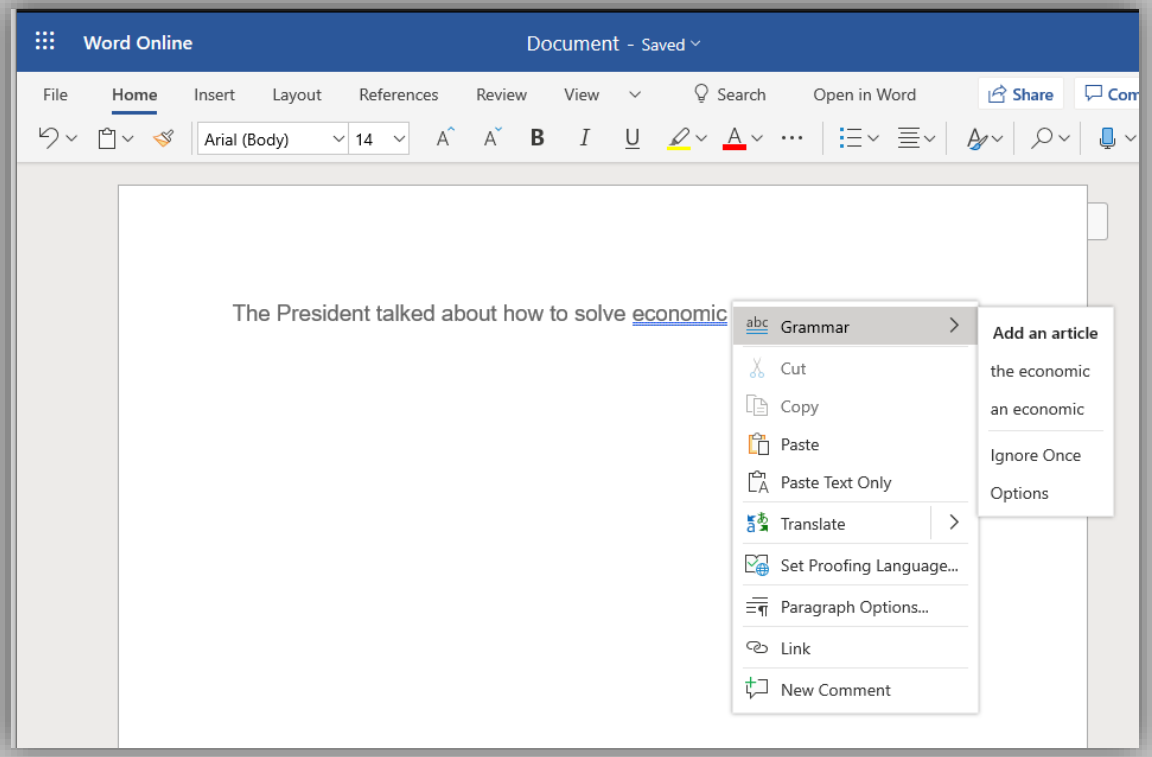
on average ~3x perf improvement

Latency Improvement Summary



Office : Missing Determiner

FEATURE OVERVIEW



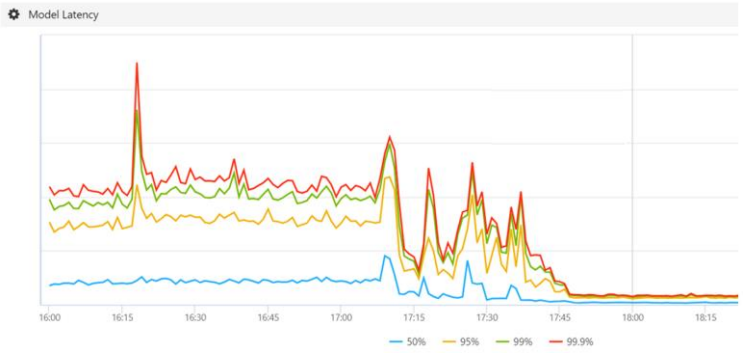
MODEL

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

PERFORMANCE

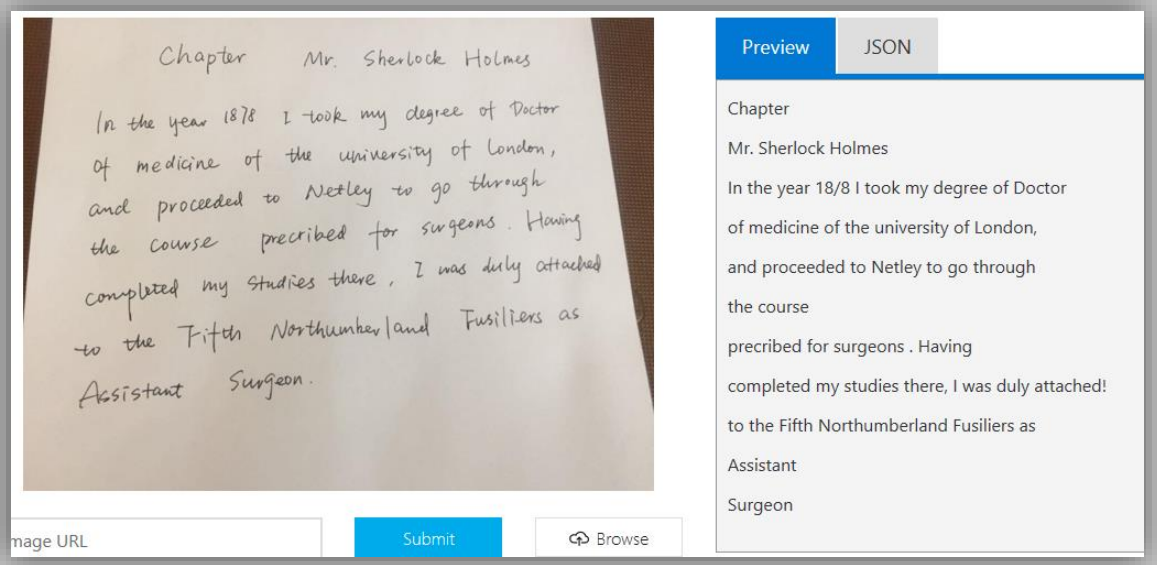
<u>14.6x</u> performance gain with ONNX and ONNX Runtime

ONNX Runtime deployed to production



Cognitive Service : Optical Character Recognition

FEATURE OVERVIEW

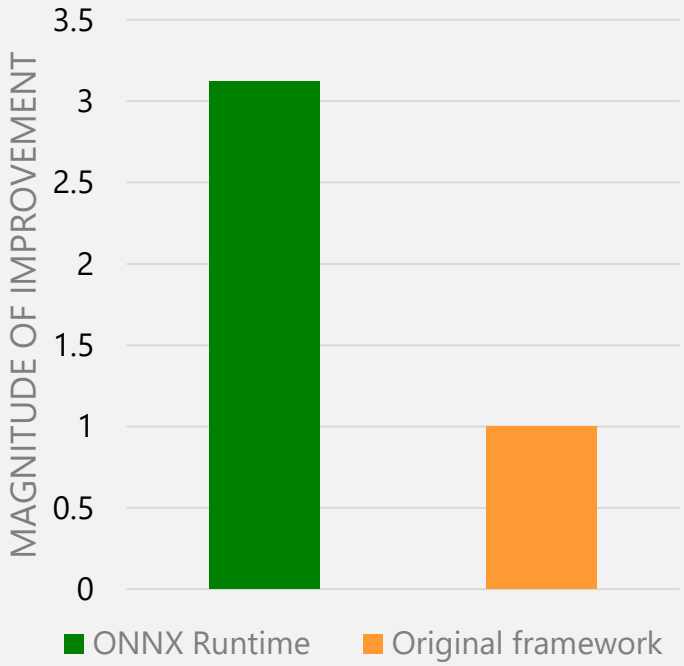


MODEL

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

PERFORMANCE

>3x perf gain by using ONNX and ONNX Runtime



Bing QnA : List and Segment

FEATURE OVERVIEW

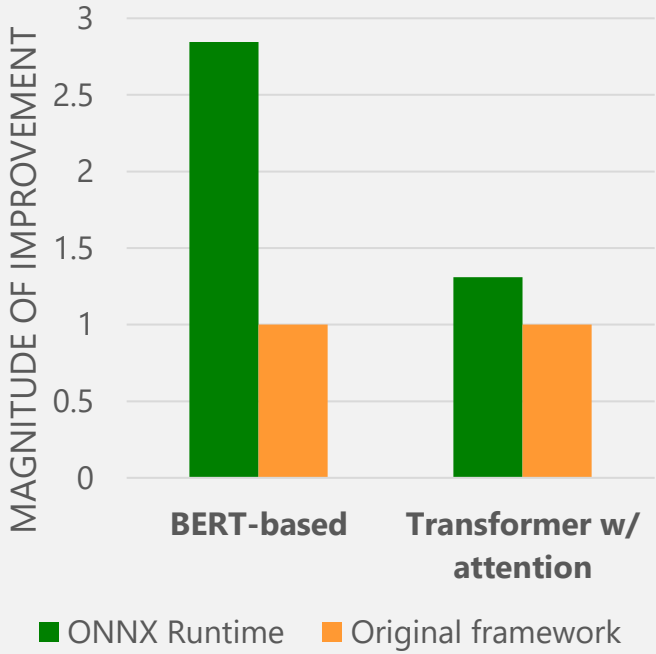
QUERY: "empire earth similar games"

MODELS

2 Bing models are used for generating answers from user queries

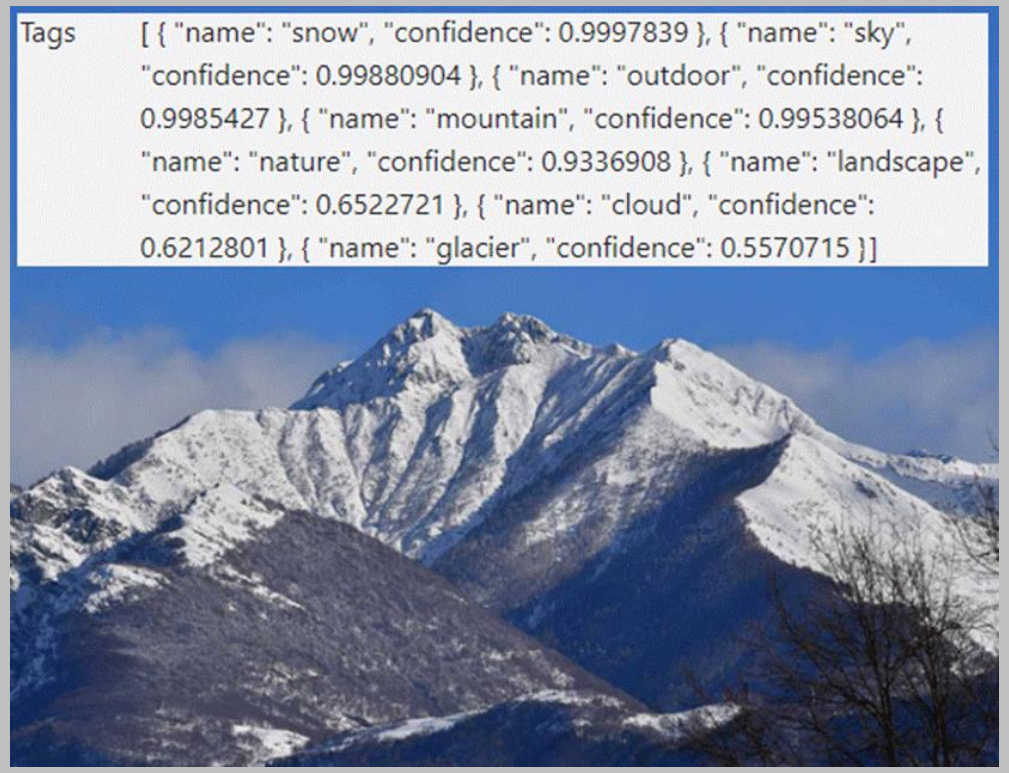
PERFORMANCE

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



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%

技术设计概述

ONNX

github.com/onnx/onnx

Technical Design Overview

ONNX——设计原则

- 同时支持 DNN 和传统机器学习
- 可互操作
- 向下兼容
- 用于序列化的紧凑和跨平台表示

ONNX – Spec

ONNX 是一个开放规范，由以下组件组成：

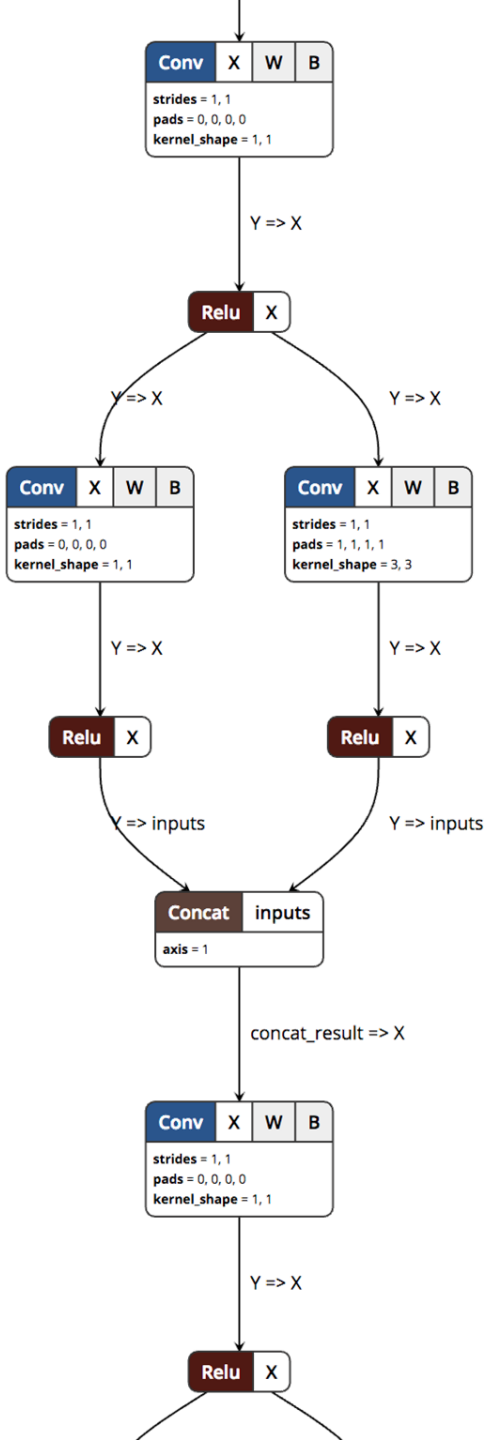
可扩展计算图模型的定义

标准数据类型的定义

内置运算符（属于版本化运算符集）的定义（架构）

ONNX – 模型文件格式

- **Model**
  - 版本信息
  - 元数据
  - 非循环计算数据流图
- **图形**
  - 输入和输出
  - 计算节点列表
  - 图名
- **计算节点**
  - 零个或多个定义类型的输入
  - 一个或多个定义类型的输出
  - 运营商操作参数



ONNX – 支持类型

- **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 – 操作

- 操作员由 <name, domain, version> 标识
- 核心操作（ONNX 和 ONNX-ML）
- 应受 ONNX 兼容产品的支持
- 一般不能有意义地进一步分解
- 目前 ai.onnx 域中有 124 个操作，ai.onnx.ml 中有 18 个操作支持多种场景/问题领域，包括图像分类、推荐、自然语言处理等。
- 自定义操作
- 特定于框架或运行时的操作
- 由自定义域名表示
- 主要是作为一个安全阀

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 – 版本

ONNX 中的版本控制在 3 个级别上完成

IR 版本（文件格式）：目前版本 7

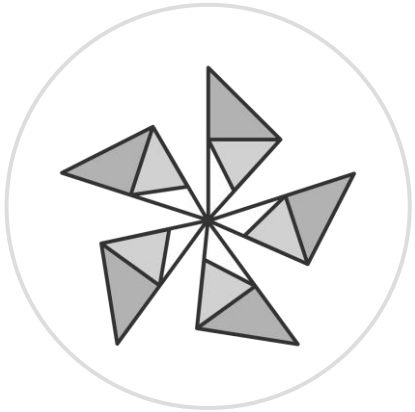
Opset 版本：ONNX 模型将它们需要的运算符集声明为两部

分运算符 ID（域、opset_version）的列表

运算符版本：给定运算符由三元组标识：（域、op_type 和 op_version）

详细

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



ONNX Runtime

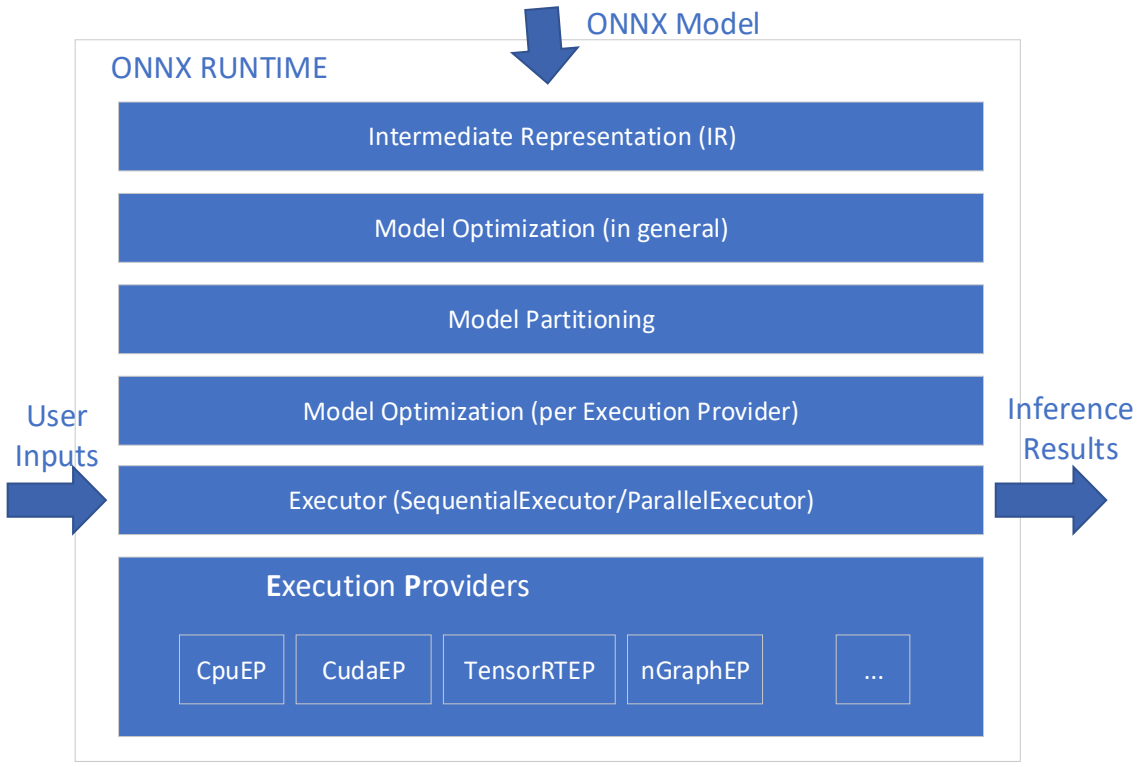
github.com/microsoft/onnxruntime

Technical Design Overview

ONNX Runtime – 设计原则

- 提供 ONNX 标准的完整实现——实现所有版本的运算符（since opset 7）
- 向后兼容
- 高性能
- 跨平台
- 利用自定义加速器和运行时实现最高性能（执行提供程序）支持模型的混合执行
- 可通过可插拔模块进行扩展

ONNX Runtime – 架构



图优化

节点消除（dropout、identity等）
节点融合、不断折叠等。

模型分区

基于执行提供者能力的图分区
基于用户偏好的贪心算法（当前）
基于机器学习的分区器？ （下一个）

执行提供者

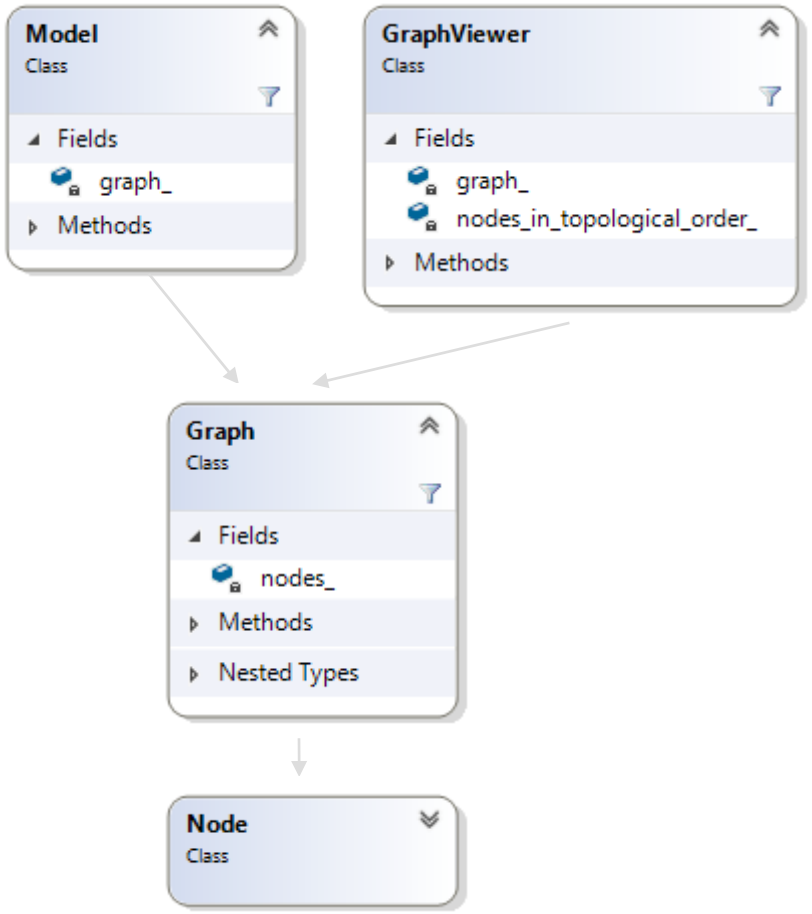
插件硬件加速器

KeyAPI

GetCapability – 给定一个图，返回它可以运行的子图的集合

编译——给定一个子图（节点），返回运行子图的函数指针

ONNX Runtime – IR



Model/Graph/Node

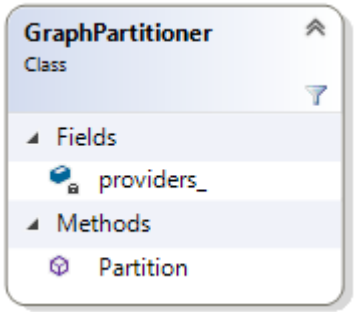
内存对象映射到 ONNX 模型文件格式设计。 提供 API 来读/写计算图。

GraphViewer

计算图的只读视图。 用于：

IExecutionProvider（运行时和硬件加速器之间的 API）
模型评估（在模型优化和分区之后）

ONNXRUNTIME – 图分区



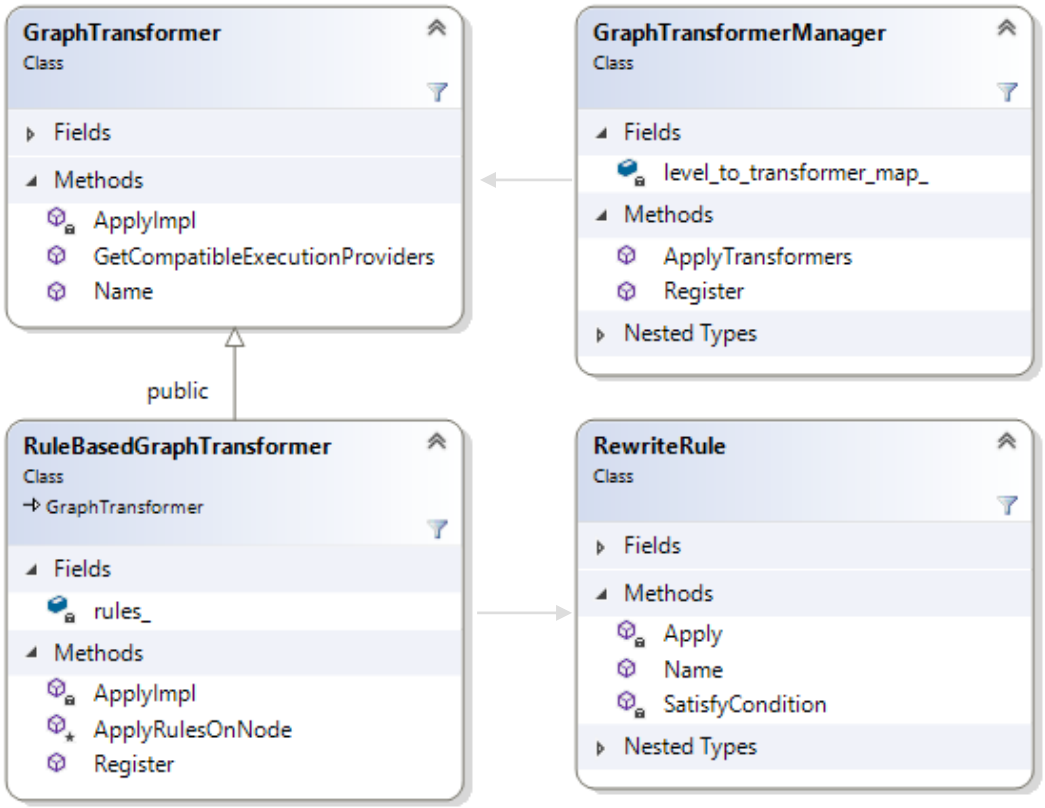
GraphPartitioner

给定一个可变图，图分区器根据每个执行提供者的能力将图节点分配给每个执行提供者，其想法目标是在异构环境中达到最佳性能。

ONNX RUNTIME 使用“贪婪”节点分配机制

- 用户按顺序指定首选执行提供程序列表
- ONNX RUNTIME 将遍历该列表，以检查每个提供程序的能力并为其分配节点（如果它可以运行节点）。

ONNX Runtime – 图优化



RewriteRule

为查找模式（具有特定节点）并针对子图应用重写规则而创建的界面。

GraphTransformer

为应用具有完整图形编辑功能的图形转换而创建的界面。

TransformerLevel

级别 0：无论如何都会在图形分区后应用 Transformer（例如，强制转换插入、内存复制插入）

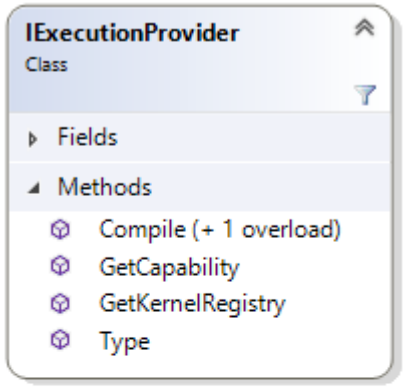
级别 1：不特定于任何特定执行提供程序的通用转换器（例如退出消除）

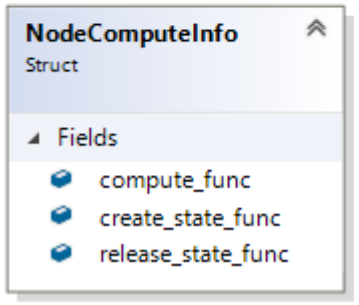
级别 2：执行提供程序特定的转换器（例如 FPGA 的转置插入）

图优化

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

ONNX Runtime – 执行提供者





IExecutionProvider

一个硬件加速器接口，用于查询其能力并获得相应的可执行文件。

基于内核的执行提供程序

这些执行提供程序提供 ONNX 中定义的运算符的实现（例如 CPUExecutionProvider、CudaExecutionProvider、MKLDNNExecutionProvider 等）

基于运行时的执行提供程序

这些执行提供程序可能没有具有 ONNX 操作粒度的实现，但它可以运行整个或部分 ONNX 图。 比如说，它可以运行多个 ONNX 操作（一个子图）以及它拥有的一个函数（例如 TensorRTExecutionProvider、nGraphExecutionProvider 等）

NodeComputeInfo

数据结构携带由基于运行时的执行提供程序返回的可执行文件。

ONNX Runtime – 执行提供者

通过利用硬件加速器提高性能

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% ↓)

GPU: GTX1080
CUDA10, TRT5.0

Model	CUDA (ms)	TensorRT (ms)
resnet50	5.60	3.16 (44% ↓)
Inception_v1	4.66	1.49 (67% ↓)
bvlc_googlenet	4.78	1.64 (66% ↓)

扩展 ONNX 运行时

- **执行提供者**

  实现 IExecutionProvider 接口
  示例：TensorRT、OpenVino、NGraph、Android NNAPI 等

- **自定义运算符**

  支持 ONNX 标准之外的运营商
  支持在 C/C++ 和 Python 中编写自定义操作

- **图优化器**

  实现 GraphTransformer 接口

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 –自定义操作符

自定义运算符允许注册 ONNX 未正式支持的运算符

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 API 与 ABI 兼容并遵循语义版本控制。 与当前版本的 ORT 库链接的程序将继续与后续版本一起使用，而无需更新任何客户端代码或重新链接。
- ONNX 1.6 兼容性 - 操作员支持所有 opset11 操作，包括序列操作。
- 支持 CentOS 6
- 预览执行提供程序：NUPHAR、DirectML、ARM Compute Library
- ONNX Go Live 工具的可用性，该工具通过将模型转换、正确性测试和性能调优作为一系列 Docker 映像组合到单个管道中来自动化交付 ONNX 模型的过程。

ONNX Go Live (OLive)

自动化 ONNX 模型运输过程

集成

模型转换

正确性测试

性能调优

进入单个管道并输出具有 ONNX 运行时配置（执行提供程序 + 优化选项）的生产就绪 ONNX 模型

https://github.com/microsoft/OLive

QnA

Microsoft

Reactor

Thank You!