Microsoft

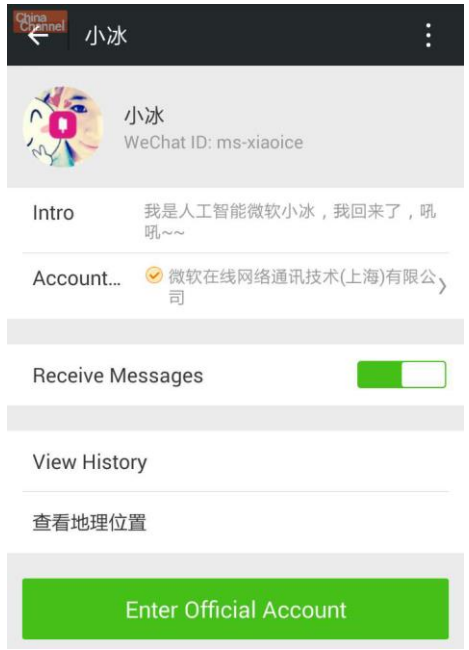
人工智能系统 System for AI

深度学习推理系统
Inference systems

主要内容

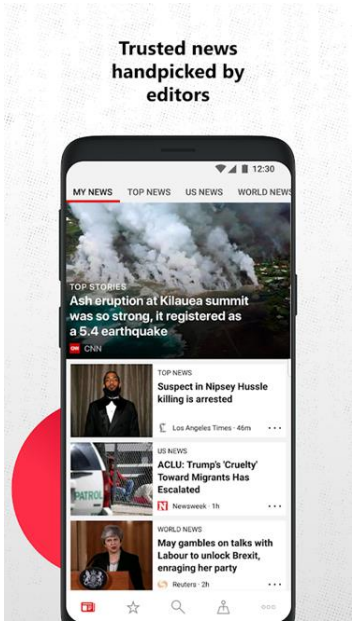
- 推理(Inference)系统简介
- 推理系统设计与优化
  - 延迟(Latency)
  - 吞吐(Throughput)
  - 效率(Efficiency)
- 部署(Deployment)
  - 扩展性(Scalability)
  - 灵活性(Flexibility)

典型深度学习推理应用



对话机器人

(e.g. Microsoft Xiao Ice, etc.)



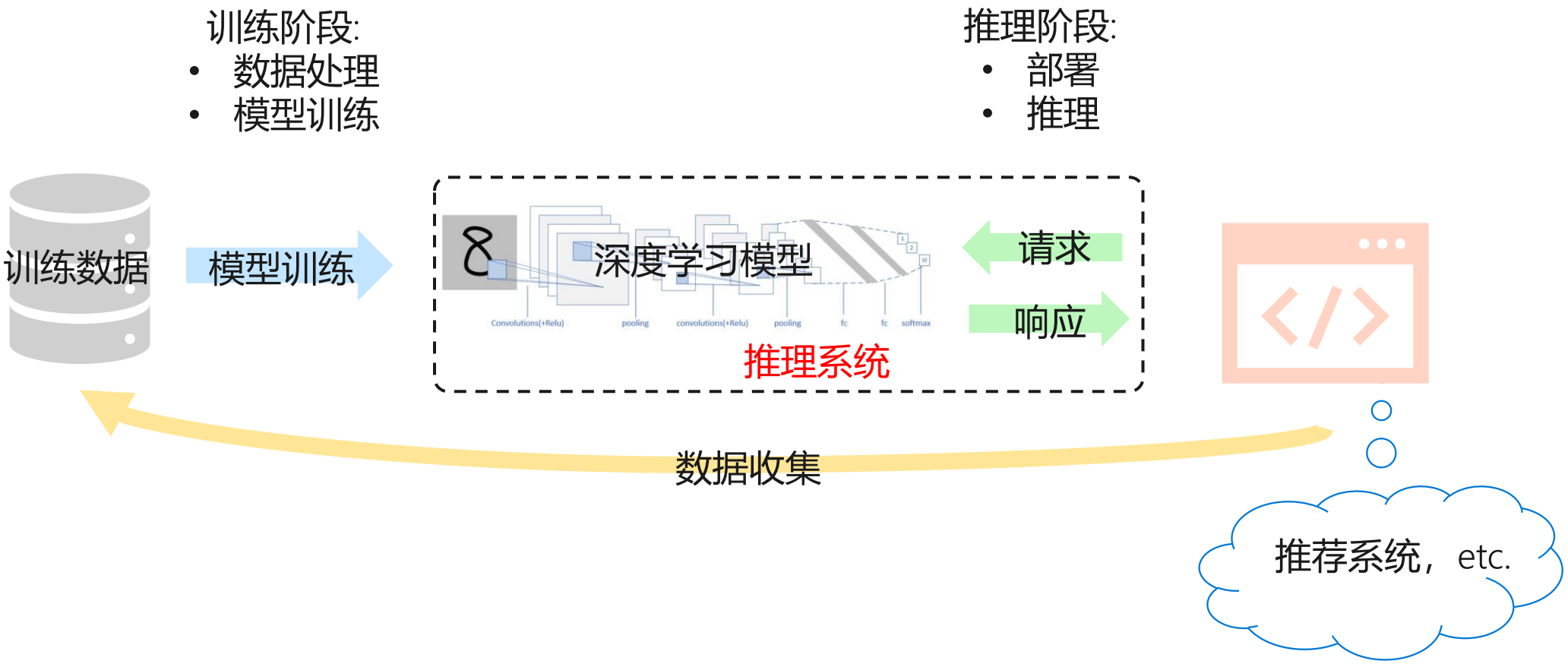
新闻推荐系统

(e.g. Bing News, etc.)

物体检测

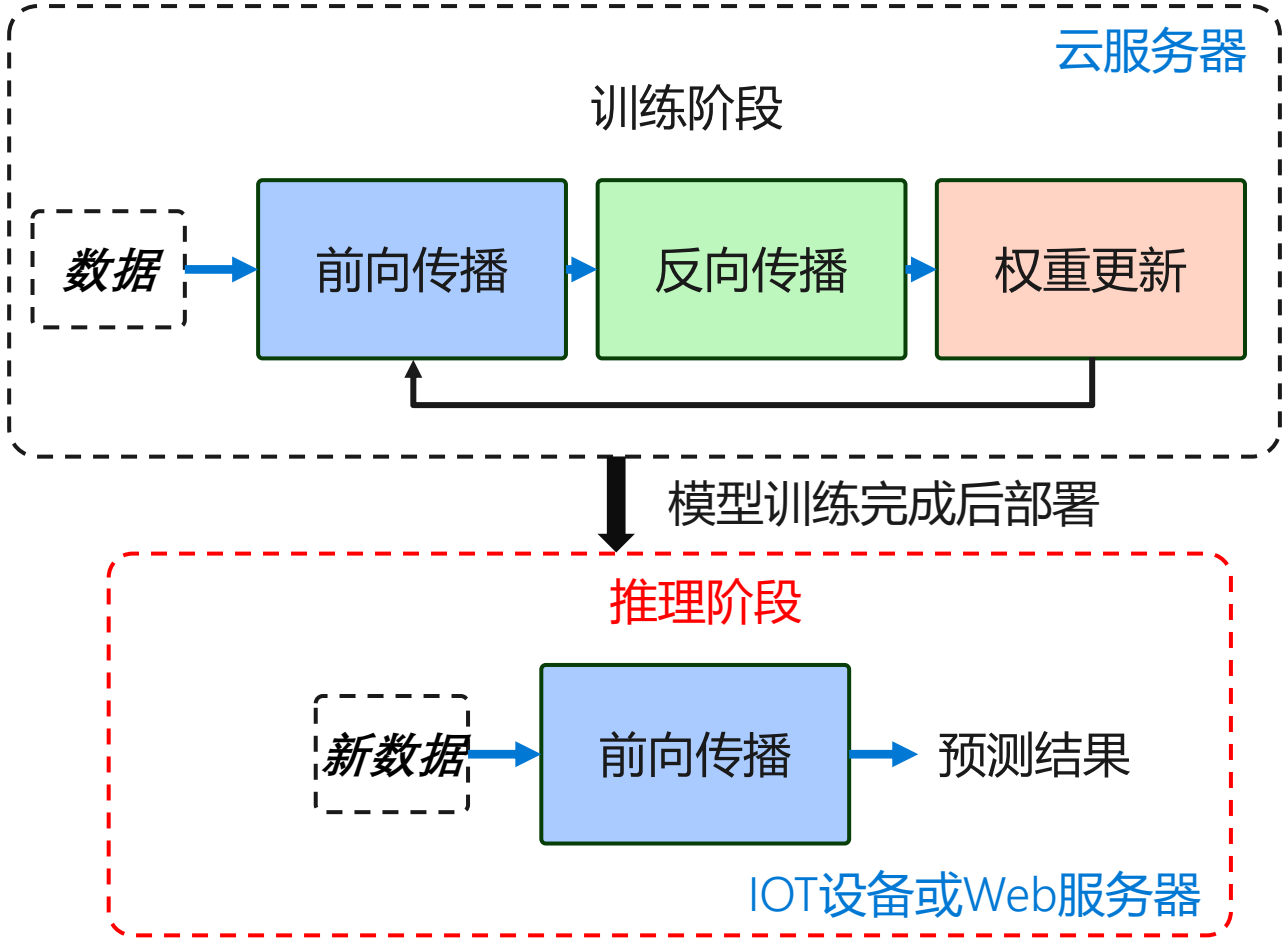
(e.g. Azure Cognitive Services, etc.)

深度学习模型的生命周期



推理相比训练的新特点与挑战

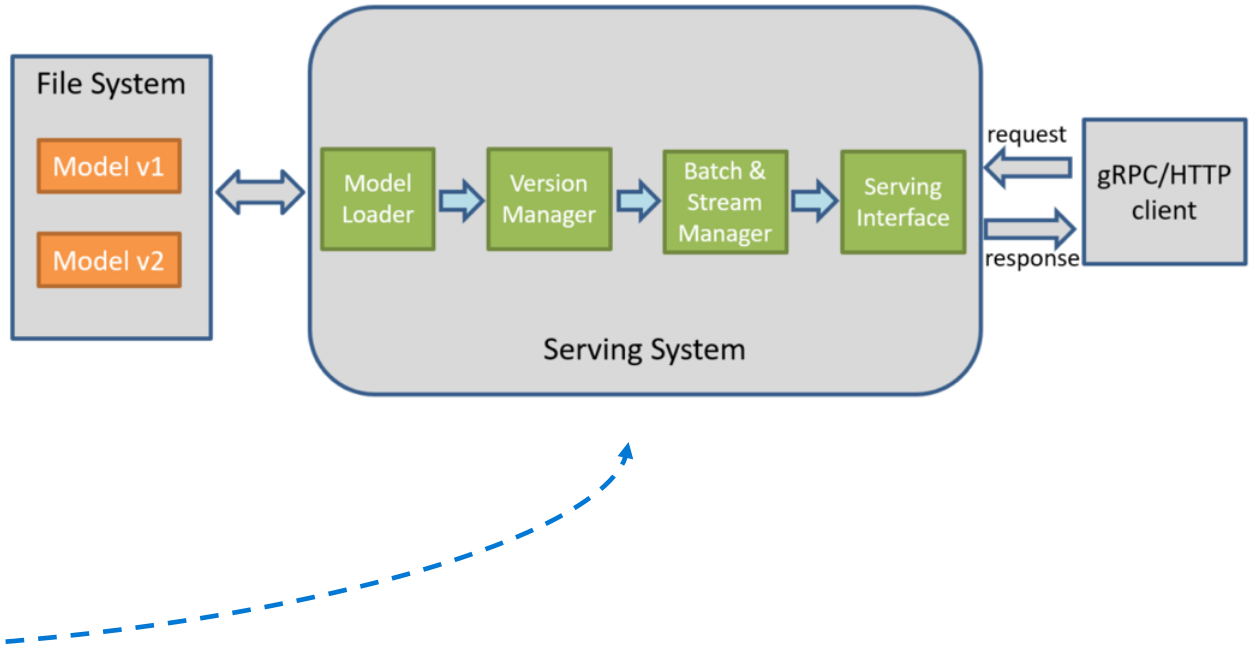
- 模型被部署为长期运行的服务才能被使用
  - 服务有明确对请求的低延迟高吞吐需求
- 推理有更苛刻的资源约束
  - 更小的内存，更低的功耗等
- 推理不需要反向传播梯度下降
  - 可以牺牲一定的数据精度
- 部署的设备型号更加多样
  - 需要定制化的优化



模型部署与推理

```
runtime = trt.infer.create_infer_runtime(G_LOGGER)
context = engine.create_execution_context()
output = np.empty(1000, dtype = np.float32)

# Alocate device memory
d_input = cuda.mem_alloc(1 * img.nbytes)
d_output = cuda.mem_alloc(1 * output.nbytes)
bindings = [int(d_input), int(d_output)]
stream = cuda.Stream()
# Transfer input data to device
cuda.memcpy_htod_async(d_input, img, stream)
# Execute model
context.enqueue(1, bindings, stream.handle, None)
# Transfer predictions back
cuda.memcpy_dtoh_async(output, d_output, stream)
# Syncronize threads
stream.synchronize()
```



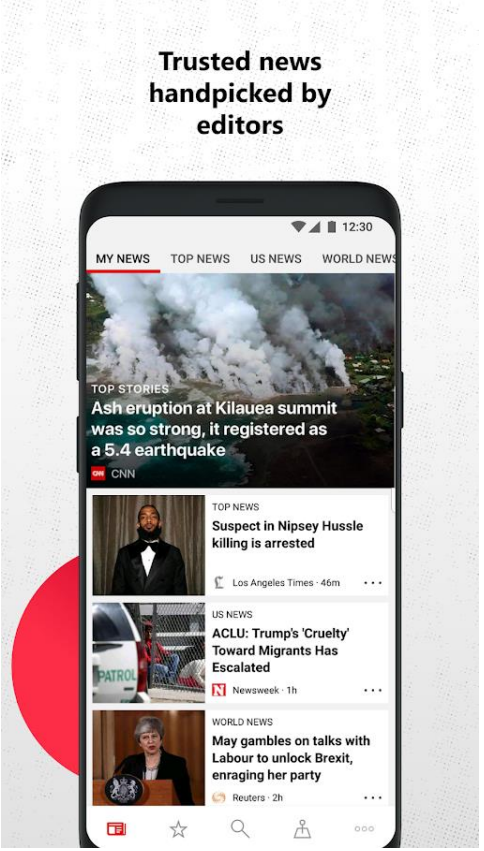
MMdnn: Using TensorRT to Accelerate Inference

在线推荐系统的服务需求

某在线新闻APP公司希望部署内容个性化推荐服务

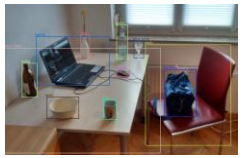
期望该服务能满足以下需求：

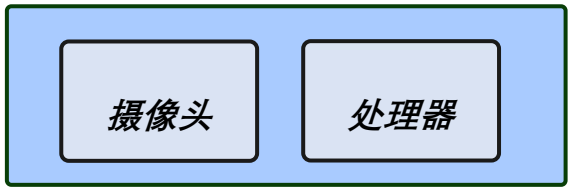
- 低延迟：
  - 互联网上推荐文章延迟（<100毫秒）
- 高吞吐：
  - 突发新闻驱动的暴增人群的吞吐量需求
- 扩展性：
  - 扩展到不断增长的庞大的用户群体
- 准确度：
  - 随着新闻和读者兴趣的变化提供准确的预测



IOT视觉应用的资源使用需求

Azure GPU Standard_NC6s_v3
1 × nvidia V100 GPU
7TFLOPS 3.0$ / hour





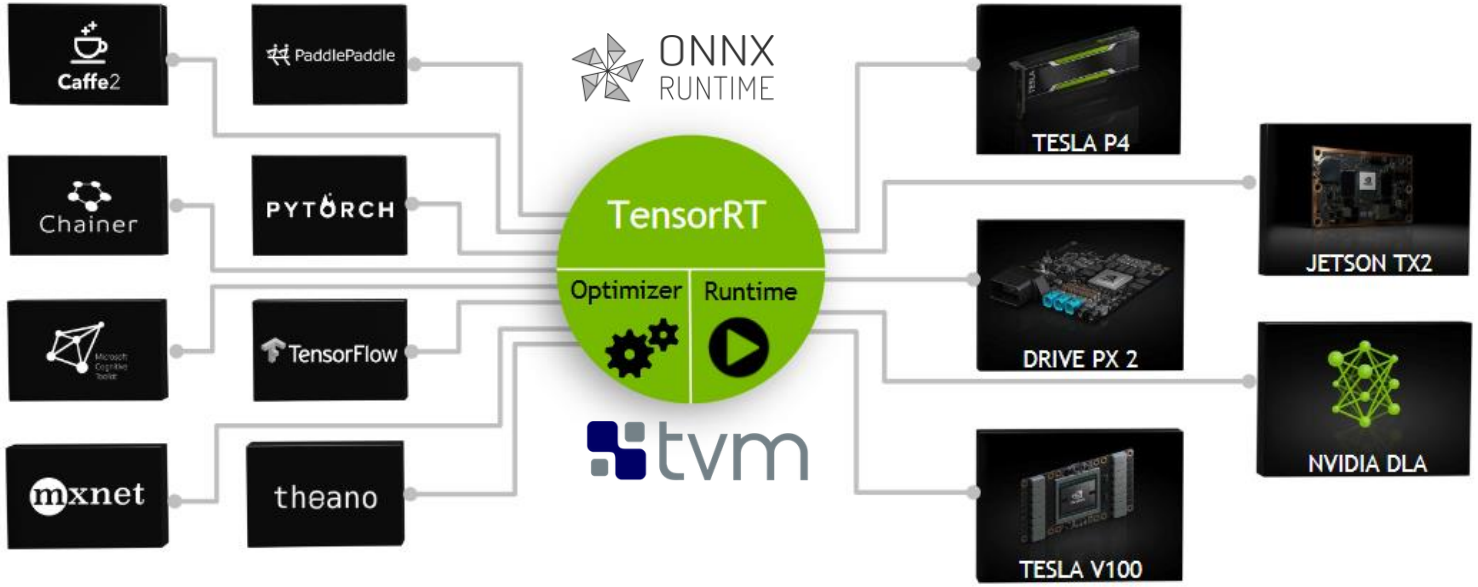
工作负载	深度学习 300GLOPS => 30GFLOPs / frame, 10FPS	
	设备功耗	云端开销
预算	30% of 10Wh for 10h = 300mW	$10 人/年
计算能力	9GFLOPS	2.25GFLOPS(GPU)

计算能力和工作负载需求差距较大

推理系统部署灵活性需求

机器学习服务的部署，优化和维护困难且容易出错

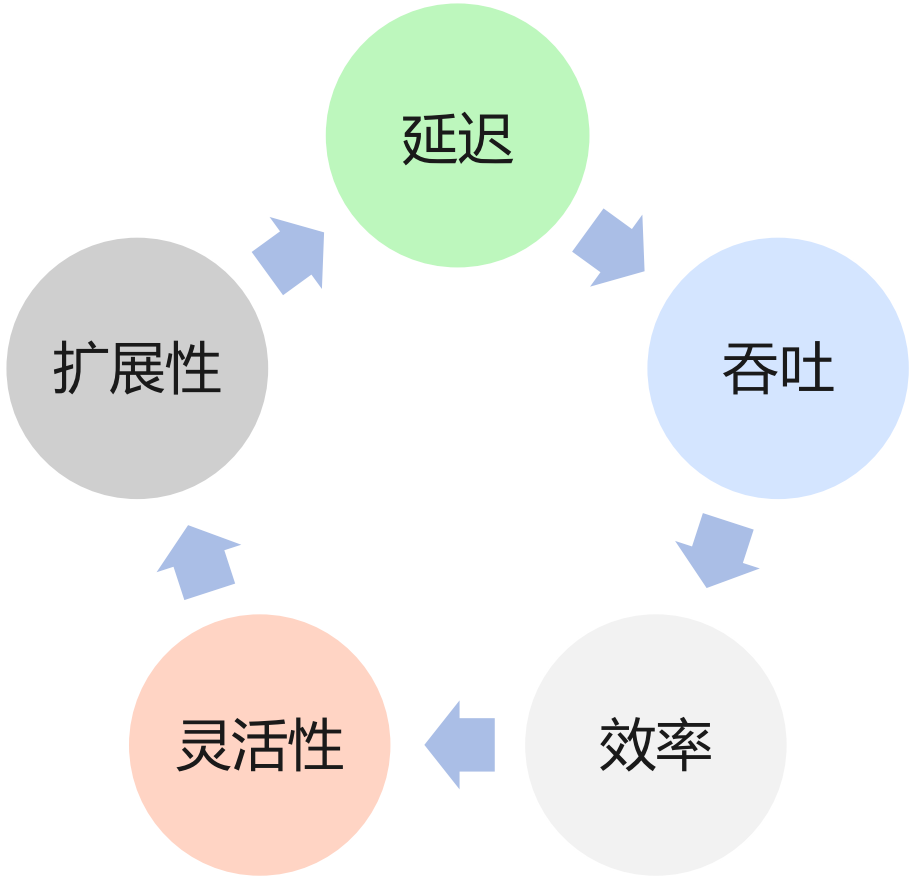
- 大多数框架都是为训练设计和优化
- 开发人员需要将必要的软件组件拼凑在一起
- 跨多个不断发展的框架集成和推理需求
- 多种部署硬件的支持



DEEP LEARNING DEPLOYMENT WITH NVIDIA TENSORRT

设计推理系统的优化目标

- 延迟(Latency):
  - 满足服务等级协议的延迟
- 吞吐量(Throughputs):
  - 暴增负载的吞吐量需求
- 效率(Efficiency):
  - 高效率，低功耗使用GPU, CPU
- 灵活性(Flexibility):
  - 支持多种框架, 提供构建不同应用的灵活性
- 扩展性(Scalability):
  - 扩展支持不断增长的用户或设备



推理系统的约束

- SLA对延迟的约束
- 资源约束
  - 设备端电池约束
  - 设备与服务端内存约束
  - 云端资源的预算约束
  - …
- 准确度(Accuracy)约束
  - 使用近似模型产生的一些误差可以接受

小结与讨论

- 深度学习推理系统设计需要考虑多目标和约束
- 推理系统相比传统服务系统有哪些新的挑战?
- 云和端的服务系统有何不同的侧重和挑战?

主要内容

- 推理(Inference)系统简介
- 推理系统设计与优化
  - 延迟(Latency)
  - 吞吐(Throughput)
  - 效率(Efficiency)
- 部署(Deployment)
  - 扩展性(Scalability)
  - 灵活性(Flexibility)

延迟(Latency)

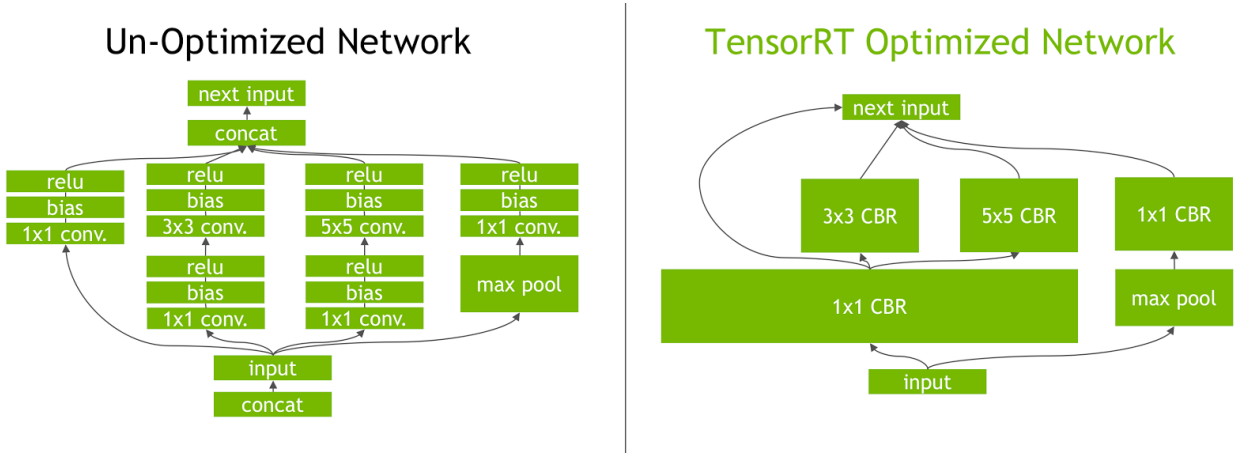
- 推理(inference)延迟：
  - 延迟是在给出查询后呈现推理结果所花费的时间。
  - 推理服务通常位于关键路径上，因此预测必须既快速同时满足有限的尾部延迟(Tail Latency)才能满足服务水平协议(SLA)
- 需要低延迟的原因：
  - SLA: 次秒(Sub-second)级别延迟服务水平协议
- 低延迟的挑战：
  - 交互式应用程序的低延迟需求通常与离线批处理训练框架设计的目标不一致
  - 简单的模型速度快，复杂的模型更加准确，但是浮点运算量更大
  - 次秒(Sub-second)级别延迟约束制了批尺寸(Batch Size)
  - 模型融合或多租容易引起长尾延迟(Long Tail Traffic)现象

低延迟的策略

- 模型优化
  - 层间融合或张量融合(Layer & Tensor Fusion)
  - 目标后端自动调优
  - 内存分配策略调优
- 降低一定的准确度
  - 低精度推理与精度校准(Precision Calibration)
  - 模型压缩(Model Compression)
- 自适应批尺寸(Batch Size)
- 缓存(Caching)结果

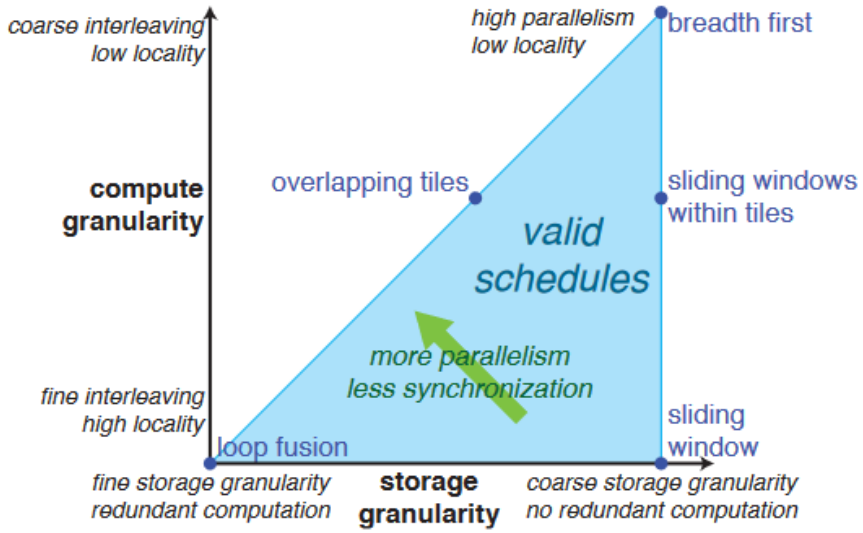
层间与张量融合(Layer and Tensor Fusion)

- 融合的原因：
  - 相对于内核启动开销和每个层的张量数据读写成本，内核(Kernel)计算通常非常快
  - 导致内存带宽瓶颈和可用GPU资源的利用不足
- 目标：最小化GPU访存和最大化GPU资源利用率
- 搜索计算图的最优融合策略

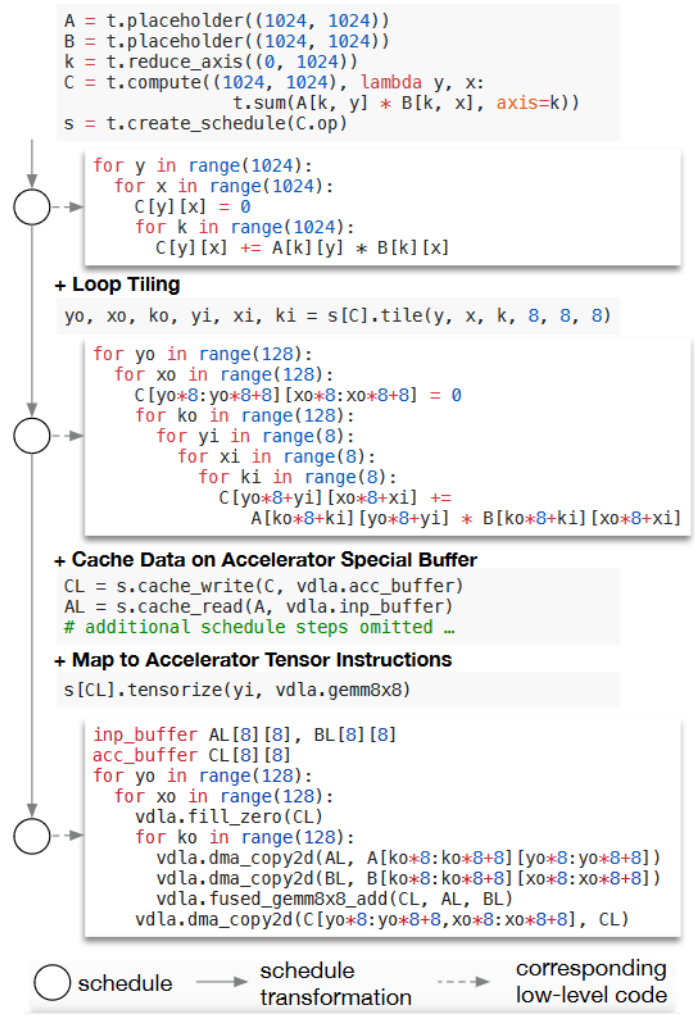


TensorRT 3: Faster TensorFlow Inference and Volta Support

内核(Kernel)目标后端自动调优



调度策略空间



矩阵乘在特定加速器调度优化实例

Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines

TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

内存分配策略调优

- 设备或服务端内存是紧缺资源
- 目标：最小化内存占用和内存分配调用开销
  - 仅在每个张量为其分配内存
- 约束：保证延迟SLA
- 策略：
  - Cached Allocator
  - Swap: Pre-fetching and Off-loading
  - Operator Fusion

低精度推理

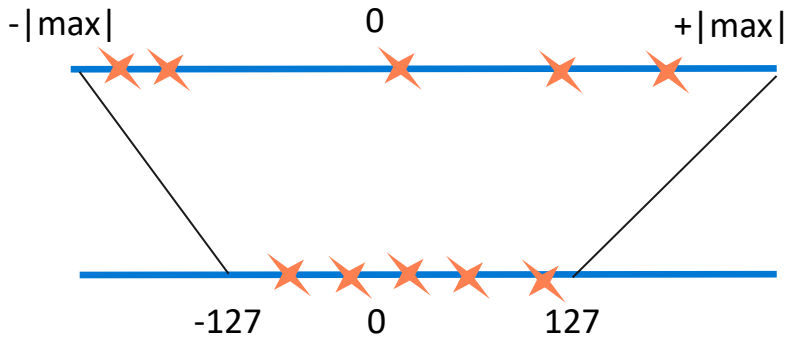
- 推理阶段可以适当降低精度
  - 大多数深度学习框架都以完整的32位精度（FP32）训练神经网络
  - 对模型进行充分训练后，由于不需要进行梯度反向传播，因此推理计算可以使用半精度FP16甚至INT8张量运算
  - 使用较低的精度会导致较小的模型大小，较低的内存利用率和延迟以及较高的吞吐量
- 目标：最小化数据精度
- 约束：准确度损失

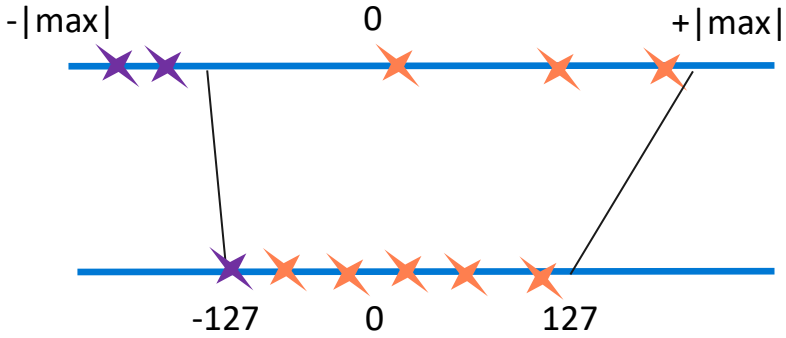
	Dynamic Range	Min Positive Value
FP32	$-3.4 \times 10^{38} \sim +3.4 \times 10^{38}$	-1.4×10^{-45}
FP16	$-65504 \times +65504$	5.96×10^{-8}
INT8	$-128 \sim +127$	1

精度校准(Precision Calibration)

- INT8编码FP32同样信息:
  - Tensor Values = FP32 scale factor * int8 array
  - 添加饱和阈值准确度损失更低
- 目标：最小化信息损失
- KL散度度量
  - P, Q - two discrete probability distributions.
  - KL_divergence(P,Q):= SUM(P[i] * log(P[i] / Q[i]), i)
- 策略:
  - 校准数据集运行FP32推理
  - 对每层数据收集激活输出直方图
  - 生成使用不同饱和阈值产生的量化输出分布
  - 选择最小化激活输出分布与量化后的激活输出分布之间KL散度阈值

- 无饱和，映射|max|为127
- 有明显准确度损失





- 提升了每层激活输出的准确度
- 如何选择最优饱和阈值

8-bit Inference with TensorRT

模型压缩问题定义

- 模型压缩的收益：
  - 减少浮点运算量，降低延迟
  - 减少内存占用，提升利用率
- 定义模型压缩问题

$$\min_{Policy_i} \{Model\_Size(Policy_i)\}$$

- 约束

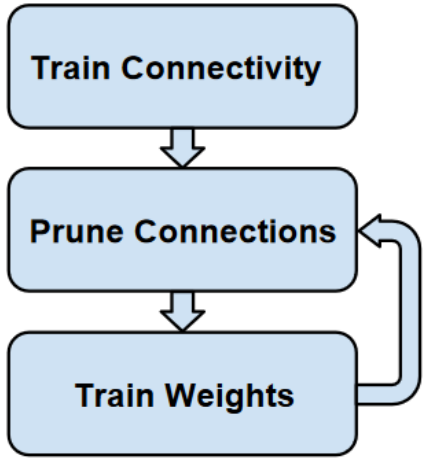
$$accuracy(Policy_i) \geq accuracy\_sla$$

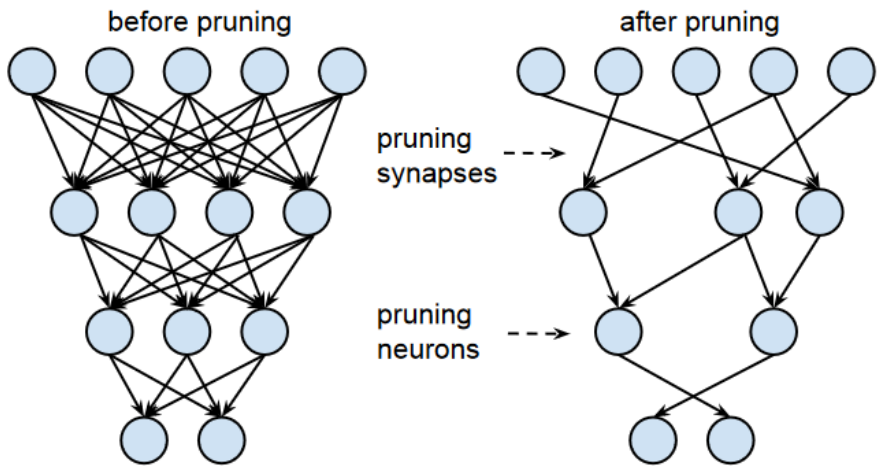
模型压缩策略

- 参数裁剪和共享（Parameter Pruning and Sharing）
  - 剪枝(Pruning)
  - 量化(Quantization)
  - 编码(Encoding)
- 低秩分解（Low-rank Factorization）
- 迁移/压缩卷积滤波器（Transferred / Compact Convolutional Filter）
- 知识精炼（Knowledge Distillation）

剪枝算法

- Train Connectivity
  - 通过正常的网络训练来学习连通性
  - 但是，与常规训练不同，不是为了学习权重的最终值，而是在学习哪些连接重要
- Prune Connections
  - 所有权重低于阈值的连接都将删除，从而将稠密网络转换为稀疏网络
- Train Weights
  - 对剩余网络权重进行重新训练
  - 如果使用修剪后的网络而不进行重新训练，则准确性会受到很大影响





Learning both Weights and Connections for Efficient Neural Networks

小结与讨论

- 优化延迟的目标，受到空间与准确度的约束
- 层间与张量融合受到哪些约束?
- 推理和训练优化内存分配策略的侧重点是否有不同?

主要内容

- 推理(Inference)系统简介
- 推理系统设计与优化
  - 延迟(Latency)
  - 吞吐(Throughput)
  - 效率(Efficiency)
- 部署(Deployment)
  - 扩展性(Scalability)
  - 灵活性(Flexibility)

吞吐量(Throughputs)

- 需要高吞吐的目的
  - 突发的请求数量暴增
  - 不断扩展的用户和设备
- 达到高吞吐的策略：
  - 利用加速器并行
    - 批处理请求
    - 利用优化的BLAS矩阵运算库, SIMD指令和GPU等加速器
  - 自适应批尺寸(Batch Size)
  - 多模型装箱使用加速器
  - 容器扩展副本(Replica)部署

提升批处理(Batch Size)可以提升吞吐量

对于高请求数量和频率的场景

- 通过大的批处理(Batch Size)可以提升吞吐
- 但是需要满足一定的延迟约束

V100 Inference Performance

Network	Network Type	Batch Size	Throughput	Efficiency	Latency	GPU
GoogleNet	CNN	1	1610 images/sec	15 images/sec/watt	0.62	1x V100
	CNN	2	2162 images/sec	18 images/sec/watt	0.93	1x V100
	CNN	8	5368 images/sec	35 images/sec/watt	1.5	1x V100
	CNN	82	11869 images/sec	45 images/sec/watt	6.9	1x V100
	CNN	128	12697 images/sec	47 images/sec/watt	10	1x V100

https://developer.nvidia.com/deep-learning-performance-training-inference

吞吐量优化问题

- 定义优化问题

$$\max_{batch\_size} Throughput(batch\_size)$$

- 约束

$$latency(batch\_size) + overhead(batch\_size) \leq latency\_sla$$

动态批处理尺寸(Batch Size)

Additive Increase Multiplicative Decrease (AIMD) 策略

- 加性增加(Addictive Increase):
  - 将批次大小累加增加固定数量，直到处理批次的延迟超过目标延迟为止
- 乘性减少(Multiplicative Decrease):
  - 当达到后，执行一个小的乘法回退，将批次大小减少了10%
  - 因为最佳批次大小不会大幅波动，所以使用的退避常数要比其他AIMD方案小得多

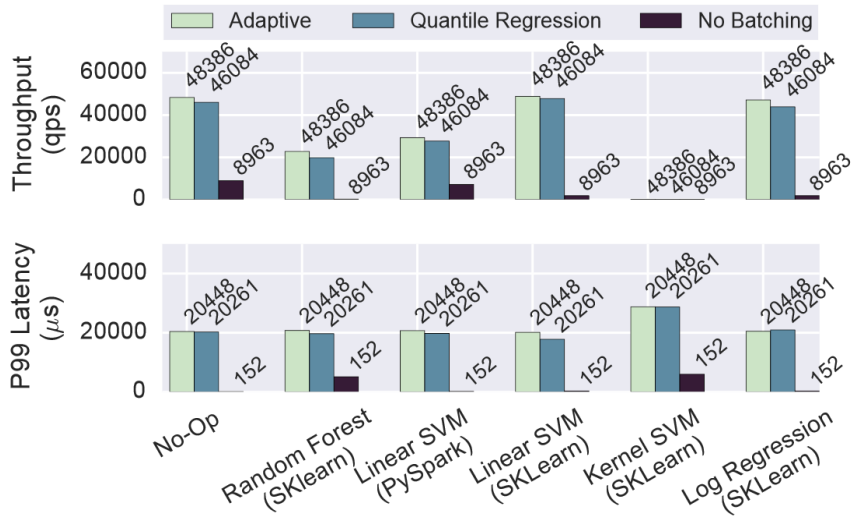


Figure 4: Comparison of Dynamic Batching Strategies.

主要内容

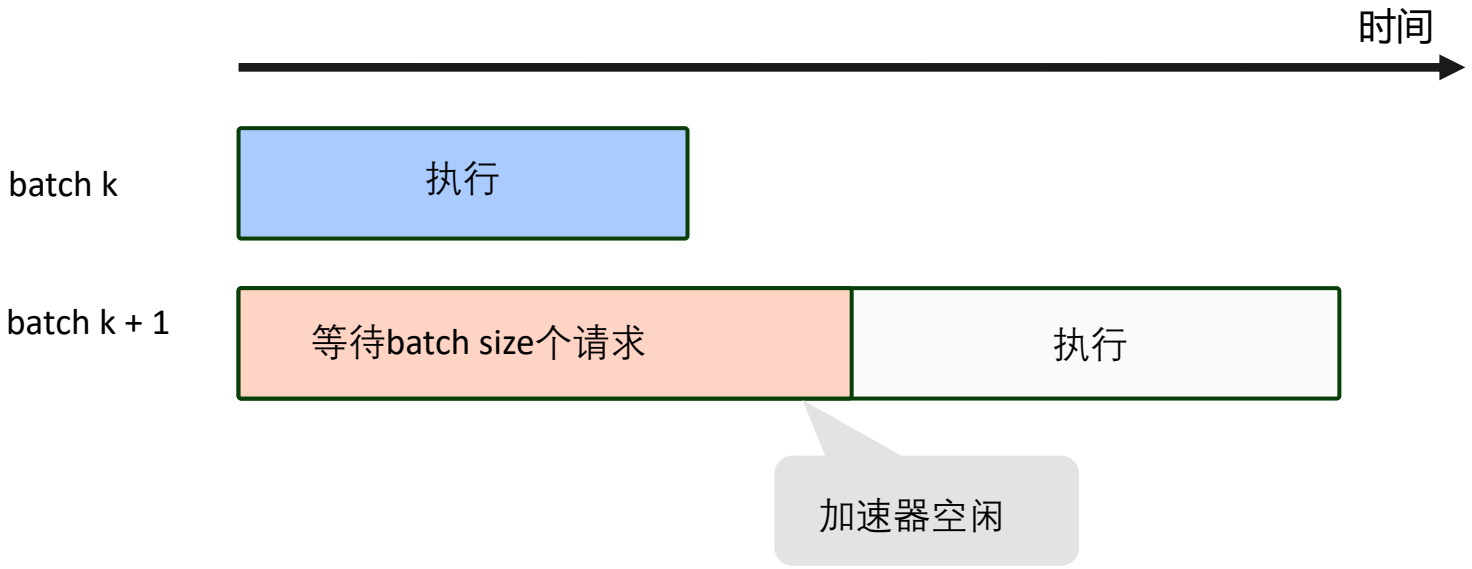
- 推理(Inference)系统简介
- 推理系统设计与优化
  - 延迟(Latency)
  - 吞吐(Throughput)
  - 效率(Efficiency)
- 部署(Deployment)
  - 扩展性(Scalability)
  - 灵活性(Flexibility)

效率(Efficiency)

- 需要高效(High Efficiency)的原因
  - 资源(内存等)约束
  - 移动端有功耗的约束
  - 云端有预算(Budget)的约束
- 高效率策略：
  - 模型压缩
  - 高效使用GPU
    - 利用加速器并行和优化模型执行
  - 装箱(bin-packing)使用加速器

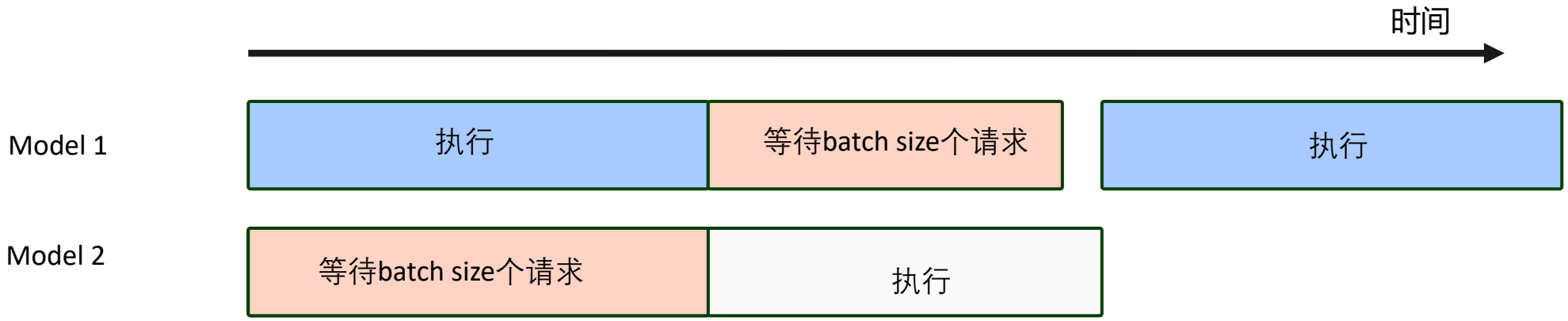
加速器的低效率使用

由于等待批处理请求，可能造成GPU空闲



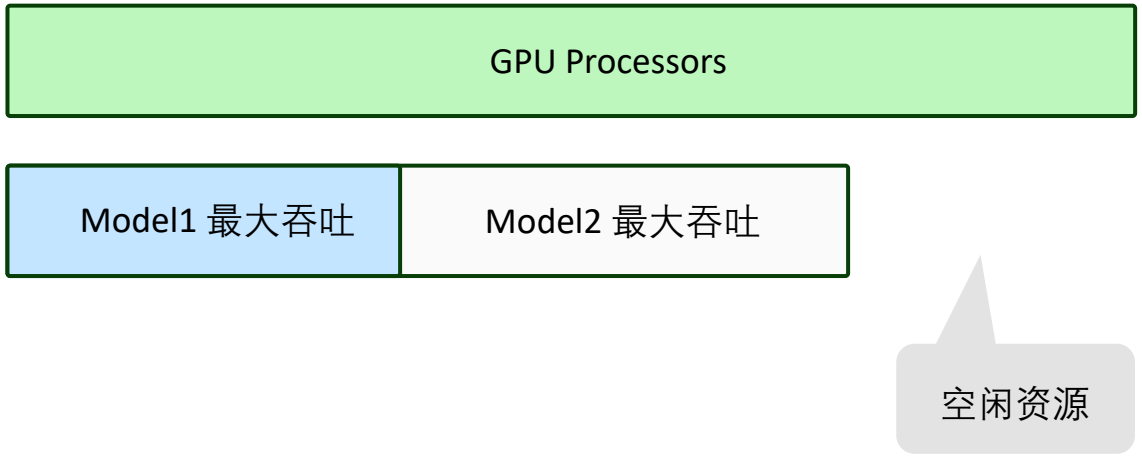
单加速器运行多模型

- 时分复用策略，将等待时间给其他模型进行执行
- 同时可以动态调整Batch Size减少空闲时间



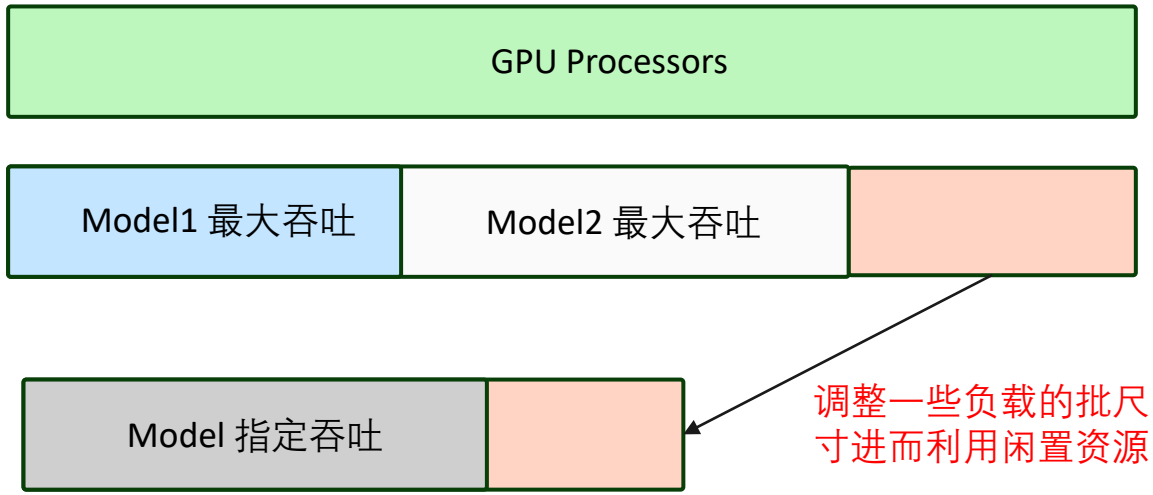
加速器的低效率使用

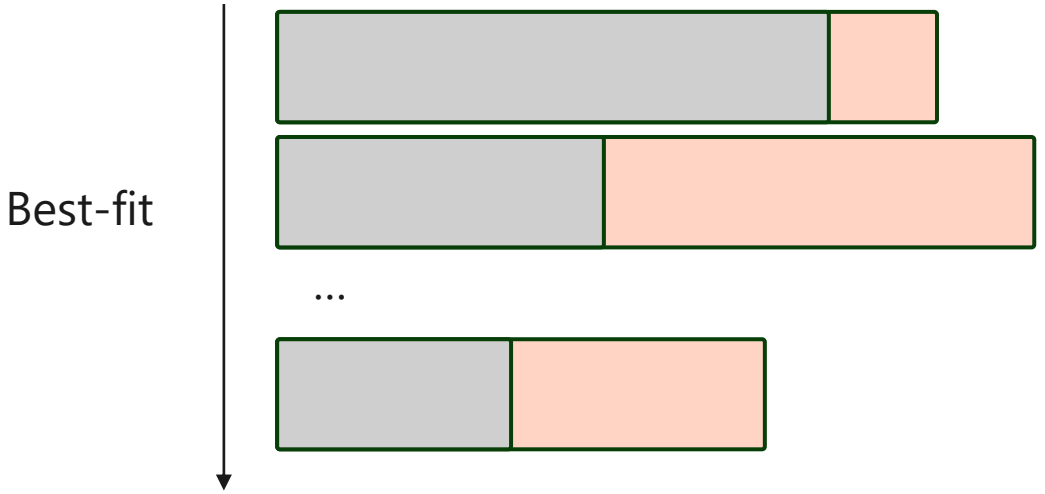
在延迟SLA约束下，模型在指定的GPU下按最大吞吐量进行分配，但是可能仍有空闲资源



装箱

Best-fit策略装箱(bin-packing)
碎片化的模型请求





小结与讨论

- 当前吞吐量和效率的优化策略是否会对延迟产生影响?
- 设计其他策略进行吞吐量或使用效率的优化?

主要内容

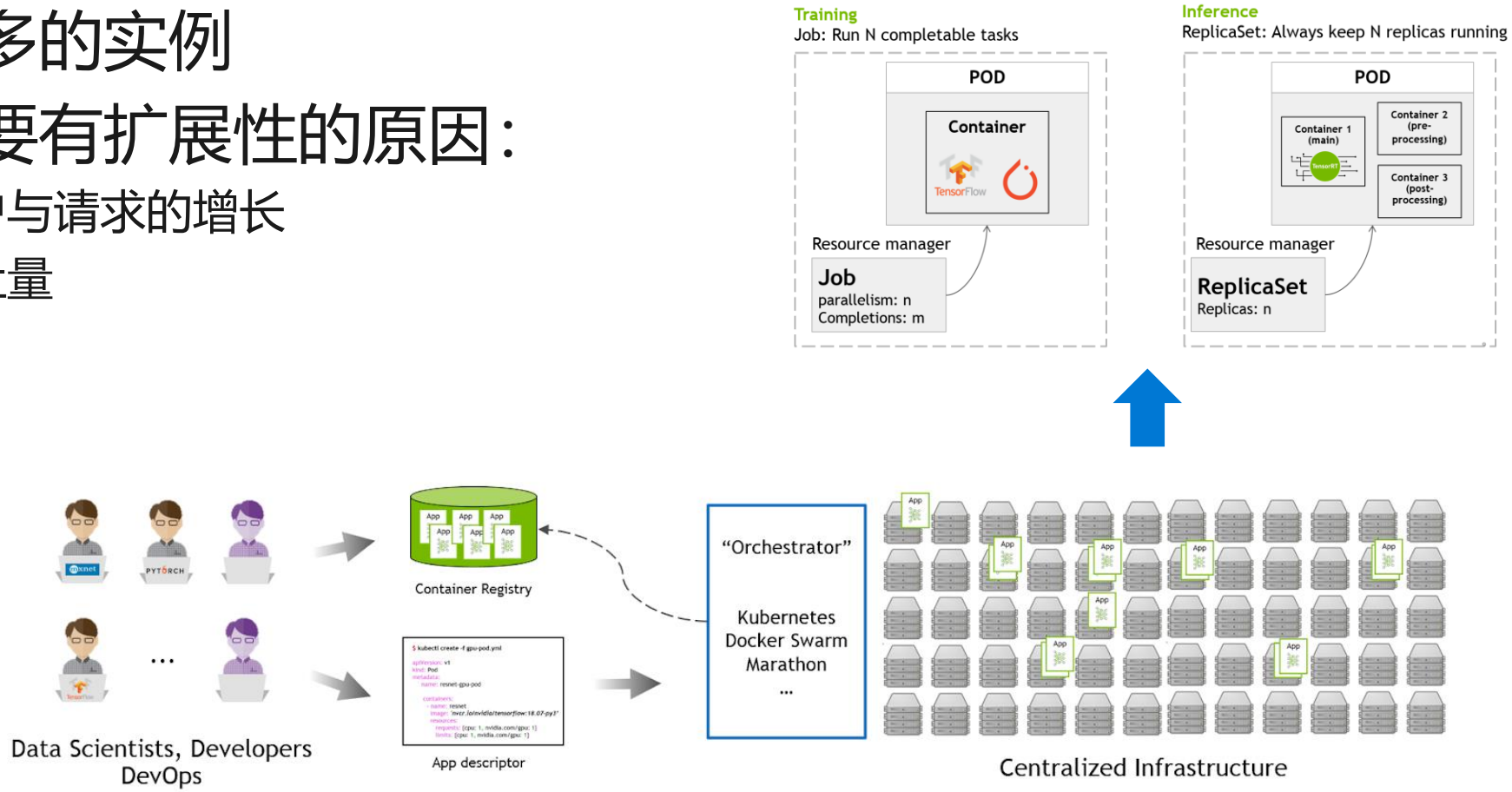
- 推理(Inference)系统简介
- 推理系统设计与优化
  - 延迟(Latency)
  - 吞吐(Throughput)
  - 效率(Efficiency)
- 部署(Deployment)
  - 扩展性(Scalability)
  - 灵活性(Flexibility)

部署(Deployment)

- 扩展性(Scalability)
- 灵活性(Flexibility)
- 模型版本(Version)管理

扩展性

- 随着请求负载增加而自动和动态的部署更多的实例
- 系统需要有扩展性的原因：
  - 应对用户与请求的增长
  - 提升吞吐量



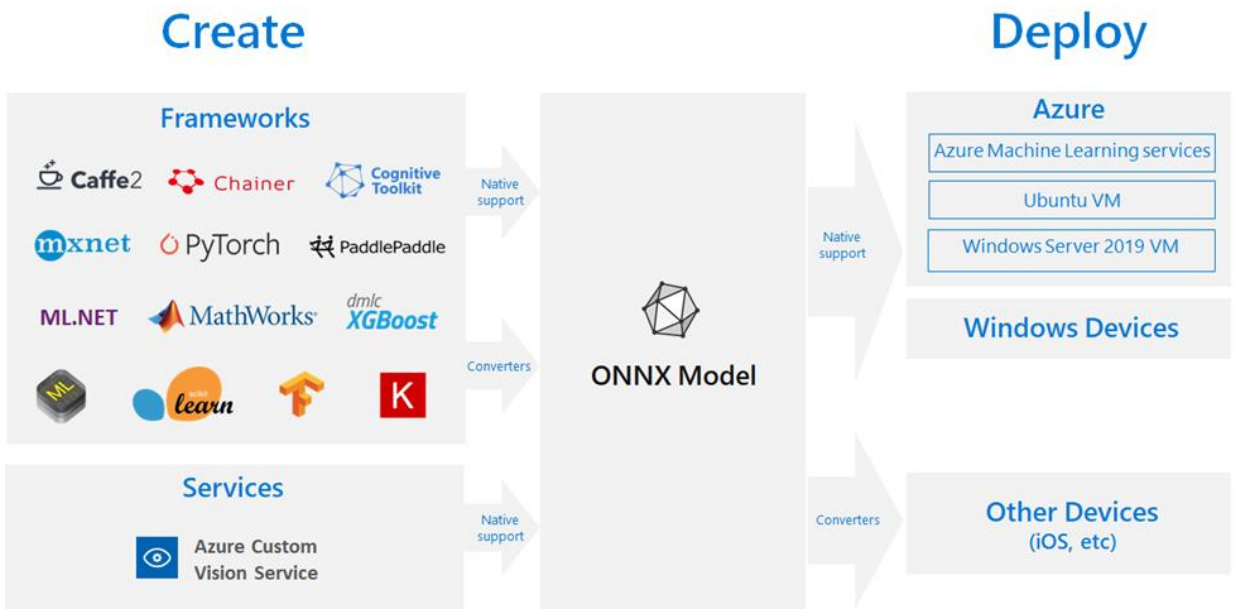
https://devblogs.nvidia.com/kubernetes-ai-hyperparameter-search-experiments/

部署灵活性(Flexibility)

- 服务系统需要灵活性的原因：
  - 支持加载不同框架的模型
  - 框架不断的更新，大多数是为训练优化，有些框架甚至不支持在线推理
  - 与不同语言接口和不同逻辑的应用结合
- 解决方法：
  - 接口抽象:
    - 提供构建不同应用逻辑的灵活性
    - 提供不同框架的通用抽象
  - RPC：
    - 跨语言，跨进程通信
  - 深度学习模型开放协议：
    - 跨框架模型转换
  - 容器：
    - 运行时环境依赖与资源隔离

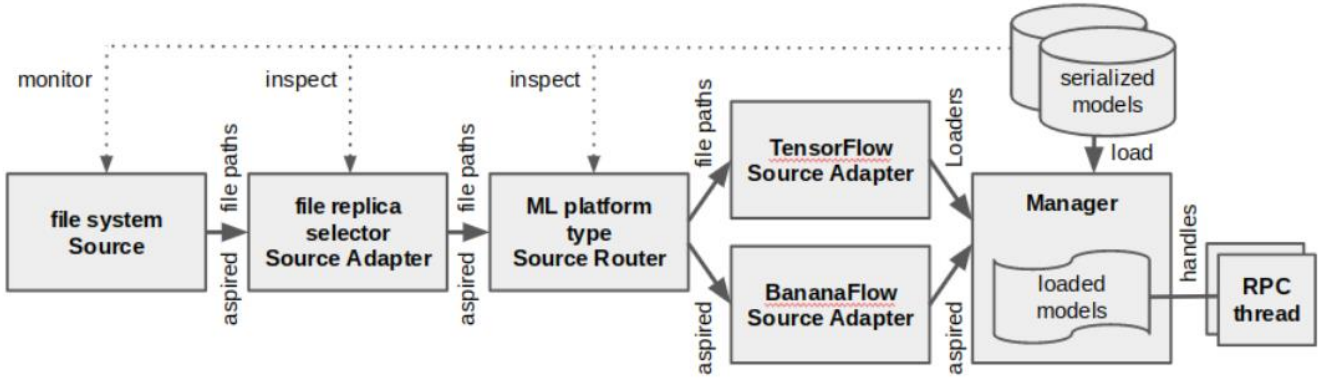
模型转换与开放协议

- MMdnn
  - 模型通过中间表达(IR)跨框架转换
- ONNX
  - 模型中间表达标准
  - 模型优化与部署(ONNX Runtime)



模型版本管理

- 需要模型版本管理的原因
  - 每隔一段时间训练出的新版本模型替换线上模型，但是可能存在缺陷
  - 如果新版本模型发现缺陷需要回滚
- 模型生命周期管理
  - 模型生命周期
  - 金丝雀(Canary)策略
  - 回滚(Rollback)策略



模型生命周期管理工作流实例

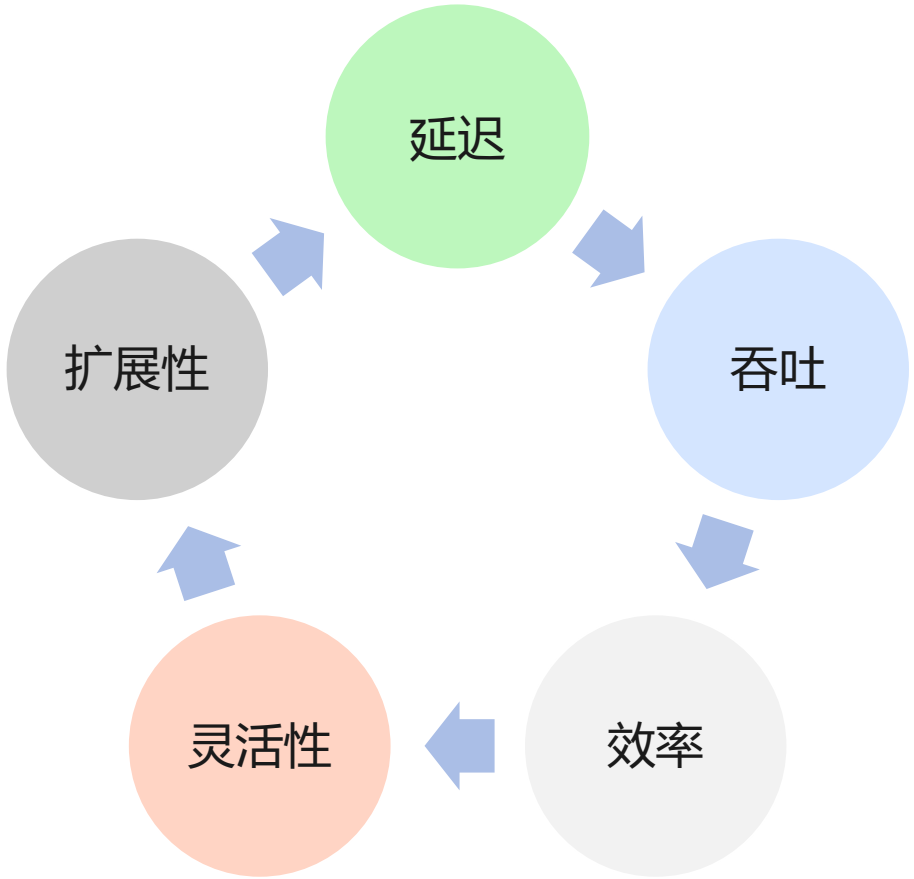
TensorFlow Serving: Flexible, High-Performance ML Serving

金丝雀(Canary)和回滚(Rollback)策略

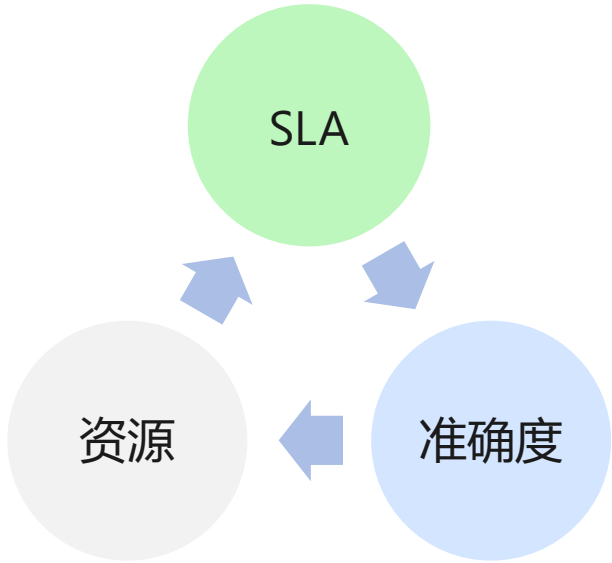
- 金丝雀策略
  - 当获得一个新训练的模型版本时，当前服务的模型成为第二新版本(second-newest)时，用户可以选择同时保持这两个版本
  - 将所有推理请求流量发送到当前两个版本，比较它们的效果
  - 一旦对最新版本达标，用户就可以切换到仅该版本
  - 方法需要更多的高峰资源，避免将用户暴露于缺陷模型
- 回滚策略
  - 如果在当前的主要服务版本上检测到缺陷，则用户可以请求切换到特定的较旧版本
  - 卸载和装载的顺序应该是可配置的
  - 当问题解决并且获取到新的安全版本模型时，从而结束回滚

TensorFlow Serving: Flexible, High-Performance ML Serving

小结



推理系统设计目标



约束

参考文献

- Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications
- Clipper: A Low-Latency Online Prediction Serving System
- TFX: A TensorFlow-Based Production-Scale Machine Learning Platform
- TensorFlow-Serving: Flexible, High-Performance ML Serving
- Optimal Aggregation Policy for Reducing Tail Latency of Web Search
- A Survey of Model Compression and Acceleration for Deep Neural Networks
- CSE 599W: System for ML - Model Serving
- https://developer.nvidia.com/deep-learning-performance-training-inference
- DEEP COMPRESSION:　COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING
- Learning both Weights and Connections for Efficient Neural Networks
- DEEP LEARNING DEPLOYMENT WITH NVIDIA TENSORRT
- Halide: A Language and Compiler for Optimizing Parallelism,Locality, and Recomputation in Image Processing Pipelines
- TVM: An Automated End-to-End Optimizing Compiler for Deep Learning
- 8-bit Inference with TensorRT