Apparate: Rethinking Early Exits to Tame Latency-Throughput Tensions in ML Serving

SOSP

Yinwei Dai ⋆ Rui Pan ⋆ Anand Iyer Kai Li Ravi Netravali

Princeton University Georgia Institute of Technology

汇报人：冯敏远

2024年11月10日

提纲

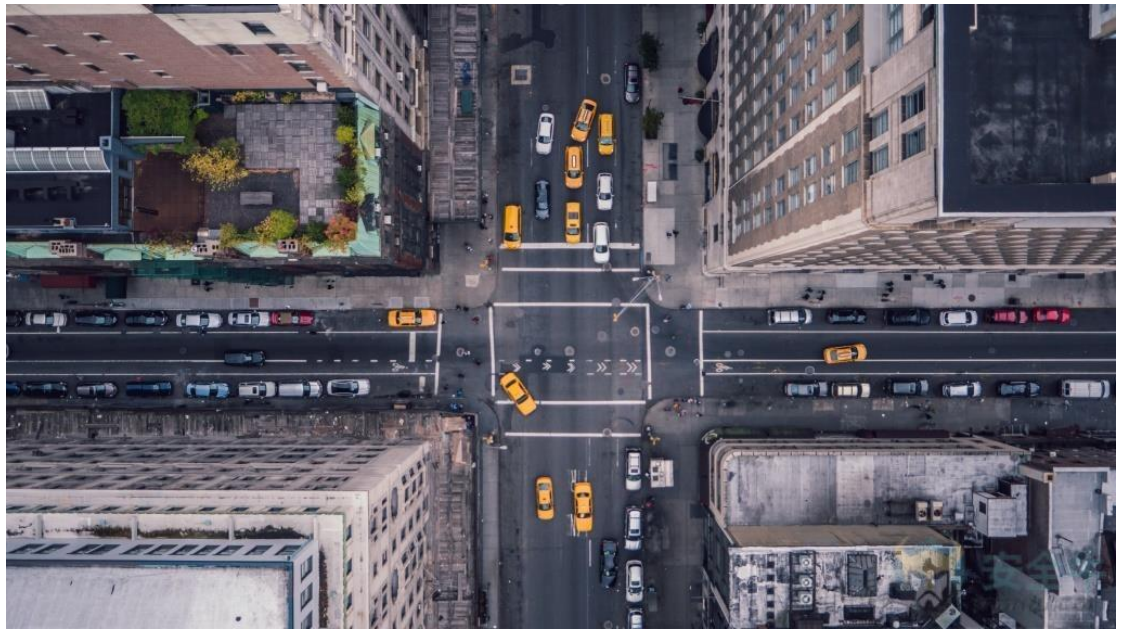
- 研究背景
- 研究问题
- 方法设计
- 实验评估
- 工作总结

提纲

- 研究背景
- 研究问题
- 方法设计
- 实验评估
- 工作总结

研究背景

流量分析



聊天机器人

研究背景

推荐引擎

语音助手

小爱同学　小艺　Siri　Bixby

随着用户需求的扩大，请求量上升，推理平台的高效管理至关重要

研究背景

服务平台

ONNX runtime **TensorFlow-Serving** **PyTorch Serve** **NVIDIA Triton Inference serve**

每天高达数十亿次的请求量

Service Level Objective

响应时间(10–100ms)

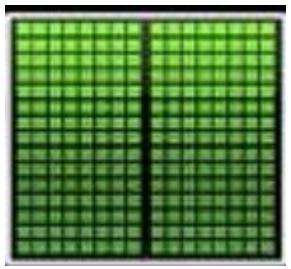
简称SLO

研究背景

延迟与吞吐量的关系

Batch

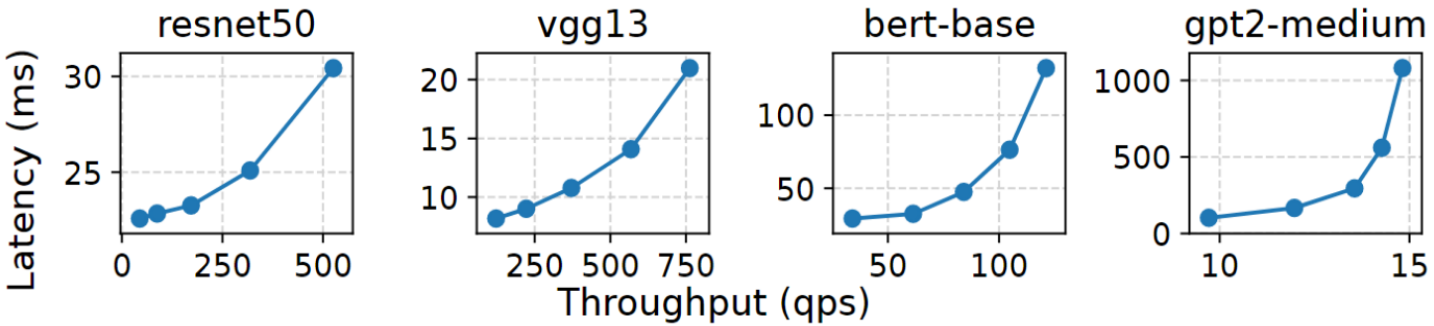
GPU并行处理



Batch Procesing

增加批处理量提高吞吐量

等待一个批次完成



吞吐量

时延

研究背景

目前服务平台的解决方案：

Service Level Objective

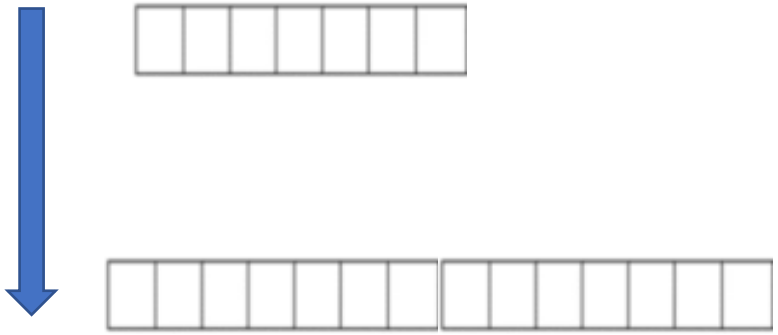
满足SLO的前提下，最大化批量大小

< threshold √

>= threshold ×

批量大小

时延



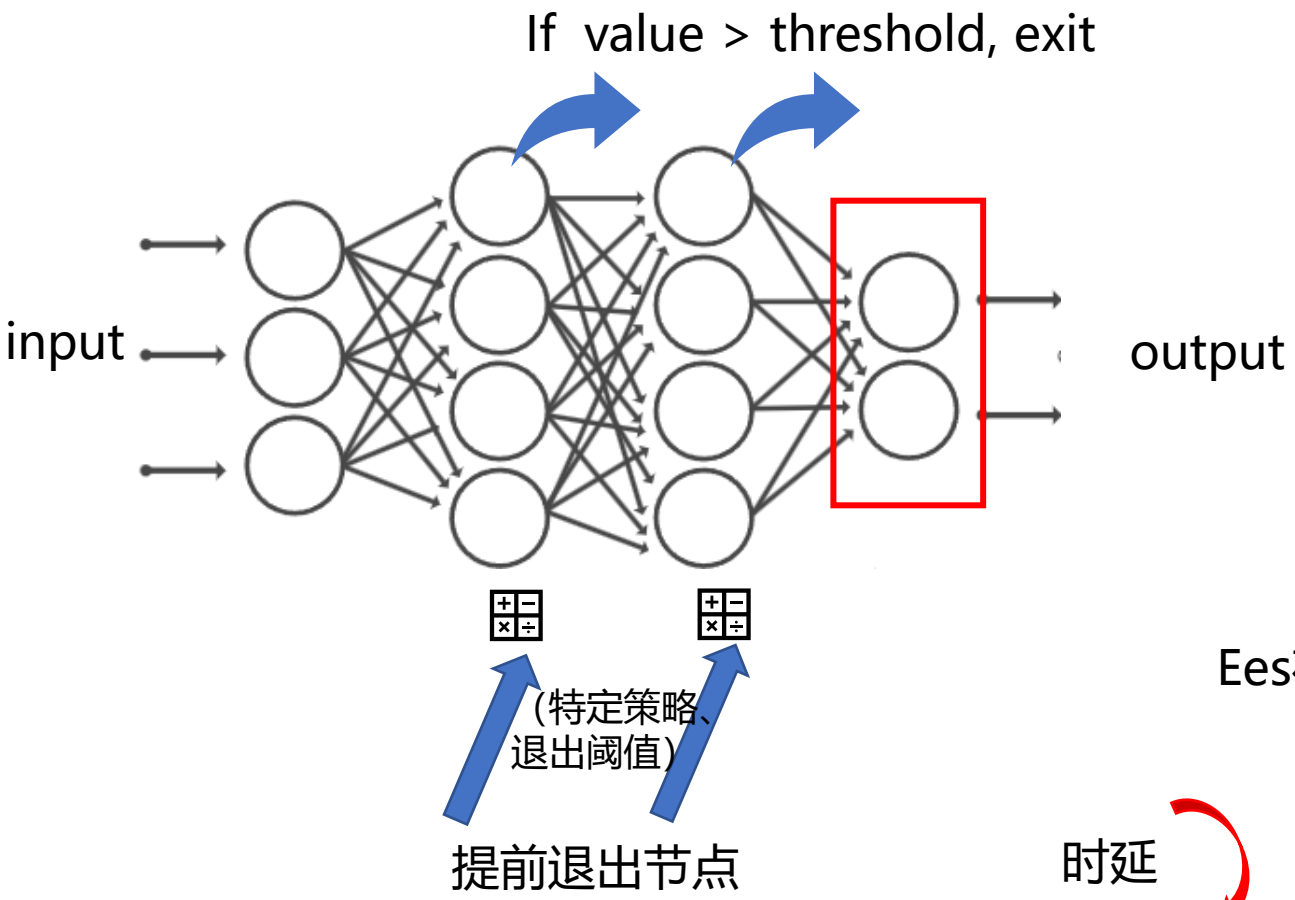
80ns/l00ns

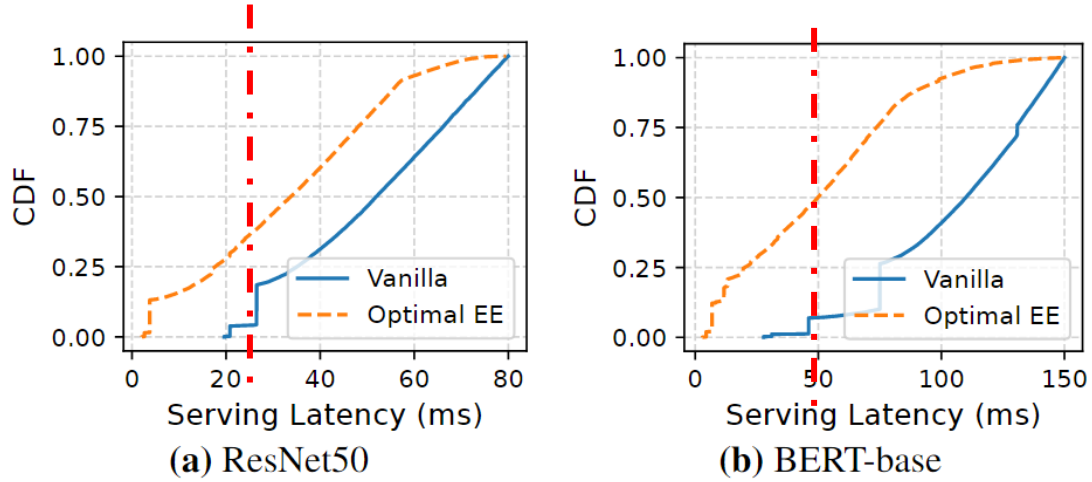
99.999ns/100ns

是否存在一个办法：可在不降低服务平台吞吐量的情况下，降低每次请求的延迟时间

研究背景

早期退出模型（Early Exits）





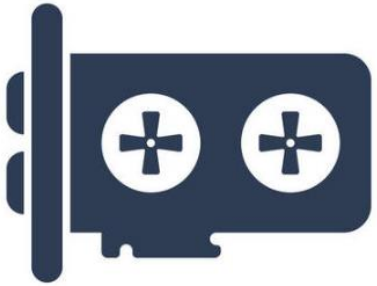
Ees在降低时延的情况下不降低吞吐量，但会降低准确性

时延 → 计算量 → 准确性

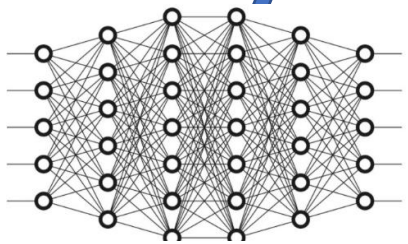
提前退出节点的作用是模拟完整模型的输出行为，以实现提前退出（EE），从而节省计算资源和降低延迟

研究背景

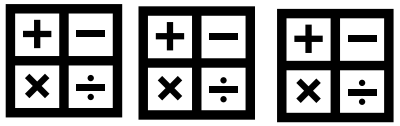
1.额外的资源开销



GPU Memory

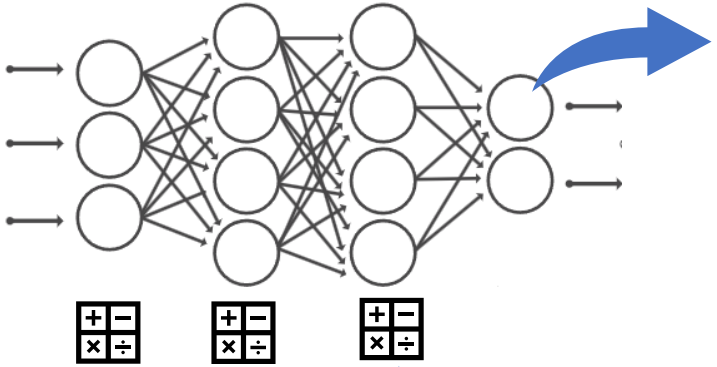


模型



早退出节点

2.缺乏准确性反馈



完整模型结果

是否准确?

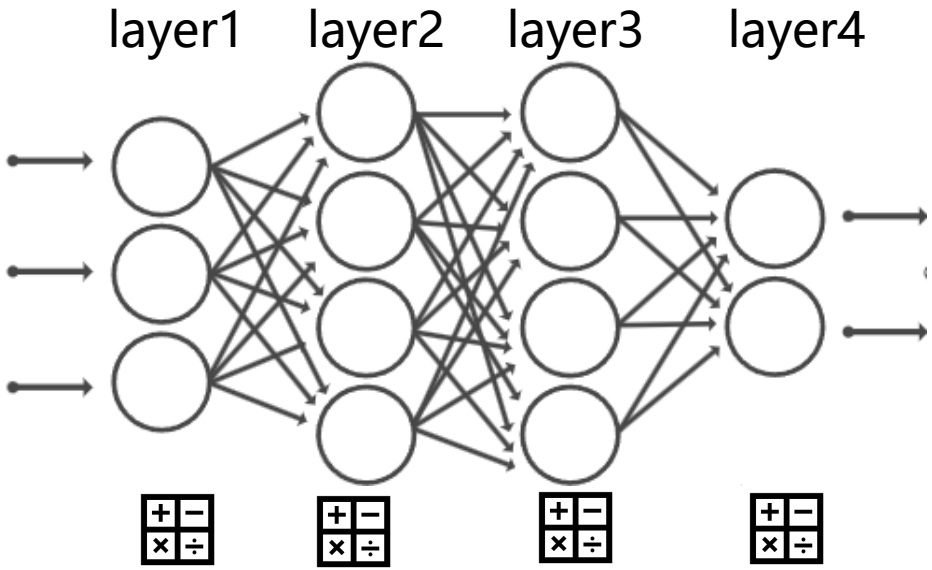
早退出结果（Early Exit Result）

不知道准确性也就无法调整早退出口的配置

研究背景

3.频繁且代价高昂的适应性调整

工作负载变化



导致最佳 的EE 配置随时发生频繁变化

最佳 的EE 配置 ： 在不牺牲响应精度的情况下最大限度地降低时延

(1) 大量的 EE 文献都没有提供在服务过程中调整早退出点和退出阈值的策略

(2) 提出的 EE 模型都配备了最大数量的早退出点，调整配置代价高昂

提纲

- 研究背景
- **研究问题**
- 方法设计
- 实验评估
- 工作总结

研究问题

Serving System



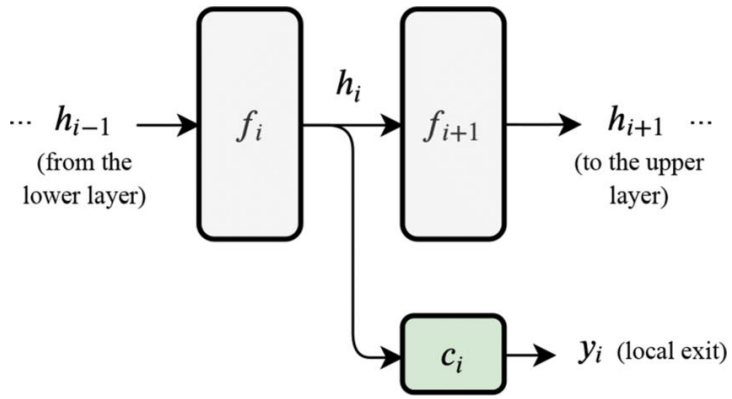
吞吐量与时延的平衡问题

需要解决的问题

1. 如何获取EE模型的准确性?
2. 如何根据结果进行参数调整?
3. 如何保证低时延、高精度、高吞吐?

Serving System





时延 ,吞吐量

满足精度约束

提纲

- 研究背景
- 研究问题
- **方法设计**
- 实验评估
- 工作总结

方法设计

Apparate

Apparate 是一个端到端系统，可自动将早期退出集成到模型中，并在整个推理过程中管理其运行。

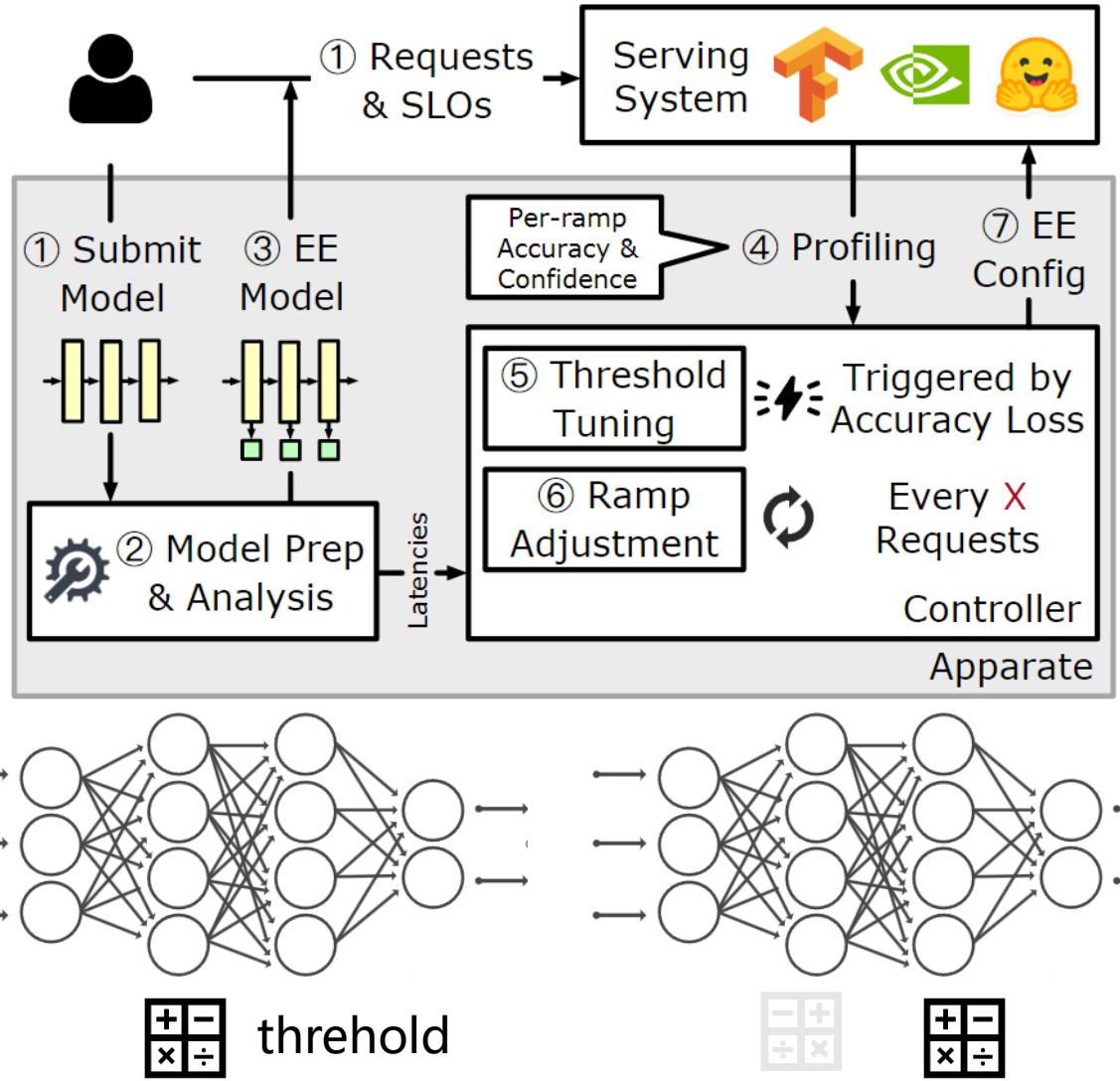
其总体目标是优化每次请求的延迟时间，同时遵守严格的精度限制和吞吐量目标。

EE模型准备阶段 → **管理阶段**

1、分析早退出点(ramps)的放置位置
2、根据引导数据进行训练

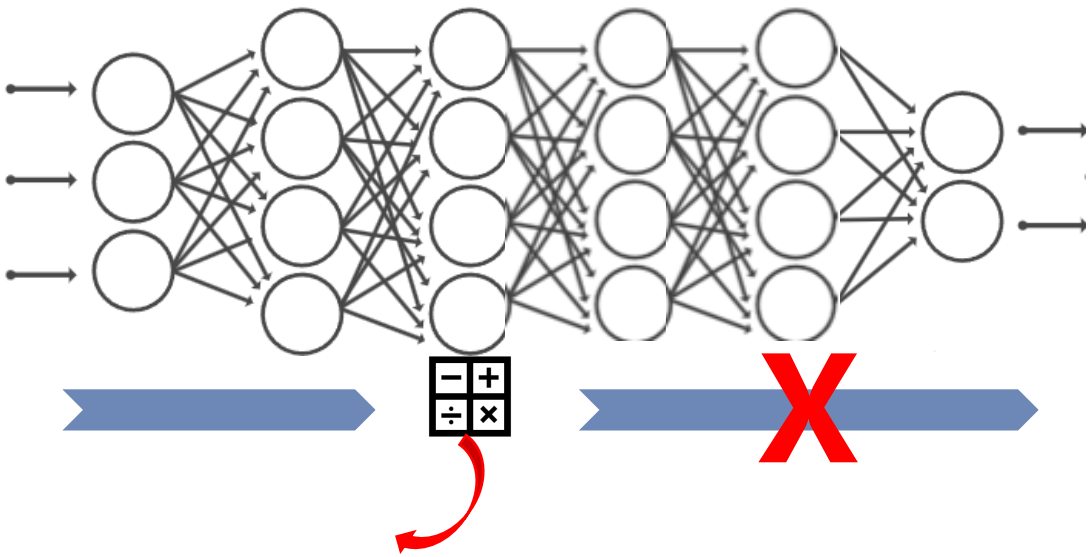
1. 收集实时反馈
2. 调整EE配置

早退出点: Ramps



方法设计

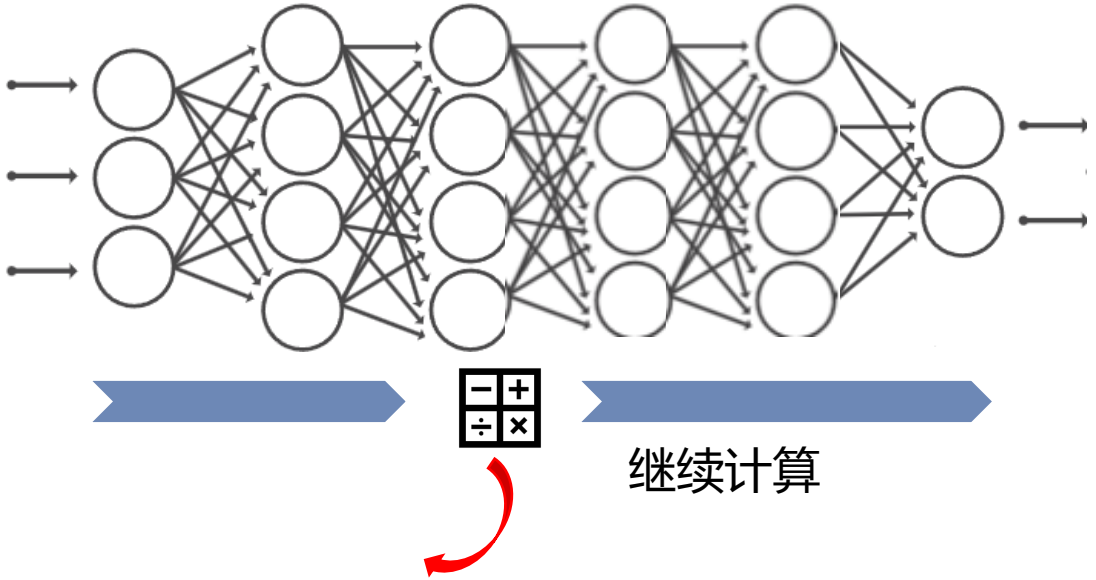
早期退出模型



直接退出

时延 ↑ 计算量 ↑ 获得早退出结果

Apparate



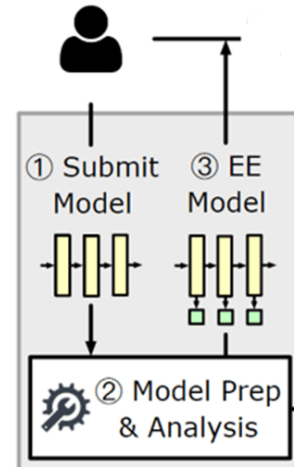
继续计算

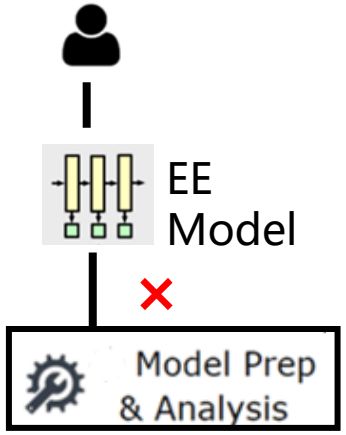
返回结果

时延 ↓ 计算量不变 可以获得全模型结果 √

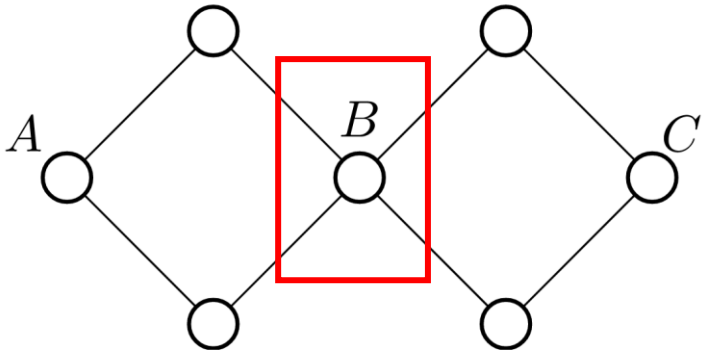
方法设计

1.早期退出模型准备

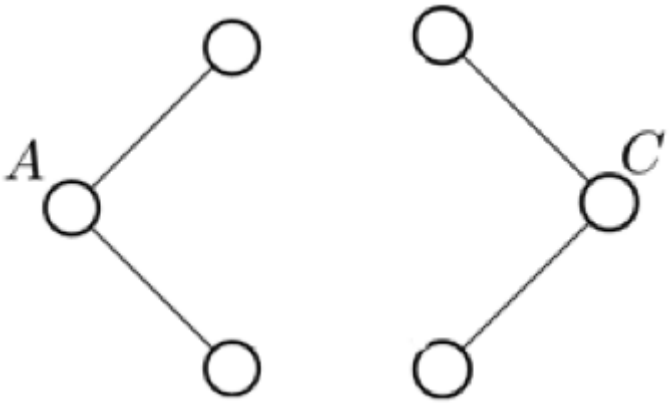




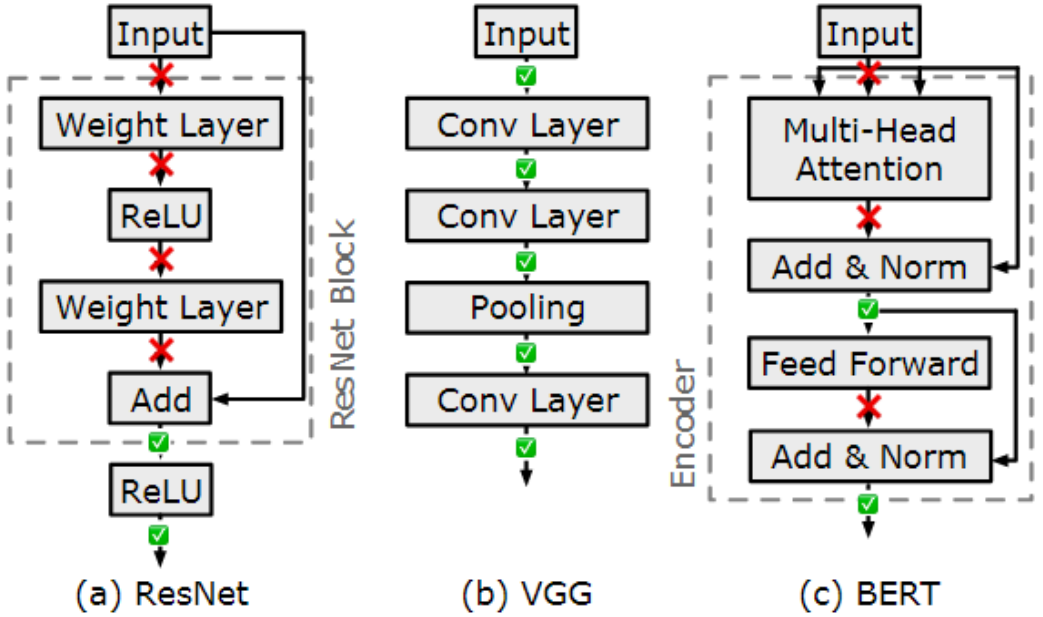
Ramps位置选择 1



选择割点位置



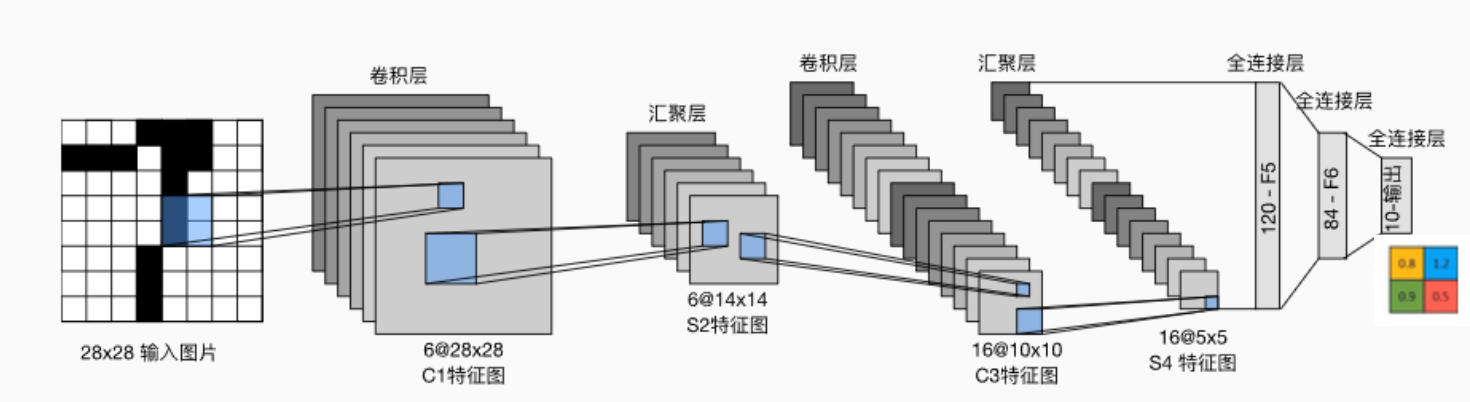
Ramps位置选择 2

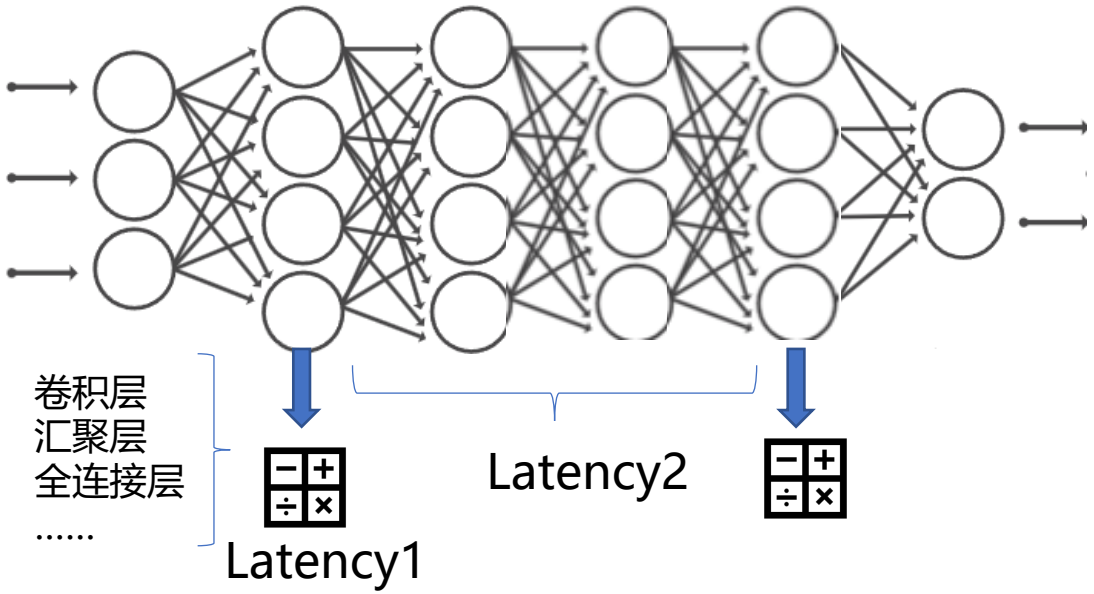


在区块间进行部署，块内不部署

方法设计

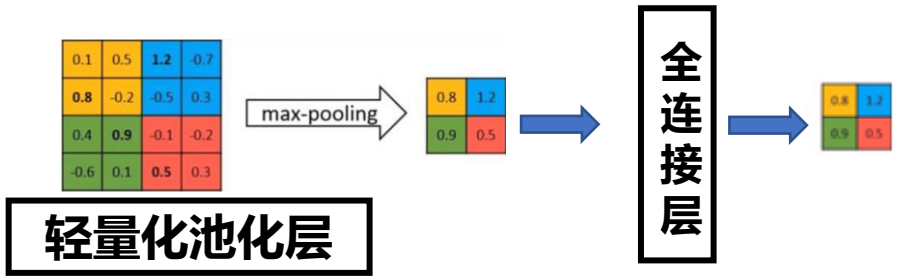
Ramps结构





可以由任意层和计算组成
但是输出格式必须一致

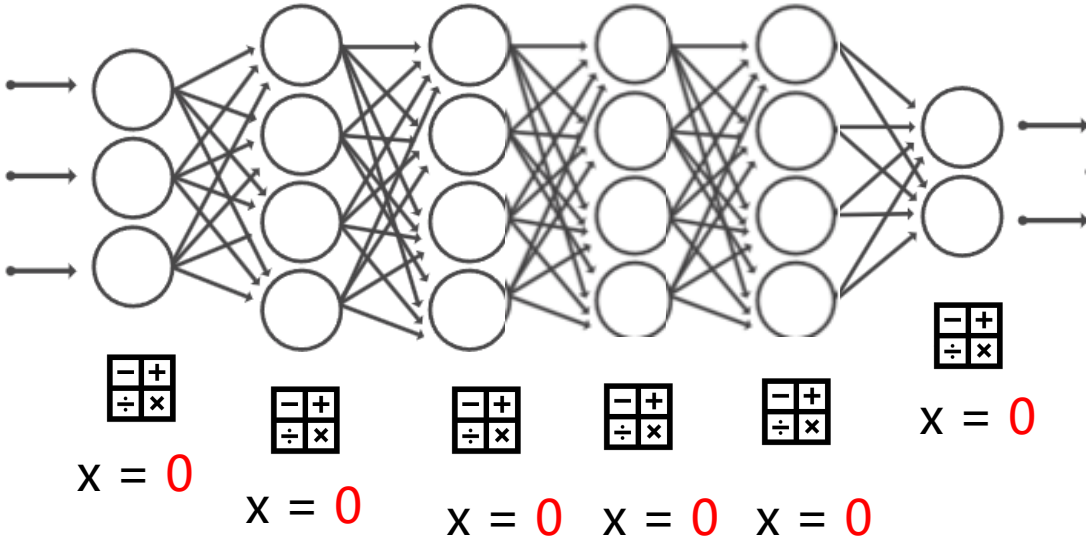
Ramp



Latency1 < Latency2, Ramp才有意义

方法设计

训练Ramp和部署模型



1. 均匀分布Ramp位置

在训练开始时，Apparate会根据允许的最大Ramp数量，将Ramp均匀地分布在模型的不同位置。

2.初始值退出概率设为0

定义x表示在这层 Ramp 提前退出的容易程度。
x的初始值设为0：为了避免误退出

3.禁用提前退出

在训练期间，所有输入会经过所有的Ramp，而不会被提前退出。这种禁用提前退出的策略可以确保每个Ramp在没有上游Ramp影响的情况下独立训练

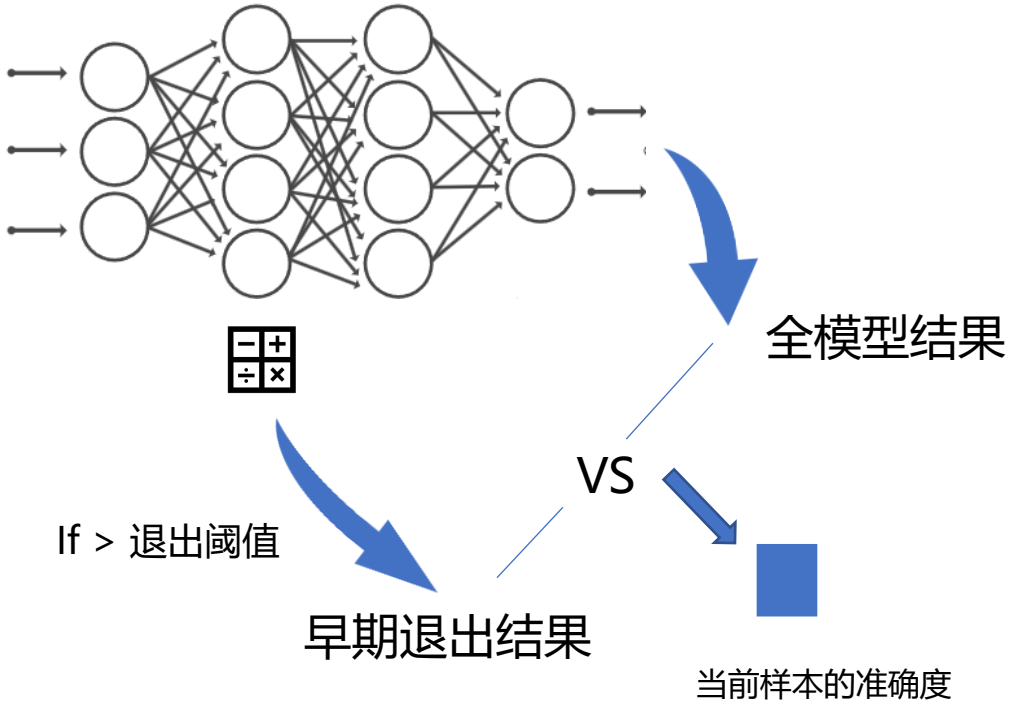
4.数据集分割和训练集使用

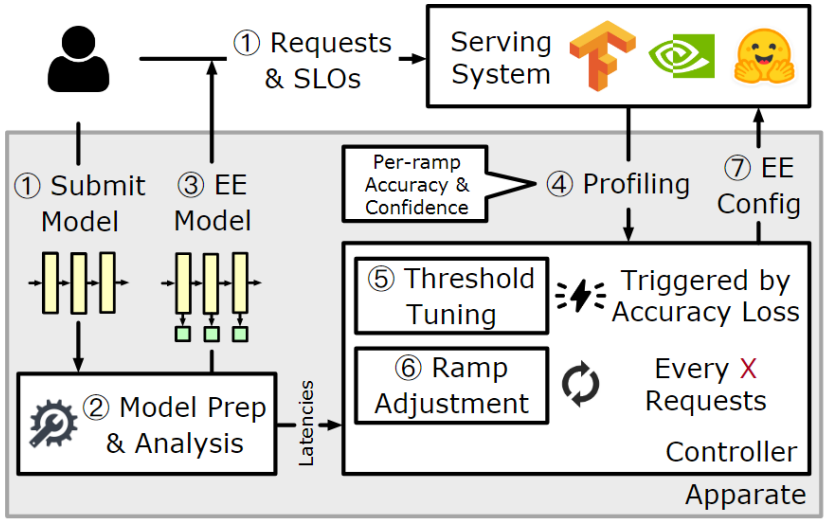
Apparate使用数据集的前10%用于Ramp的初始训练和验证，按1:9的比例划分。这种设置让模型可以快速完成初步训练。

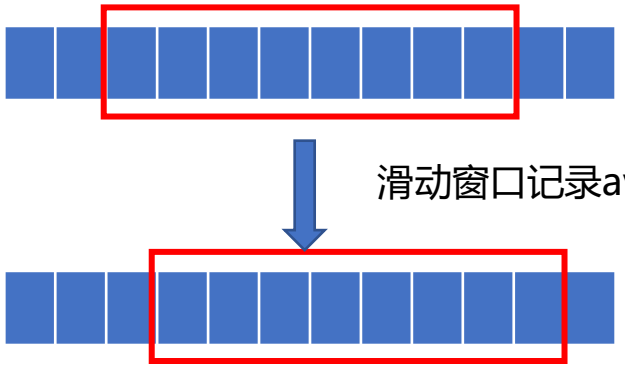
方法设计

2. 精度感知阈值调整

触发调整







如果average_accuracy < threshold，就进行调整

方法设计

贪心搜索

x 表示在这层早退出点提前退出的容易程度 （0 <= x <= 1）

x = 0 ， 即这层不会提前退出

爬山算法

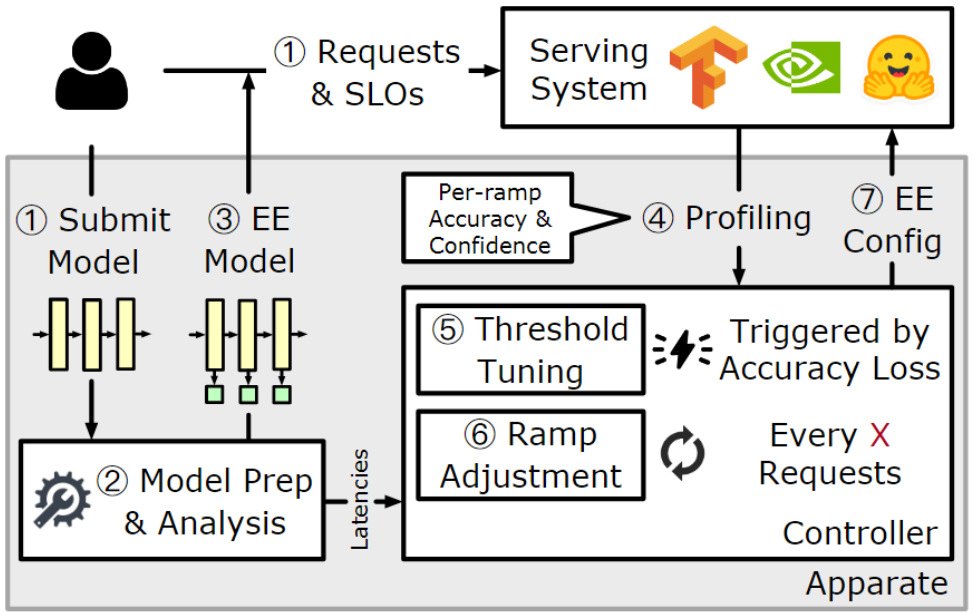
步长 len = 0.1，time(x)表示x下的时延
如果 time(x + len) < 最大时延， len = 0.2、0.4 .....
否则 , len = 0 .1、0.05、0.025.....

1. 每轮评估：每一轮中，Apparate 会尝试增加每个 ramp 的阈值，逐一评估它们的延迟节省和准确性变化。
2. 选择最佳调整：在这一轮中，Apparate 选择效果最好的 ramp 进行实际阈值的更新。
   所谓效果最好：带来最大延迟节省且不会导致准确性违反
3. 下一轮探索：然后进入下一轮，继续对所有 ramp 进行评估，选择下一个可以带来最大延迟节省且符合准确性要求的 ramp 进行调整。

评估时延节省：运行推理任务，记录新配置下所需的时间与基线延迟进行比较
评估准确性变化： 与完整模型进行对比

方法设计

3.调整Ramp



每128个采样点进行一次调整

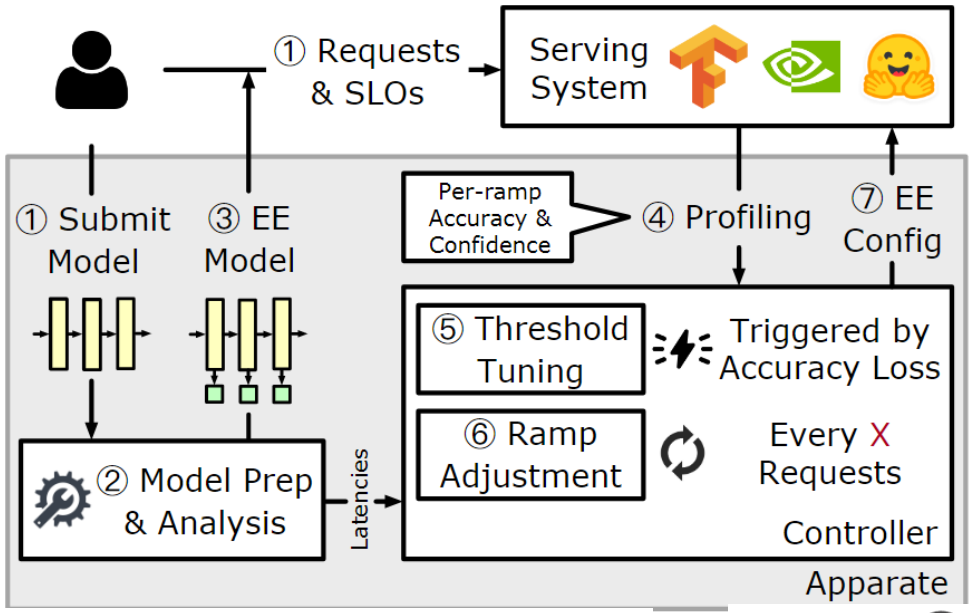
在每一轮迭代中，Apparate的控制器都会为每个激活的Ramp计算一个效用得分F

通过这种计算方式，Apparate可以评估每个Ramp的实际贡献，确保在权衡节省的延迟和增加的延迟开销后，优先选择对整体延迟改善效果最大的Ramp。

Apparate将Ramp的效用F定义为**延迟节省 - 延迟开销**，具体为：

- **延迟节省（savings）**：使用该Ramp的退出功能后，提前退出的输入节省的总延迟。
- **延迟开销（overheads）**：该Ramp对未退出输入增加的总延迟。

方法设计

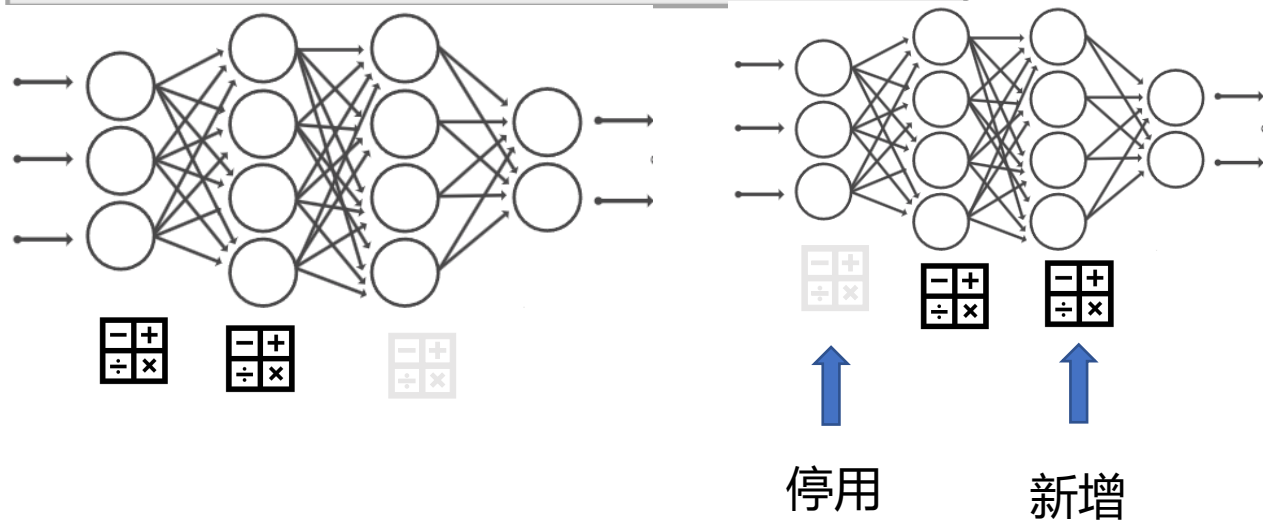


效用值F

if F ＜ 0，进行阈值调整
if F still ＜ 0，停用Ramp

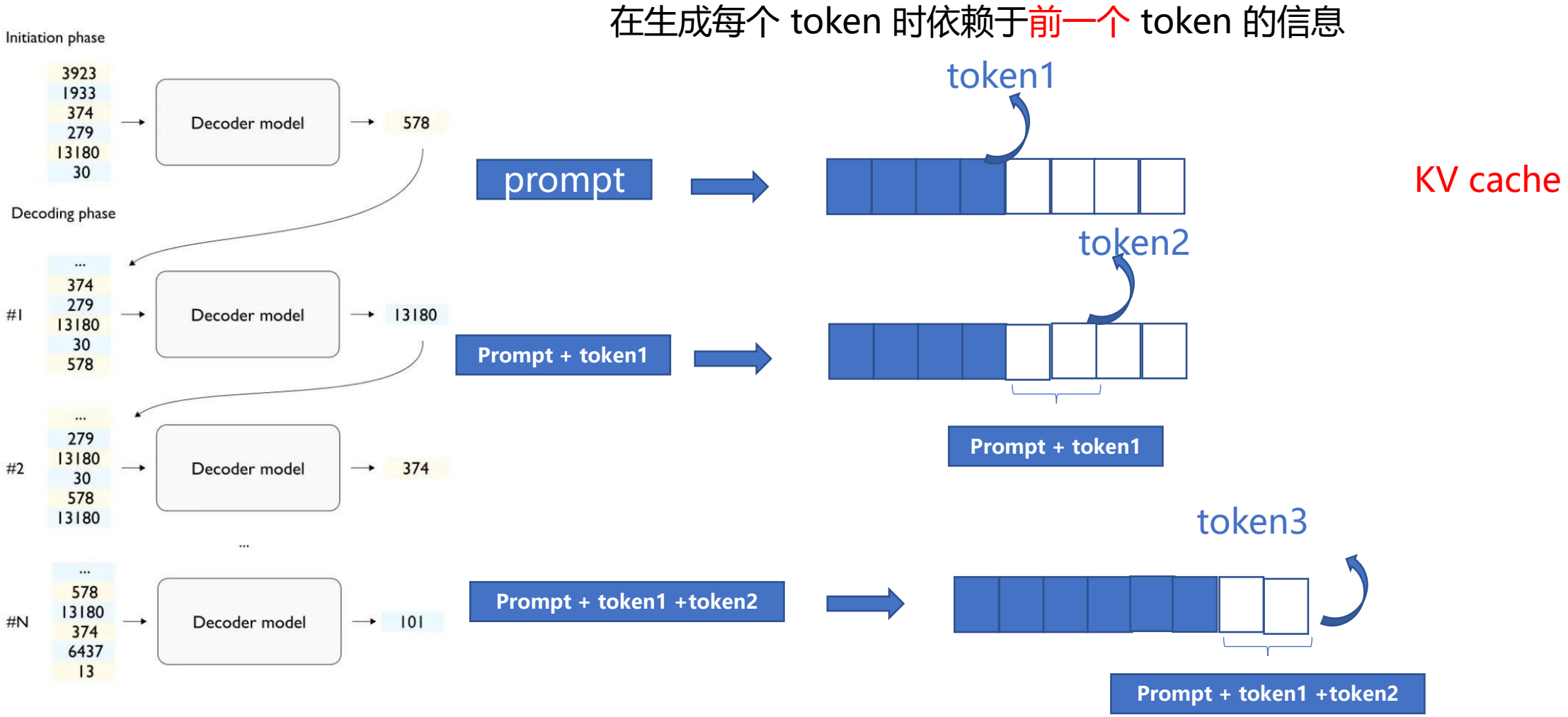
停用ramp后，Apparate会尝试寻找新的Ramp位置，根据效用值F寻找最佳位置。

新添加的Ramp初始时的x设为0，以确保不会误判退出，并将在后续的阈值调整中进行优化。



方法设计

4.支持生成式大模型



提纲

- 研究背景
- 研究问题
- 方法设计
- **实验评估**
- 工作总结

实验评估

模型选择

- CV：ResNet 、VGG
- NLP: BERT-base、 BERT-large、 DistilBERT
- Generative: T5 、 Llama

数据集

- CV: 八段一小时的视频，包含实时目标分类任务（如人、车的识别）
- NLP：Amazon product reviews、IMDB movie reviews
- Generative: CNN/DailyMail 数据集进行文本摘要任务
  SQuAD 数据集进行问答任务

参数配置

- 服务级别目标 (SLOs) ： 10~200ns
- 准确率约束：允许的准确率损失不超过1%
- 延迟预算：不会对模型延迟产生超过2%的影响

硬件配置

- NVIDIA RTX A6000 显卡
- AMD EPYC 7543P 32核 CPU
- 256GB 的 DDR4 内存

服务平台

- TensorFlow-Serving
- Clockwork

生成型任务的推理引擎

HuggingFace Pipelines 推理引擎

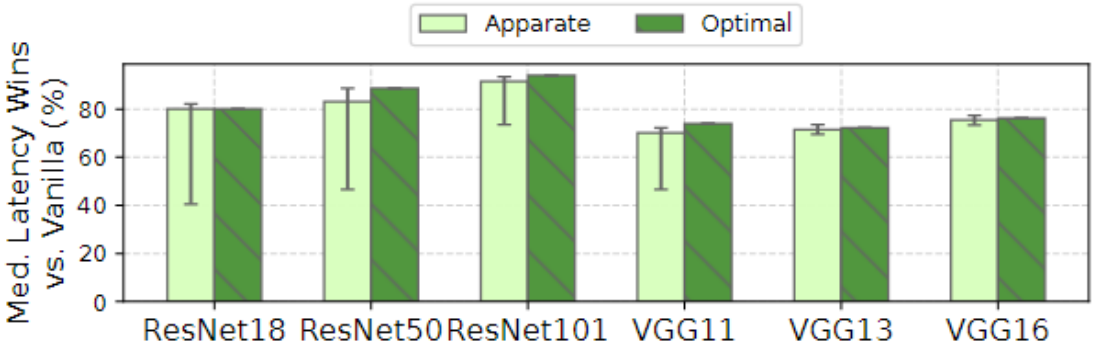
评估指标

分类任务：
- 准确率
- 响应延迟

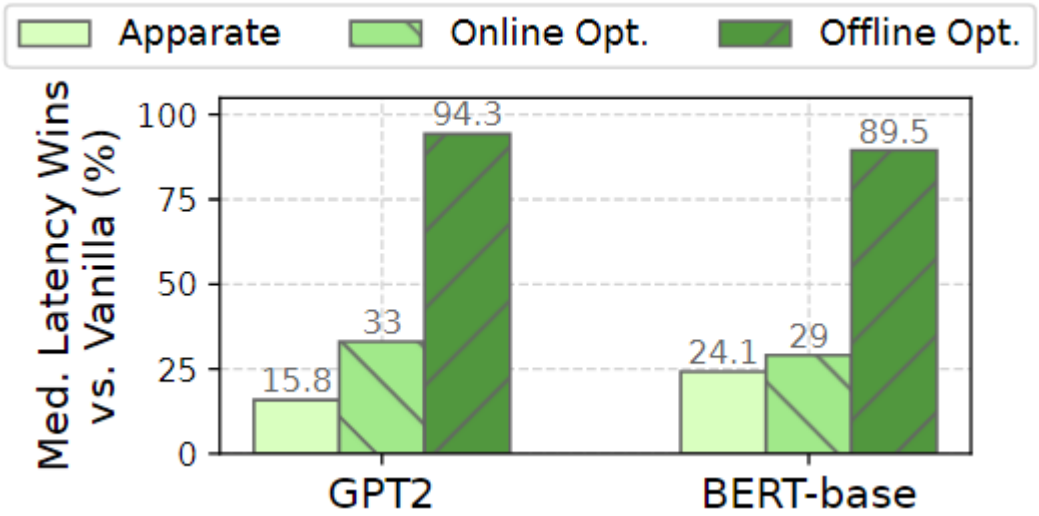
生成式模型：
- 准确性：ROUGE-L 分数、F1 分数
- 延迟：TPT

实验评估

延迟优化效果



CV　　Apparate、Optimal　　VS　Vanilla(原始模型)



NLP　　Apparate、Online Opt.、Offline Opt　　VS　Vanilla(原始模型)

Apparate 系统能够在保持准确率的前提下，显著降低 CV、NLP 任务的延迟

实验评估

与现有的EE策略对比

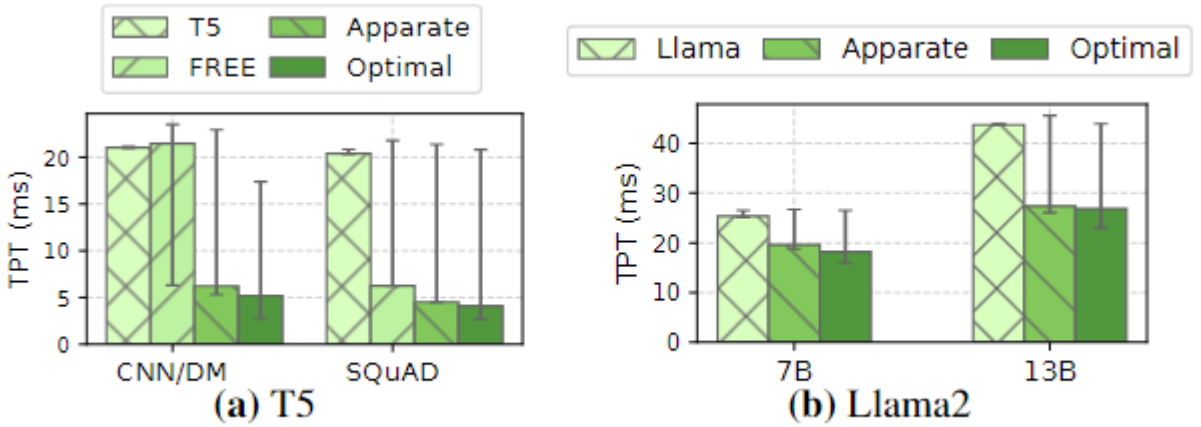
Classification.

	Avg Acc	Median Wins	P95 Wins
Apparate (ResNet50)	99.0–99.2%	46.6–88.6%	-1.6–0.0%
BranchyNet	85.8–99.8%	-11.0–88.3%	-11.0%
BranchyNet+	76.1–99.9%	-11.0–88.3%	-11.0%
BranchyNet-opt	99.0–99.7%	-11.0–74.5%	-11.0%
Apparate (BERT-base)	99.1–99.3%	13.7–14.7%	2.1–3.0%
DeeBERT	91.7–97.1%	13.2–36.1%	-1.3–6.4%
DeeBERT+	82.2–90.3%	31.7–36.1%	5.9–6.4%
DeeBERT-opt	99.0%	9.8–36.1%	-1.4–6.4%

BranchyNet 和 DeeBERT的早退出点是固定的

Apparate 可以动态调整退出点位置和阈值

Generative.

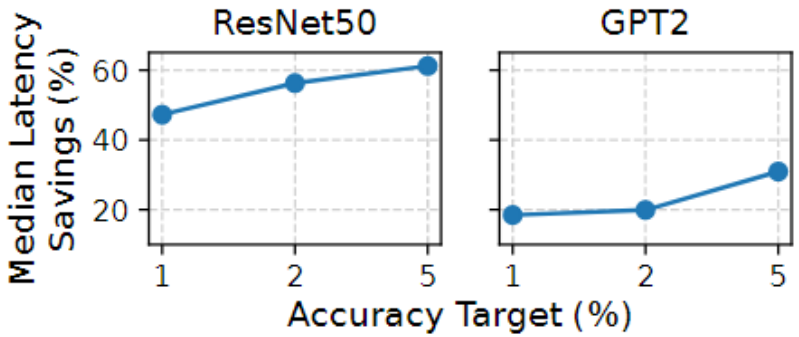


在生成任务中显著降低了每个token的中位生成时间

实验评估

微观性能测试

准确度目标



服务平台差异

System\Workload	ResNet50	GPT2
Clockwork	(20.2, 37.8)	(689.2, 779.4)
TF-Serve	(24.5, 37.8)	(709.3, 793.1)

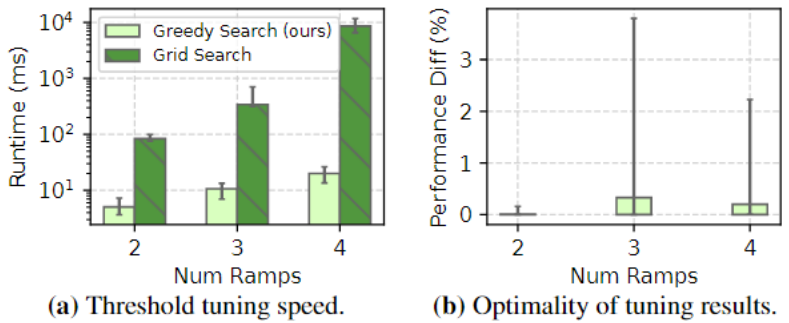
时延降低ms

早退出点数量

Ramp Budget	ResNet50	GPT2
2%	48.9%	18.5%
5%	49.6%	22.2%
10%	50.4%	24.9%

退出节点数量　　时延降低百分大小

搜索算法



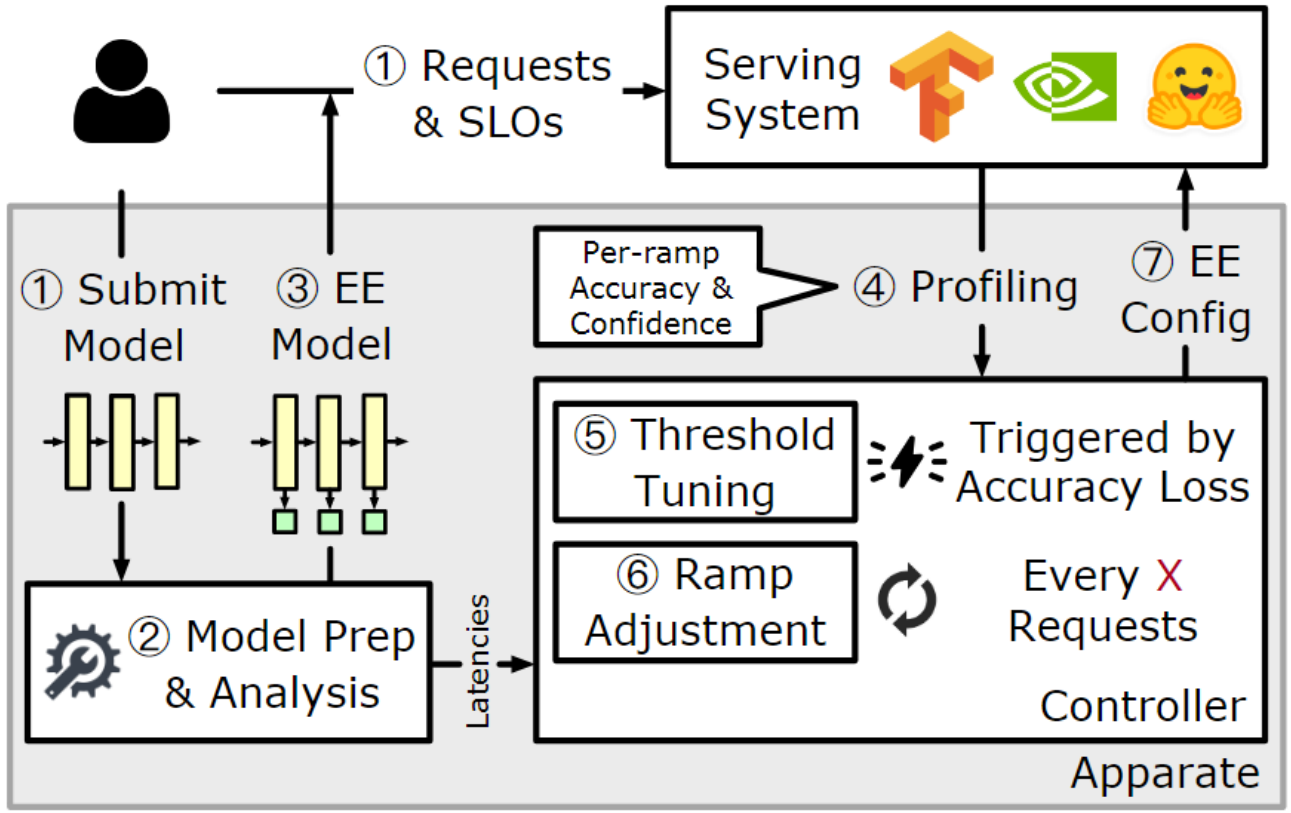
(a) Threshold tuning speed.　(b) Optimality of tuning results.

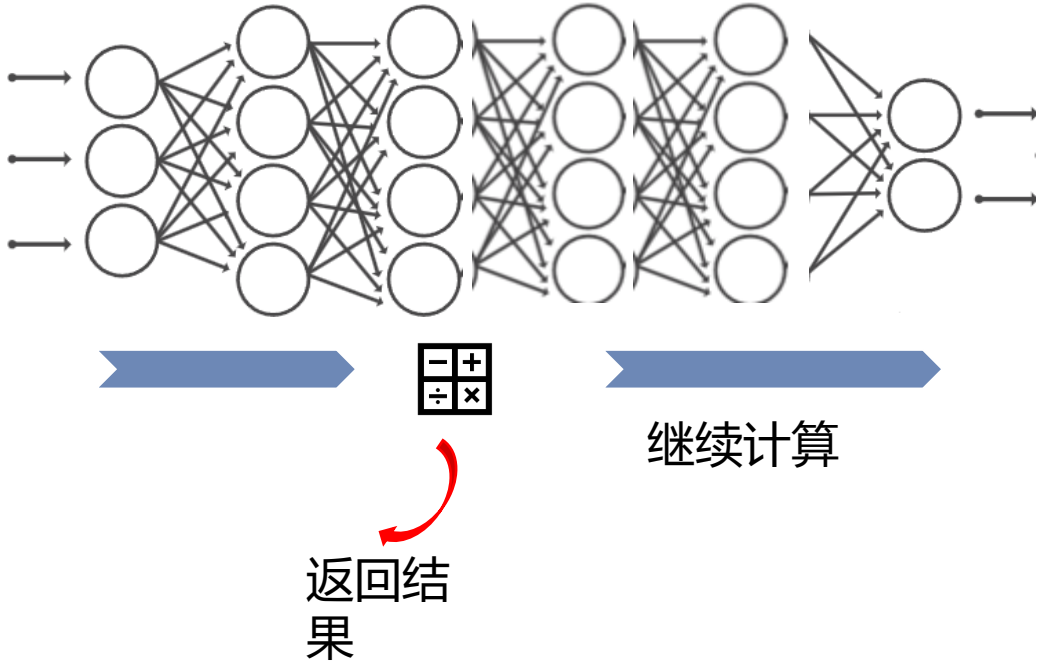
提纲

- 研究背景
- 研究问题
- 方法设计
- 实验评估
- **工作总结**

工作总结

总结


① Requests & SLOs
Serving System
① Submit Model
③ EE Model
Per-ramp Accuracy & Confidence
④ Profiling
⑦ EE Config
⑤ Threshold Tuning
Triggered by Accuracy Loss
⑥ Ramp Adjustment
Every X Requests
② Model Prep & Analysis
Latencies
Controller
Apparate


继续计算
返回结果

Q&A

2024年11月10日

冯敏远

东南大学智慧物联网实验室