東南大學
SOUTHEAST UNIVERSITY

INFERCEPT: Efficient Intercept Support for Augmented Large Language Model Inference

Reyna Abhyankar * 1 Zijian He * 1 Vikranth Srivatsa 1 Hao Zhang 1 Yiying Zhang 1

UC San Diego

Reporter: Zikang Chen

ICML
International Conference
On Machine Learning

Outline

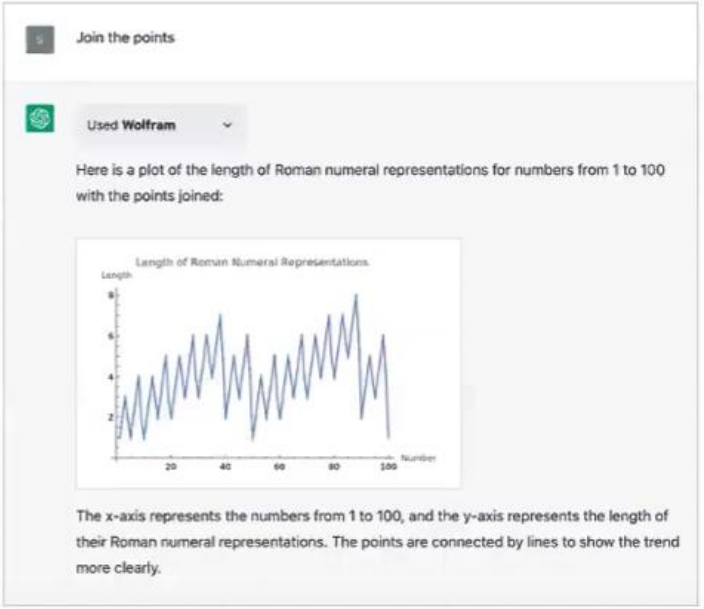
- *Background and Motivation*
- *Design*
- *Evaluation*
- *Conclusion*

Outline

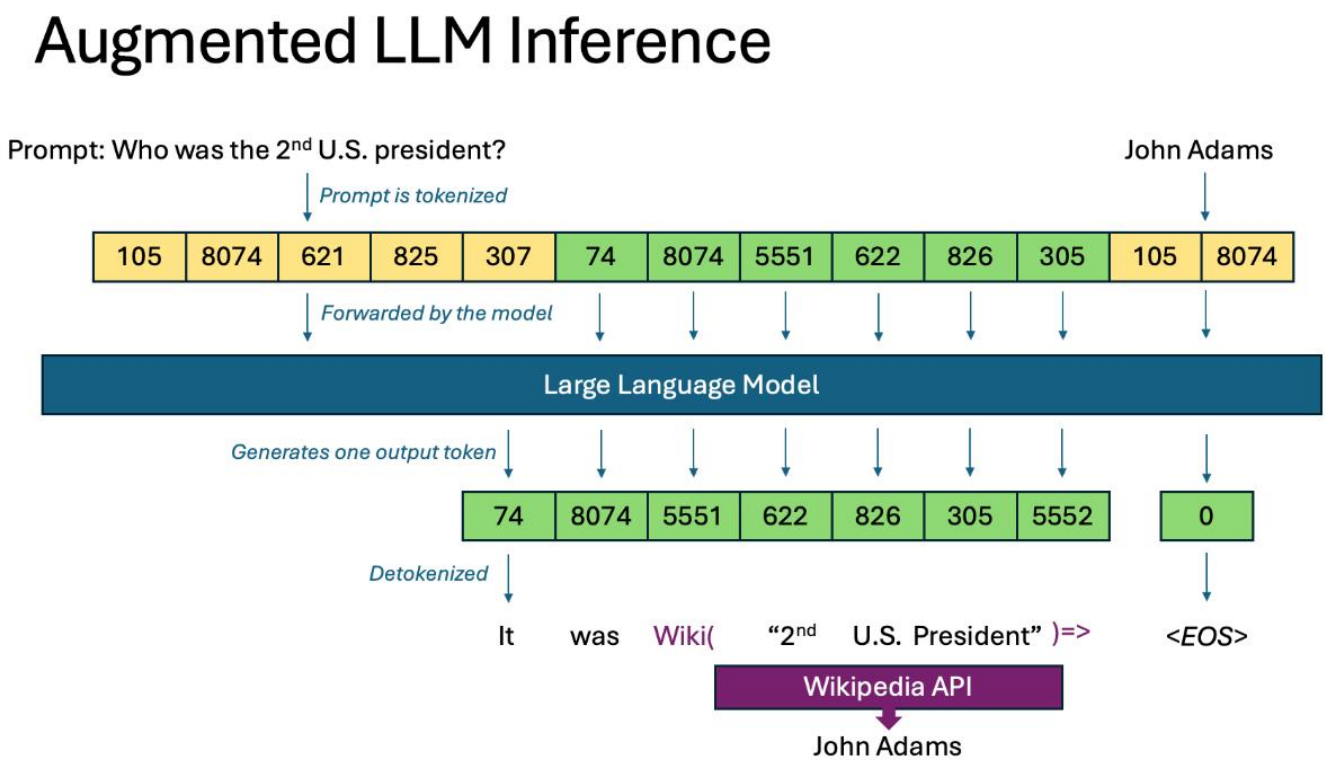
- **Background and Motivation**
- Design
- Evaluation
- Conclusion

Background： Augmenting LLMs for complex tasks

- Traditional LLMs generate text
- Now, LLMs can interact with other entities
    - •Tools like calculator, calendar, wiki search
    - •Other models like text-to-speech, image gen
    - • Can act as an agent navigating virtual env
    - • Chat bot



Background：Augmenting LLM Inference



问题：当调用外部工具或等待人类/环境响应时，正常的解码阶段会暂停，只有在拦截完成后才能恢复。

Background: Studying augmentations

- Execution time varies wildly
  - Tools can be very fast
  - Other models slower
- Invocation frequency also differs
  - Calculator is just a few calls
  - Virtual environment is 25+ calls

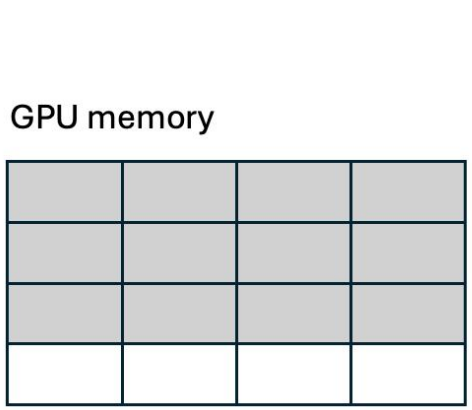
Type	Int Time (sec)	Num Interceptions	Context Len
Math	(9e-5, 6e-5)	(3.75, 1.3)	(1422, 738)
QA	(0.69, 0.17)	(2.52, 1.73)	(1846, 428)
VE	(0.09, 0.014)	(28.18, 15.2)	(2185, 115)
Chatbot	(28.6,15.6)*	(4.45,1.96)	(753, 703)
Image	(20.03, 7.8)†	(6.91,3.93)*	(1247,792)
TTS	(17.24,7.6)†	(6.91,3.93)*	(1251,792)

Motivation： How state-of-art systems handle intercepts

- LLM interception is treated as request completion
  - All context(each token's KV cache) is discarded
  - Recompute KV for all tokens in context when interception finishes
  - 37% latency spent on recomputation
  - 27% GPU memory is wasted

Motivation： How state-of-art systems handle intercepts

Strategy 1: Discard + Recompute

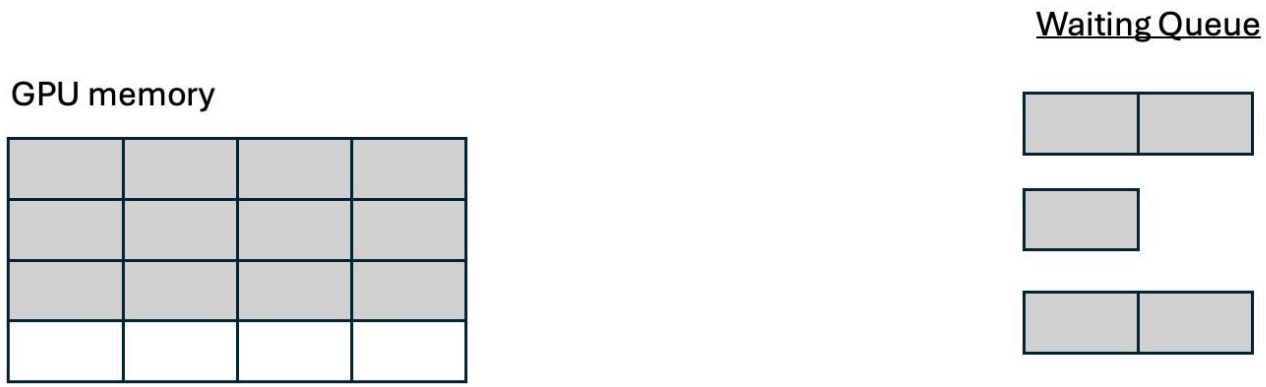


Running requests

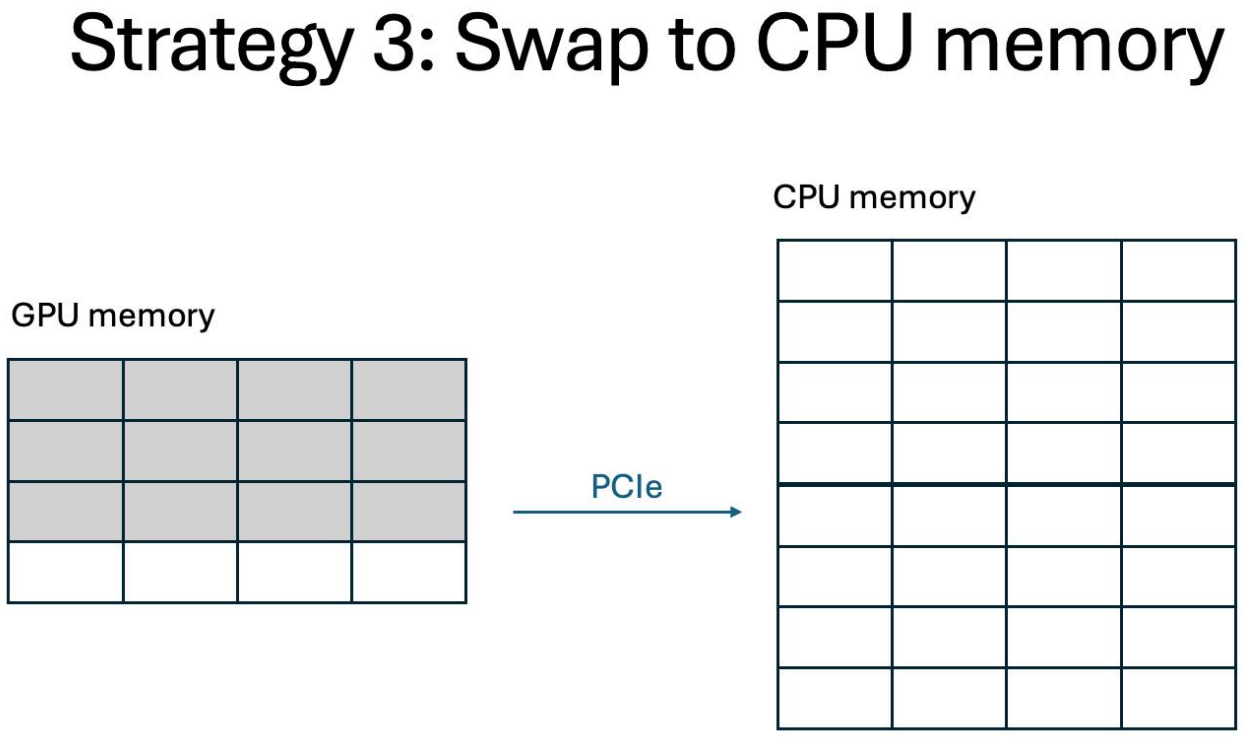
Typical iteration time: 40 ms

Motivation： How state-of-art systems handle intercepts

Strategy 2: Preserve



Motivation： How state-of-art systems handle intercepts

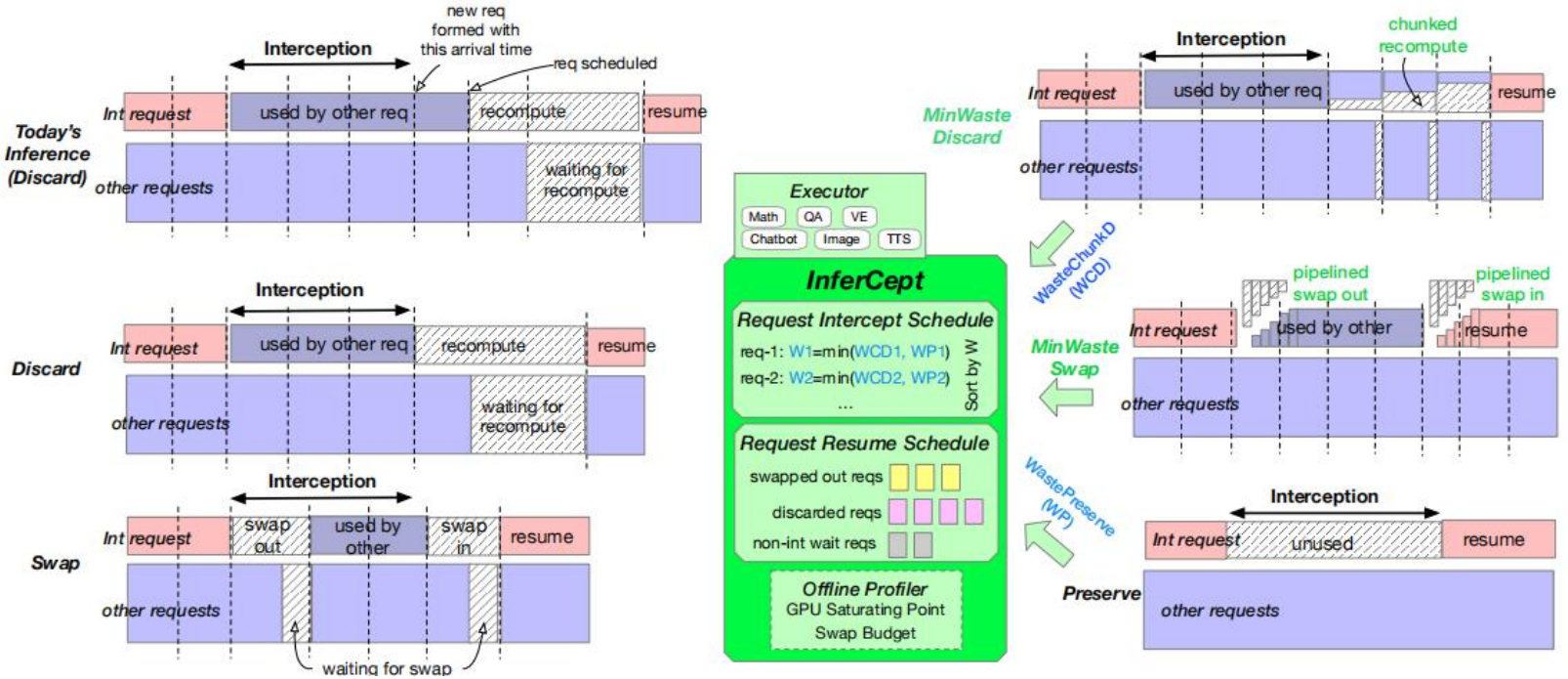


Outline

- Background and Motivation
- **Design**
- Evaluation
- Conclusion

Design: INFERCEPT

1.硬件感知的交换与重算优化（Swap Pipelining & Recomputation Chunking）。
2.系统级浪费模型驱动的动态策略选择（Inter-Request Action Decision）。
3.基于历史数据的拦截时间预测（Interception Duration Estimation）。



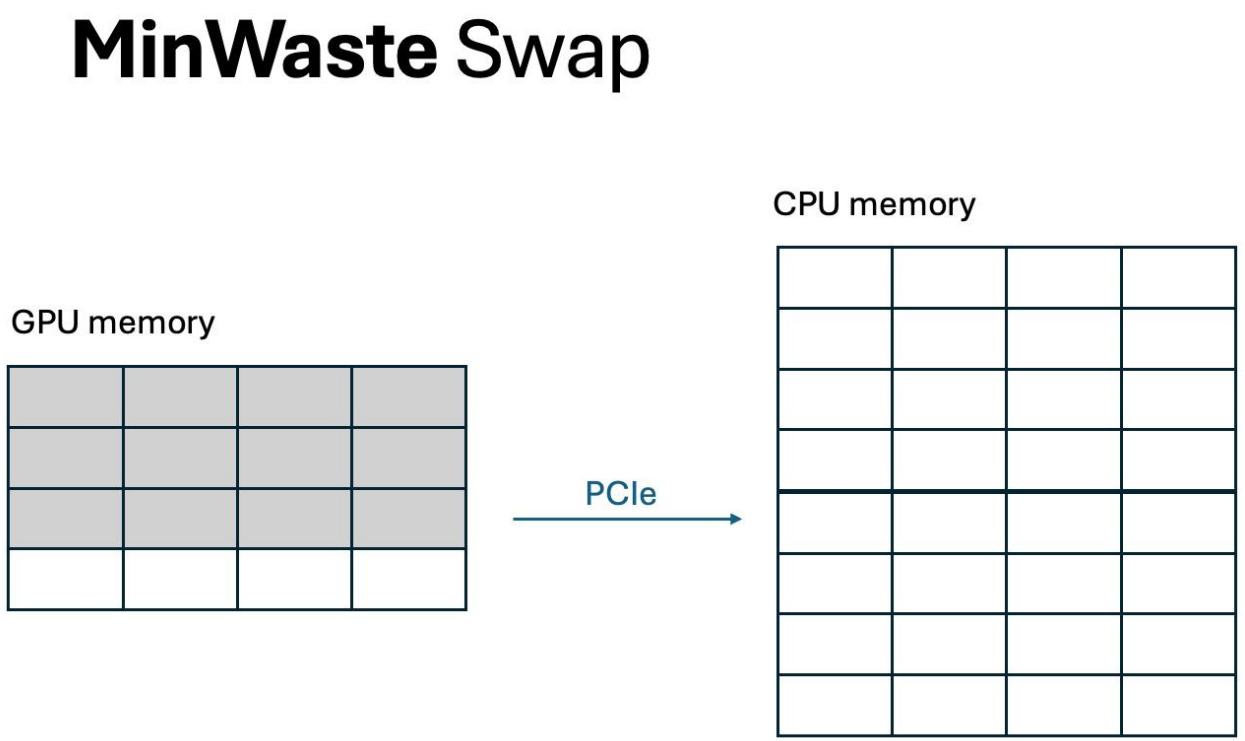
Design: Pipelining and Chunking

目标：减少交换操作的内存浪费，提升交换与计算的并行性。

分块（Chunking）：
将待交换的上下文（KV 缓存）分割为固定大小的块（chunks），每次迭代仅交换部分块，确保交换时间能被当前迭代的计算任务隐藏。
通过离线分析获得前向传播时间 T_{fwd} 和交换时间 T_{swap} 在第i次迭代下计算交换预算
交换预算为可以用计算时间掩藏交换时延的token数目 N_{i}：$T_{\mathrm{swap}}(N_{\mathrm{i}}) = T_{fwd}(B_i)$

流水线交换（Pipelining）：
传统的Swap策略在拦截时同步将上下文（如KV缓存）从GPU交换到CPU，这会阻塞计算任务，延长迭代时间。
InferCept将交换过程按模型层（layer）分阶段流水线化。例如，当在为交换第i+2层的数据准备时，第i+1层的数据正在传输，而第i层的数据已释放并用于正常计算。这种并行化减少了交换与计算之间的等待时间。

Design: Pipelining and Chunking



Design: Recomputation Chunking

目标：优化因丢弃上下文导致的重新计算开销，提高GPU核心利用率。

实现方式：

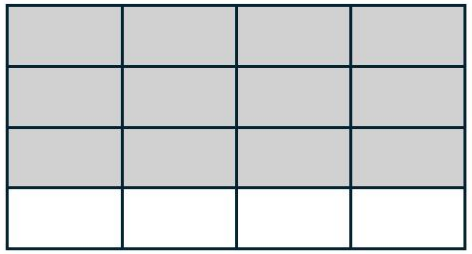
•丢弃策略（Discard）需要重新计算全部上下文，但一次性重计算长上下文会显著增加迭代时间。

•将上下文划分为多个小块，每轮迭代仅重计算一个块。块大小根据GPU的“饱和点”（即单次迭代能处理的token数）动态调整，确保重计算不超出GPU处理能力。饱和点由离线获得。其为给定模型架构下，所有 GPU 核心可以并行处理的查询token的最大数量。超过这个数量会增加迭代时间，但不会提高服务吞吐量。

Design: Recomputation Chunking

MinWaste Discard

GPU memory



Running requests

Typical iteration time: 40 ms

Design: Inter-Request Action Decision

目标：动态选择最优策略（Discard/Preserve/Swap）以最小化系统级浪费。

•浪费度量模型：

定义系统级浪费为未使用内存与时间的乘积：

$$Waste = \sum (Unused\_Memory_i \times Time_i)$$

•**Unused Memory**：因策略选择而暂时无法利用的 GPU 内存。
•**Time**：策略执行的持续时间（如拦截调用的延迟）。

Design: Inter-Request Action Decision

Discard的系统级浪费：

对于第i个请求，其在第j出发生拦截，共有C_i^j个tokens，每个token的KV cache占M大小内存，重新计算需要花费$T_{fwd}(C_i^j)$的时间。

对于recomputation其系统级浪费为：

$$Waste = T_{fwd}(C_i^j) \times C_i^j \times M$$

其次，重新计算上下文会增加迭代时间，因为现在一个迭代需要在完成其他运行请求的模型前向传播的同时，完成所有的重新计算。在额外的迭代时间里，其他运行请求所占用的内存就被浪费了。因此整体使用Discard策略所带来的系统级浪费定义为

$$Waste = T_{fwd}(C_i^j) \times C_i^j \times M + T_{fwd}(C_i^j) \times C_{other} \times M$$

Design: Inter-Request Action Decision

Preserve的系统级浪费：

对于第i个请求，其在第j出发生拦截，拦截持续时间为 T_{INT}^{j} 其系统级浪费为：

$$Waste = T_{INT}^{j} \times C_i \times M$$

Swap的系统级浪费：

其主要的内存浪费在swap过程之中定义为 $T_{swap}(C_i^j) \times C_i^j \times M$ 别的请求等待swap完成且不做任何事造成浪费为 $T_{swap}(C_i^j) \times C_{other} \times M$

$$Waste = T_{swap}(C_i^j) \times C_i^j \times M + T_{swap}(C_i^j) \times C_{other} \times M$$

Design: Inter-Request Action Decision

优化Discard后的ChunkDiscard策略系统级浪费：

对于第i个请求，其在第j出发生拦截，共有C_i^j个tokens将其拆为n块，每次迭代仅重计算一个块,对gpu内存的占用逐步释放，第一块重计算完成后，可以用于计算其他请求或者后续的块。

Recompute浪费变为：

$$Waste = \frac{T_{fwd}(C_i^j) \times C_i^j \times M}{2}$$

其他请求被浪费的时间变为了

$$Waste = \mathrm{n} \times T_{fwd}(\frac{C_i^j}{n}) \times C_{other} \times M$$

整体的系统级浪费小于原始Discard策略

Design: Inter-Request Action Decision

动态决策:

对每个请求i其拦截j计算对于使用Preserve和ChunkDiscard哪个更小并选为使用的策略:

$$Decision = \arg\min(Waste_{ChunkDiscard}, Waste_{Preserve})$$

•按浪费量降序排序，优先为浪费最大的请求分配交换带宽，剩余请求根据最小浪费原则选择保留或丢弃基于之前选择的策略。

Design: Inter-Request Action Decision

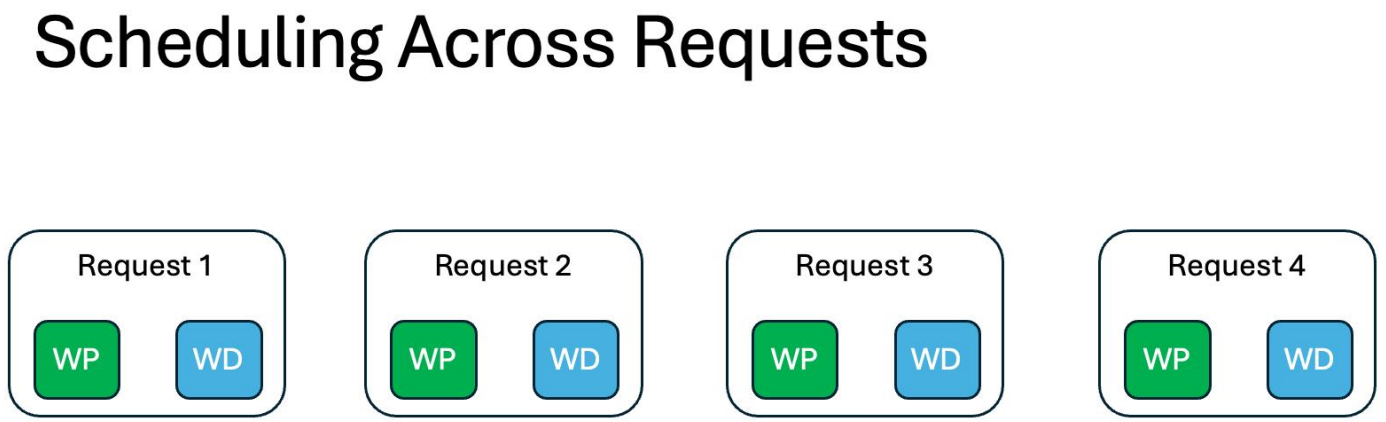
1. 三大队列

- **Running Queue**：GPU 上正在执行的请求。
- **Swap Queue**：拦截后换出到 CPU 的请求，需按带宽预算换回。
- **Waiting Queue**：
  - 新请求
  - 需重计算的丢弃请求（分块处理）
  - 因资源不足被暂停的请求

2. 调度策略

- **FCFS（先到先服务）**：所有队列按请求到达时间排序，保证公平性。

Design: Inter-Request Action Decision



Design: Interception Duration Estimation

目标：预测拦截操作的持续时间，优化调度和资源分配。

对于Preserve的内存浪费需要拦截持续时间。对于那些持续时间变化很大或未进行离线分析的拦截，提出了一种动态估算方法

$$T_{INT} = t_{now} - t_{call}$$

- t_{call}：拦截触发时间。
- t_{now}：当前时间，每轮迭代更新。

Outline

- Background and Motivation
- Design
- **Evaluation**
- Conclusion

Evaluation

- **Models & Baselines**

Model: GPT-J-6B, Vicuna-13B, Llama3-70B

Baselines: vLLM(Discard), Preserve, Swap, ImprovedDiscard

- **数据集**

Math (GSM8K-XL dataset)

QA (Multihop QA Wikipedia dataset)

Virtual Environment (VE) (ALFWorld dataset)

Chatbot (ShareGPT dataset)

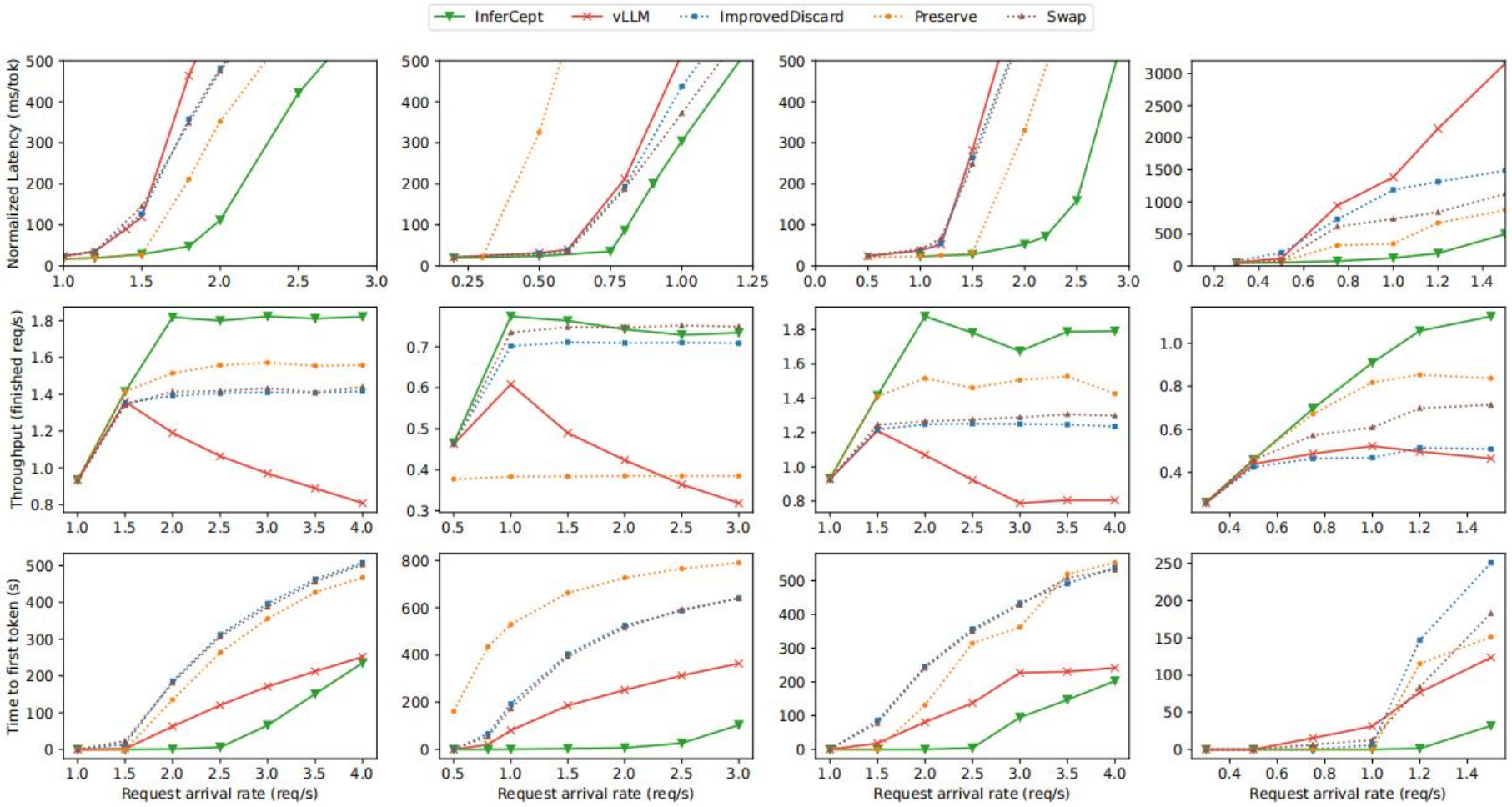
Image Generation (Stable Diffusion calls)

Text-to-Speech (TTS) (Bark TTS model)

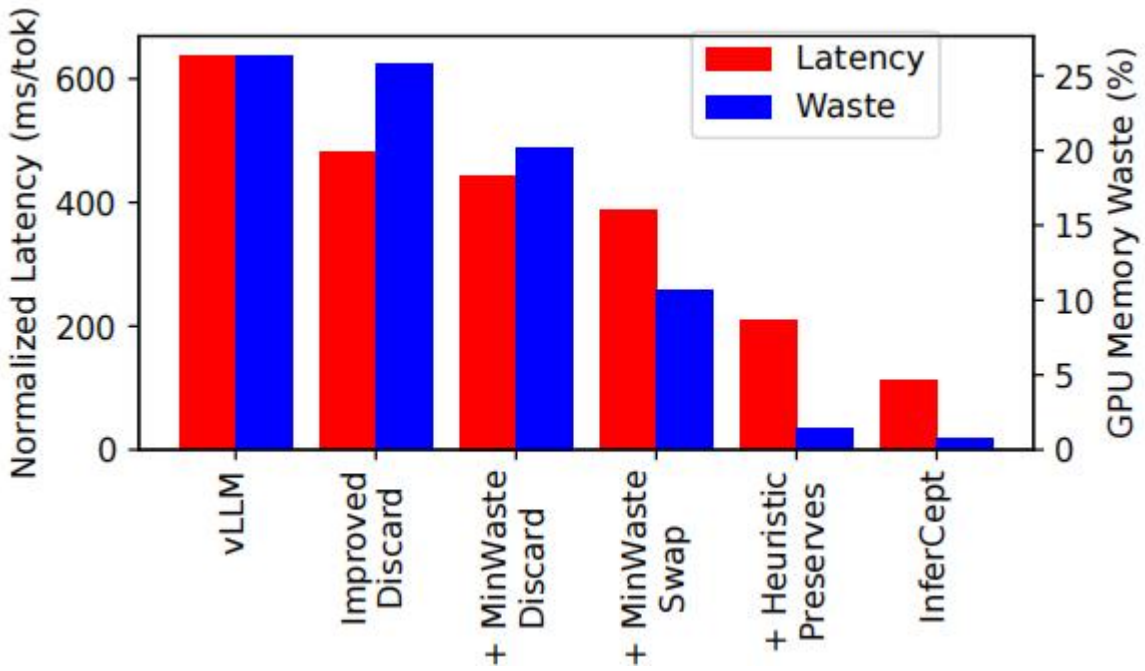
- **设备**

A100

Evaluation


InferCept
vLLM
ImprovedDiscard
Preserve
Swap
Normalized Latency (ms/tok)
Throughput (finished req/s)
Time to first token (s)
Request arrival rate (req/s)
(a) 6B model, 1 GPU
(b) 13B model, 1 GPU
(c) 13B model, 2 GPUs
(d) 70B model, 4 GPUs

Evaluation



消融实验

Outline

- Background and Motivation
- Design
- Evaluation
- **Conclusion**

Conclusion

这个paper有什么问题?

动态调度的优化不足：调度策略依赖静态启发式规则，未结合机器学习预测拦截时间或资源需求，可能无法适应动态变化的高负载场景。

系统复杂性与硬件依赖：交换流水线和分块重计算增加了实现复杂度，且依赖NVIDIA GPU的特定优化，对其他硬件（如AMD GPU、TPU）的兼容性未验证。

基于这个paper还能做什么?

1.优化其调度排队策略，改用强化学习或者别的算法，使其在预测拦截操作的持续时间上更准确。

2.探索其在别的硬件上实现可能性

3.调度策略是FCFS可否进一步优化

4.其结合在多agent场景下的混合优化

Thanks for listening

Reporter: Zikang Chen