隐私云计算视角下的大模型推理优化

管佳乐

2024-10-25

Outline

1 大模型推理 2
1.1 服务系统 3
1.2 阶段 5
1.3 优化的挑战 7
2 隐私云计算 8
2.1 背景 9
2.2 要求 10
3 无状态计算 11
3.1 定义 12
3.2 数据管理 13
3.3 传输 19
3.4 推测执行 21
3.5 状态残留 23
4 不可针对性 24
4.1 定义 25
4.2 提高针对性 27
5 总结 28
5.1 大模型推理优化的发展方向 29
5.2 相关的论文 30
5.3 工业界的推理系统 32
5.4 大模型推理与可信计算 33
6 附录 35

Outline

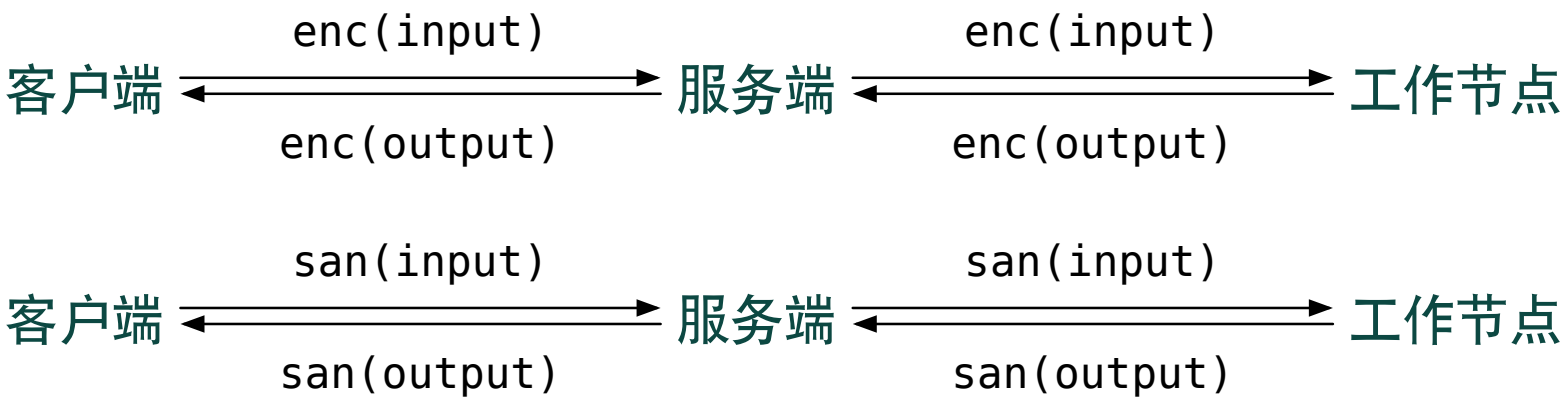
1 大模型推理 2
1.1 服务系统 3
1.2 阶段 5
1.3 优化的挑战 7
2 隐私云计算 8
2.1 背景 9
2.2 要求 10
3 无状态计算 11
3.1 定义 12
3.2 数据管理 13
3.3 传输 19
3.4 推测执行 21
3.5 状态残留 23
4 不可针对性 24
4.1 定义 25
4.2 提高针对性 27
5 总结 28
5.1 大模型推理优化的发展方向 29
5.2 相关的论文 30
5.3 工业界的推理系统 32
5.4 大模型推理与可信计算 33
6 附录 35

1.1 服务系统

传统的服务器架构

1.1 服务系统

传统的服务器架构



1.1 服务系统

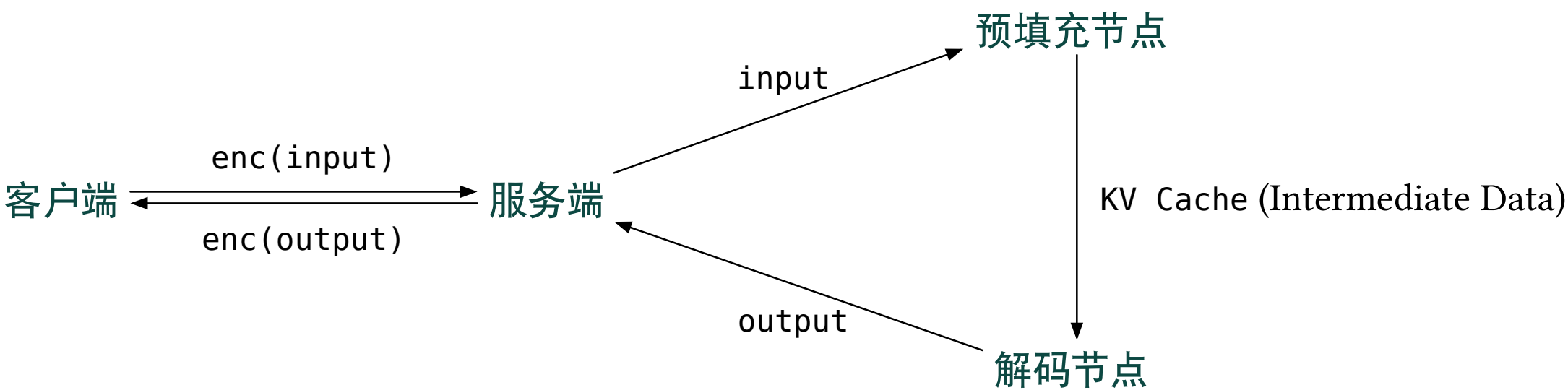
传统的服务器架构

大模型的服务器架构

1.1 服务系统

1.1 服务系统

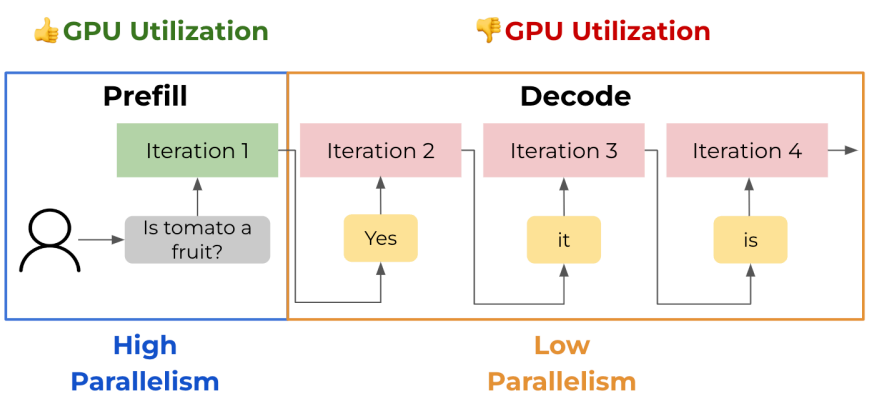

enc(input)
客户端
服务端
input
工作节点
enc(output)
output


预填充节点
input
enc(input)
客户端
服务端
KV Cache (Intermediate Data)
enc(output)
output
解码节点

1.2 阶段

预填充: Processing the input

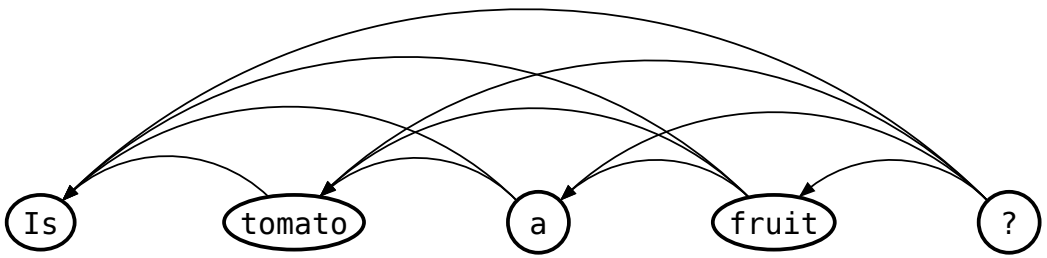
在预填充阶段，LLM 处理输入的 token 以计算中间状态 (KV Cache)，这些状态用于生成“第一个”新 token。每个 token 都依赖于之前的所有 token，但由于输入的完整内容是已知的，从宏观来看，这是一个 Matrix-Matrix 的操作，且高度并行化。这一过程有效地使 GPU 达到最大利用率。

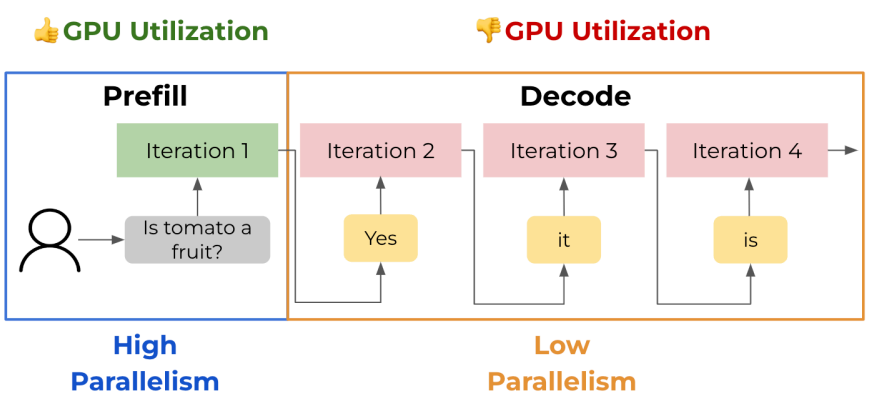


1.2 阶段

预填充: Processing the input

在预填充阶段，LLM 处理输入的 token 以计算中间状态 (KV Cache)，这些状态用于生成“第一个”新 token。每个 token 都依赖于之前的所有 token，但由于输入的完整内容是已知的，从宏观来看，这是一个 Matrix-Matrix 的操作，且高度并行化。这一过程有效地使 GPU 达到最大利用率。

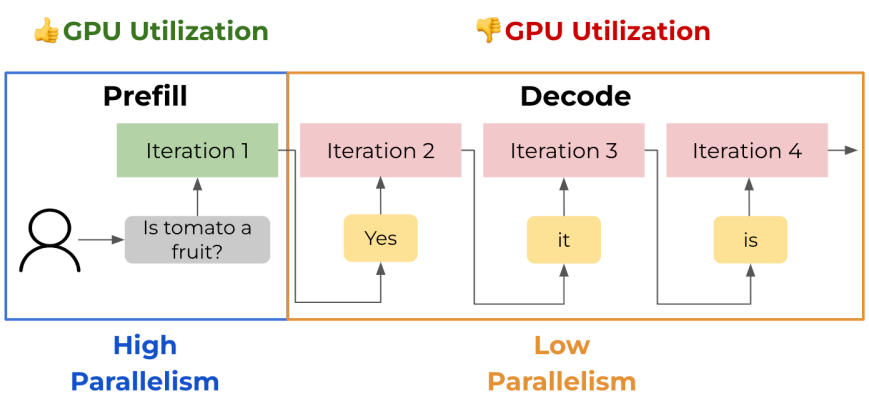


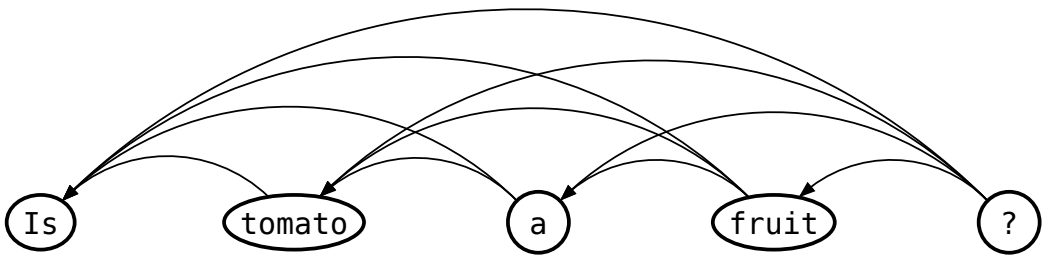


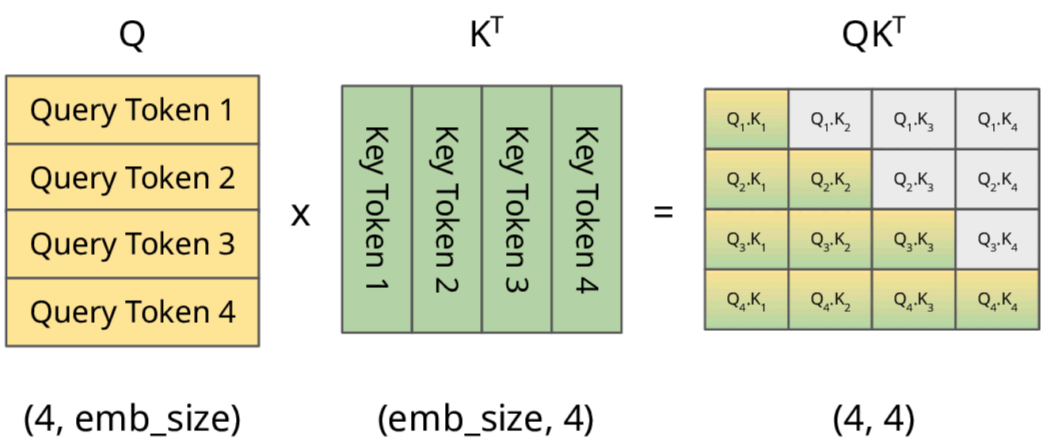
1.2 阶段

预填充: Processing the input

在预填充阶段，LLM 处理输入的 token 以计算中间状态 (KV Cache)，这些状态用于生成“第一个”新 token。每个 token 都依赖于之前的所有 token，但由于输入的完整内容是已知的，从宏观来看，这是一个 Matrix-Matrix 的操作，且高度并行化。这一过程有效地使 GPU 达到最大利用率。





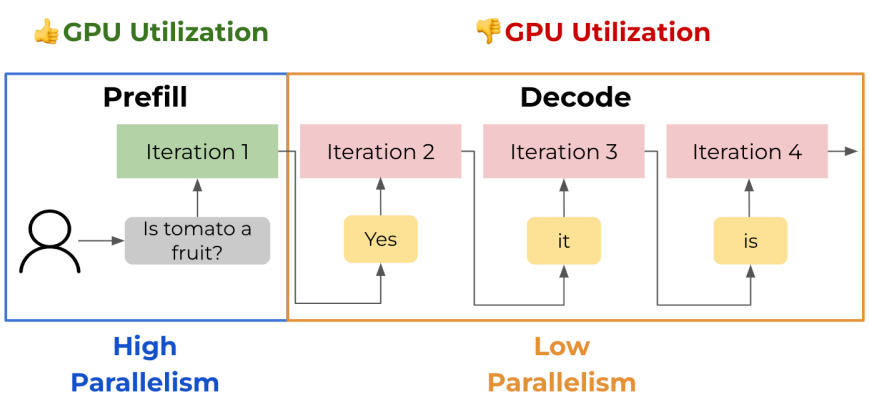


,

1.2 阶段

解码: Generating the output

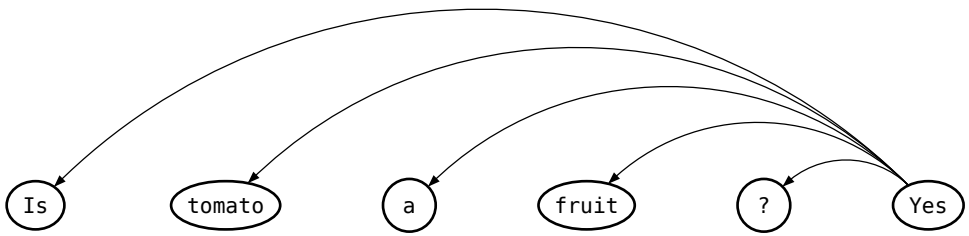
在解码阶段，LLM 以自回归的方式逐个生成输出标记，直到满足停止条件。每一个连续的输出标记都需要了解之前所有迭代的输出状态（KV Cache）。这一过程类似于 Matrix-Vector 操作，相较于预填充阶段，GPU 计算能力未被充分利用。延迟主要受数据（权重、键、值、激活）从内存传输到 GPU 的速度影响，而不是计算的速度。换句话说，这是一个受限于内存的操作。

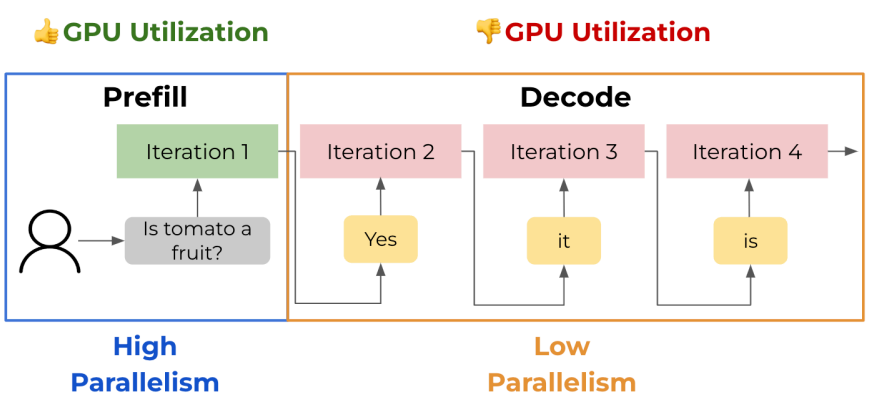


1.2 阶段

解码: Generating the output

在解码阶段，LLM 以自回归的方式逐个生成输出标记，直到满足停止条件。每一个连续的输出标记都需要了解之前所有迭代的输出状态（KV Cache）。这一过程类似于 Matrix-Vector 操作，相较于预填充阶段，GPU 计算能力未被充分利用。延迟主要受数据（权重、键、值、激活）从内存传输到 GPU 的速度影响，而不是计算的速度。换句话说，这是一个受限于内存的操作。

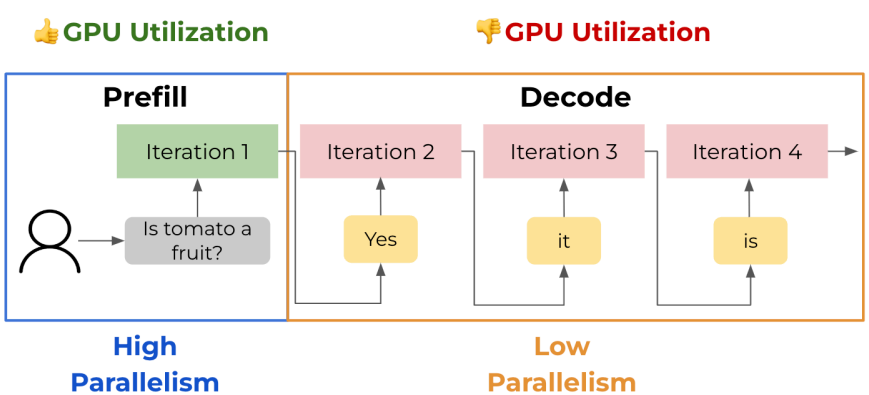


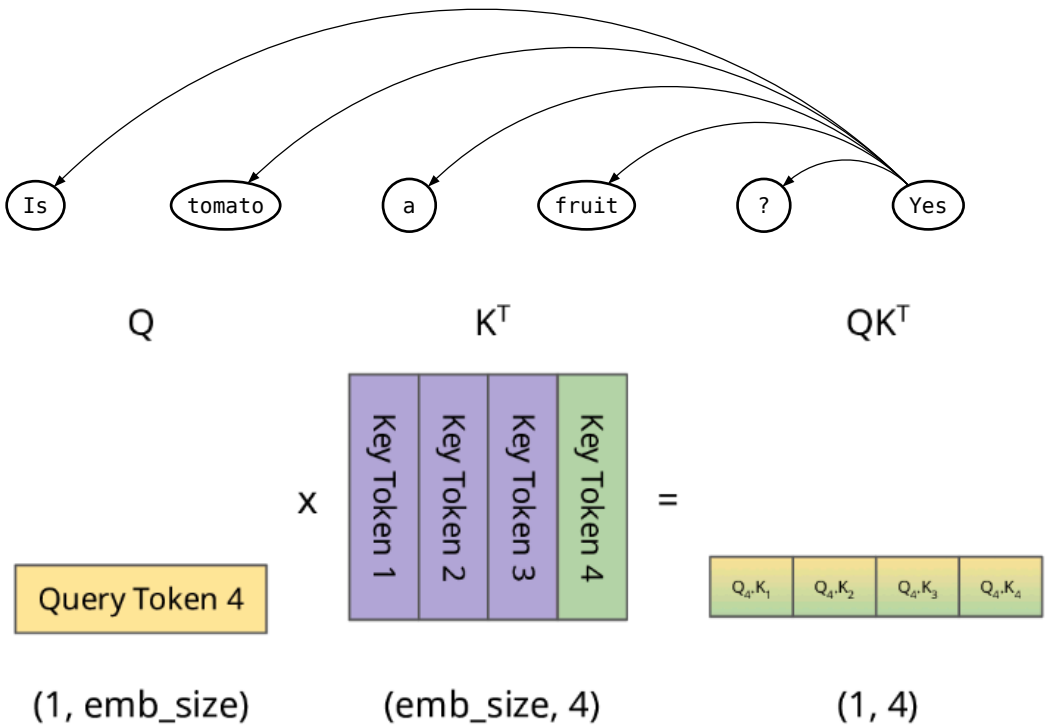


1.2 阶段

解码: Generating the output

在解码阶段，LLM 以自回归的方式逐个生成输出标记，直到满足停止条件。每一个连续的输出标记都需要了解之前所有迭代的输出状态（KV Cache）。这一过程类似于 Matrix-Vector 操作，相较于预填充阶段，GPU 计算能力未被充分利用。延迟主要受数据（权重、键、值、激活）从内存传输到 GPU 的速度影响，而不是计算的速度。换句话说，这是一个受限于内存的操作。





,

1.3 优化的挑战

任务的异质性

通用性和应用的多样性导致推理请求的异构性，表现为输入长度、输出长度、期望延迟等方面的差异。

比如在代码生成任务中，输入长度可能是几十行代码，而在问答任务中，输入长度可能是几个句子。这种异构性使得难以设计一个通用的优化策略，需要根据具体的任务特性进行优化。

执行的不可预测性

在输出结束之前，无法事先知道将生成多少个 token。因此，请求的执行时间和资源需求都是不可预测的。这种不确定性使得难以为每个请求分配合适的资源，需要动态调整资源分配策略。

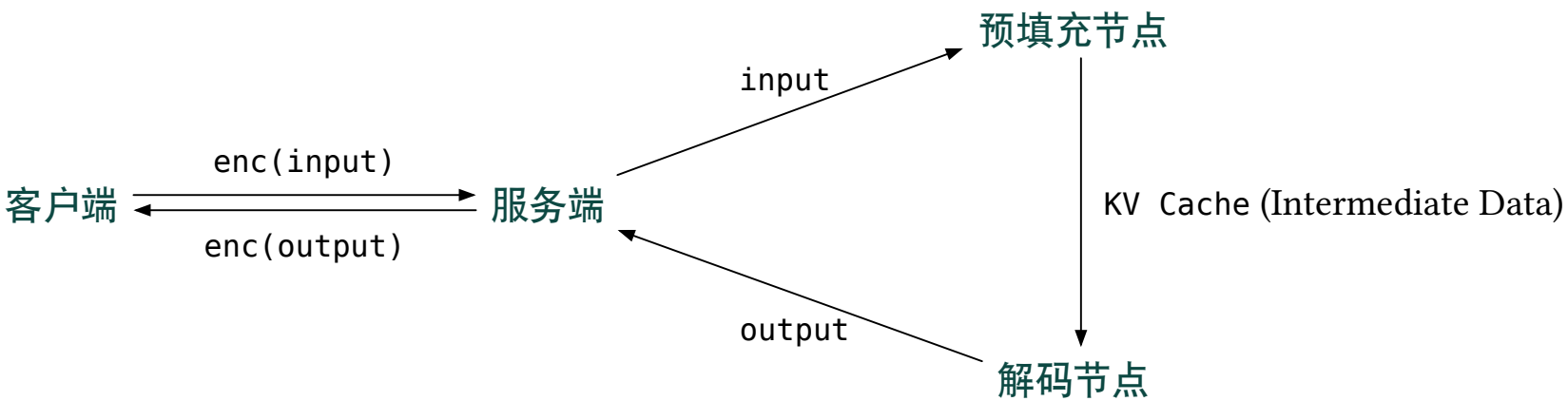
环境的动态性

系统必须能够同时支持多个用户的请求，这要求在资源调度、隔离和优先级控制方面有很强的弹性。推理环境经常变化，模型更新、负载变化等动态因素要求系统具备良好的扩展性和适应性。

Outline

1 大模型推理 2
1.1 服务系统 3
1.2 阶段 5
1.3 优化的挑战 7
2 隐私云计算 8
2.1 背景 9
2.2 要求 10
3 无状态计算 11
3.1 定义 12
3.2 数据管理 13
3.3 传输 19
3.4 推测执行 21
3.5 状态残留 23
4 不可针对性 24
4.1 定义 25
4.2 提高针对性 27
5 总结 28
5.1 大模型推理优化的发展方向 29
5.2 相关的论文 30
5.3 工业界的推理系统 32
5.4 大模型推理与可信计算 33
6 附录 35

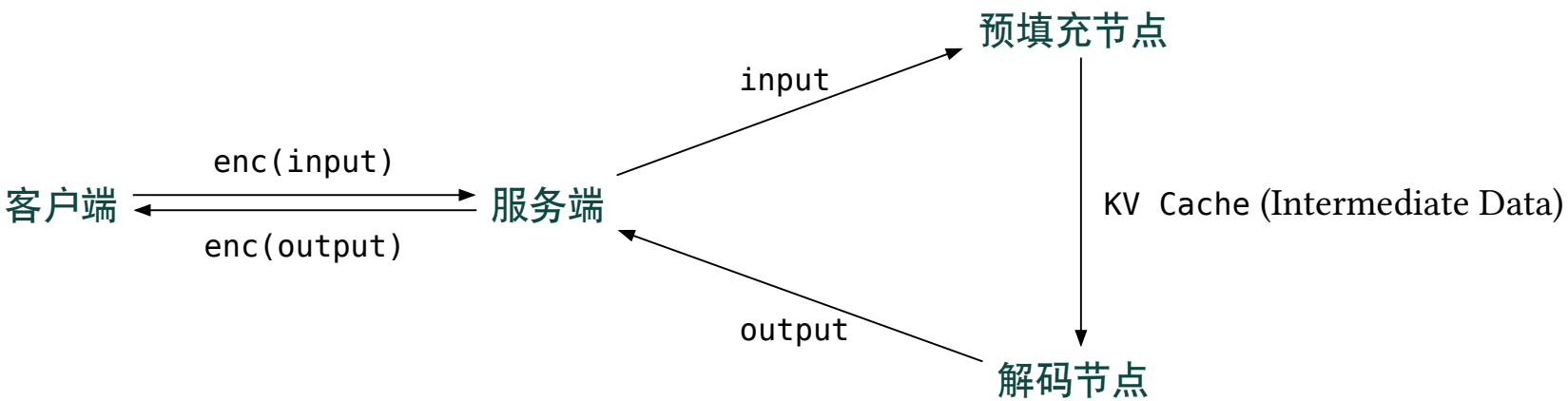
2.1 背景



挑战:

- 很难在实际应用中大规模使用端到端的加密
- 由于应用和场景的多样性，不能期待所有的数据都是匿名化的
-
-

2.1 背景



挑战:

- 很难在实际应用中大规模使用端到端的加密
- 由于应用和场景的多样性，不能期待所有的数据都是匿名化的
- 在工作节点的处理过程中，可能会产生一些中间状态，这些状态可能包含用户的敏感信息
- 部分节点的攻陷可能会导致大量用户数据的泄露

2.2 要求

类别	要求	可能的威胁	Apple 提出的一些方案
技术	无状态计算 Stateless computation	数据处理的痕迹 Example: Logging, debugging	(目的) Only use user data to perform requested operations (时间) Delete the data after fulfilling the request (范围) Not available to even Apple staff
	不可针对性 Non-targetability	定向攻击，定向引导请求 Example: Steer request to compromised nodes	(硬件) Strengthened supply chain (调度) Requests cannot be user/content-specific routed (加密匿名) OHTTP Relay, RSA Blind Signature (访问控制) No system-wide encryption
生态	可强制执行的保证 Enforceable guarantees	外部组件引入的威胁 Example: External TLS-terminating load balancer	(硬件) Secure Enclave, Secure Boot (系统) Signed System Volume, Swift on Server (软件) Code Signing, Sandboxing
	无特权运行时访问 No privileged runtime access	后门和运维 Shell Example: Shell access by SREs	No remote shell. Only pre-specified, structured, and audited logs/metrics can leave the node User data is reviewed by multiple indepedent layers
	可验证的透明性 Verifiable transparency	没有经过安全专家验证的代码	Every production build of PCC publicly available

Outline

1 大模型推理 2
1.1 服务系统 3
1.2 阶段 5
1.3 优化的挑战 7
2 隐私云计算 8
2.1 背景 9
2.2 要求 10
3 无状态计算 11
3.1 定义 12
3.2 数据管理 13
3.3 传输 19
3.4 推测执行 21
3.5 状态残留 23
4 不可针对性 24
4.1 定义 25
4.2 提高针对性 27
5 总结 28
5.1 大模型推理优化的发展方向 29
5.2 相关的论文 30
5.3 工业界的推理系统 32
5.4 大模型推理与可信计算 33
6 附录 35

3.1 定义

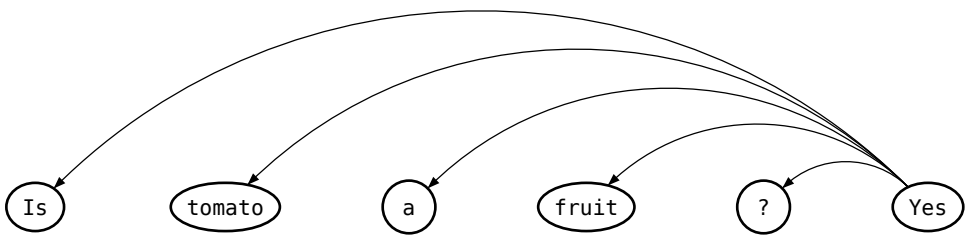
无状态计算

“Private Cloud Compute must use the personal user data that it receives exclusively for the purpose of fulfilling the user’s request. This data must never be available to anyone other than the user, not even to Apple staff, not even during active processing. And **this data must not be retained**, including via logging or for debugging, after the response is returned to the user. In other words, we want a strong form of stateless data processing where **personal data leaves no trace** in the PCC system.”

在请求完成之后，系统不应该保留任何用户数据。这意味着，系统不应该保留用户的输入、输出、中间状态等数据。这样可以避免用户数据的泄露和滥用。

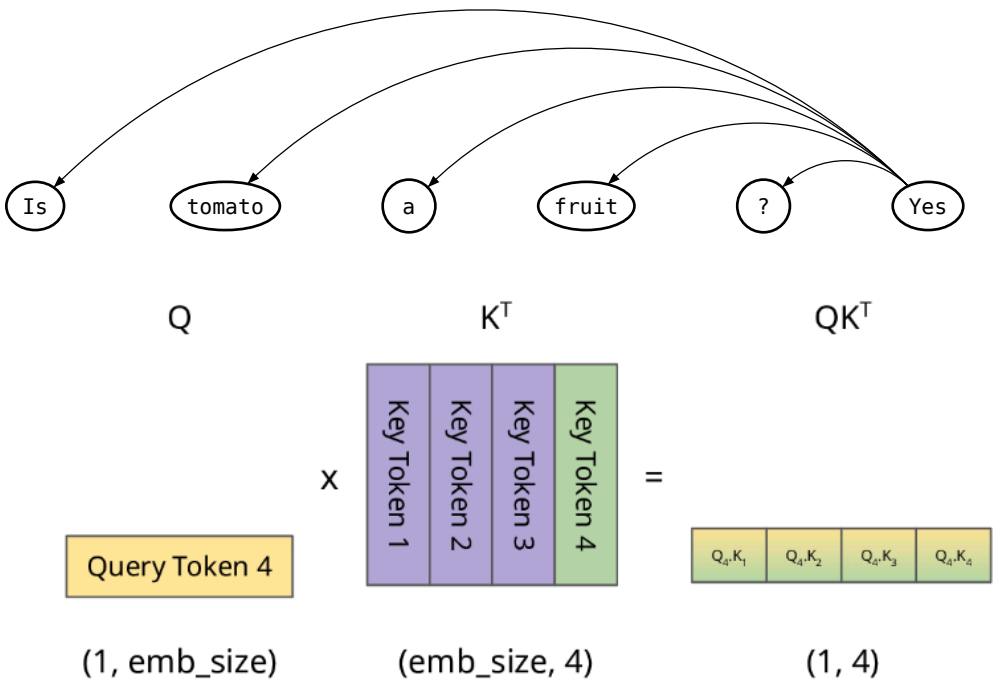
3.2 数据管理

KV Cache: 空间换时间



3.2 数据管理

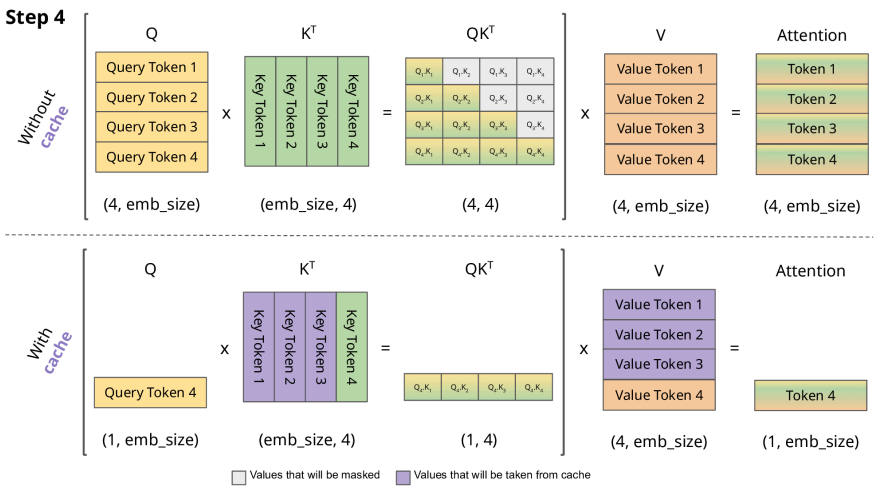
KV Cache: 空间换时间



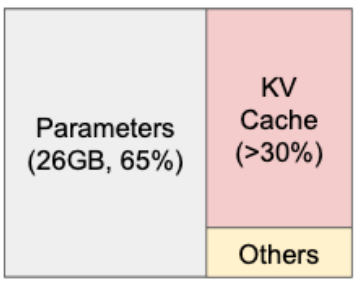
,

3.2 数据管理

KV Cache: 空间换时间



Transformers use attention mechanisms that compute attention scores between tokens. The KV Cache helps by storing previously computed key-value pairs, allowing the model to quickly access and reuse them for new tokens, avoiding redundant calculations.



NVIDIA A100 40GB

Memory layout when serving an LLM with 13B parameters on NVIDIA A100. The parameters (gray) persist in GPU memory throughout serving. The memory for the KV cache (red) is (de)allocated per serving request. A small amount of memory (yellow) is used ephemerally for activation.

3.2 数据管理

PagedAttention: 分页管理

PagedAttention* 是一种将注意力矩阵划分为较小页面的技术。这种方法一定程度上解决了内存碎片问题，因此，PagedAttention 已成为大模型推理系统中动态内存分配的事实标准。

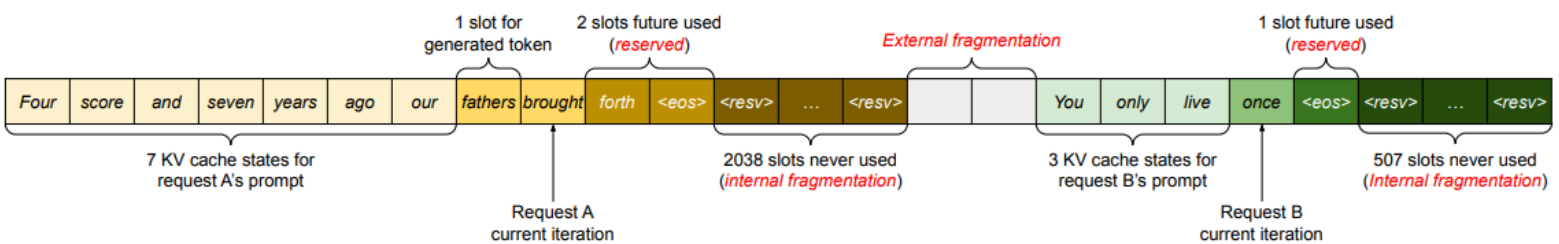


Figure 3. KV cache memory management in existing systems. Three types of memory wastes – reserved, internal fragmentation, and external fragmentation – exist that prevent other requests from fitting into the memory. The token in each memory slot represents its KV cache. Note the same tokens can have different KV cache when at different positions.

* Efficient Memory Management for Large Language Model Serving with PagedAttention

3.2 数据管理

Paged Attention

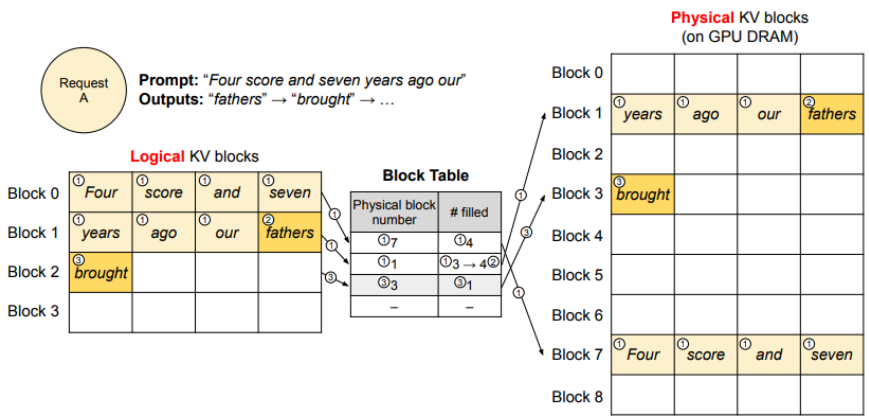


Figure 6. Block table translation in vLLM.

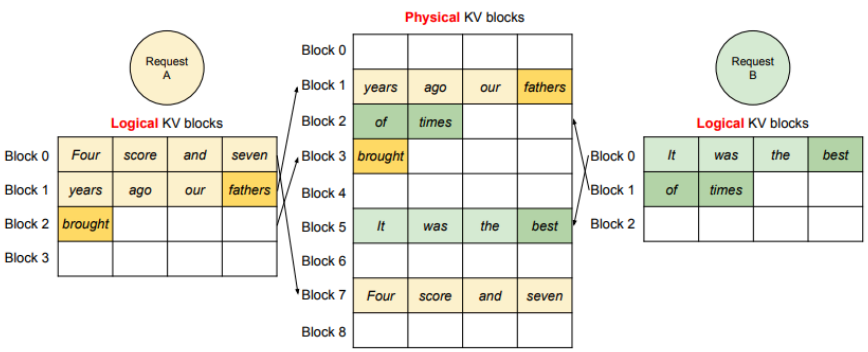


Figure 7. Storing the KV cache of two requests at the same time in vLLM.

注意到在请求完成后，这些分页数据和相关的中间结果会被释放或清除，不会留下任何可被后续请求利用的状态或痕迹。每个请求的输入和输出是独立的。PagedAttention 仅影响计算的内存管理，不会影响请求的无状态性。

3.2 数据管理

Prefix Caching

```
[Instructions]
You are an AI chatbot. You are having a conversation
with a human by following rules:
- You do not have a name.
- You are helpful, creative, clever, and friendy.
[Examples]
Human: Hello, who are you?
Al: I am an AI chatbot. How can I help you?

[Q1]
Human: Tell me about the second world war.

[Q2]
Human: What is the capital of France?

[Q3]
Human: What is the weather like in Paris today?
```

3.2 数据管理

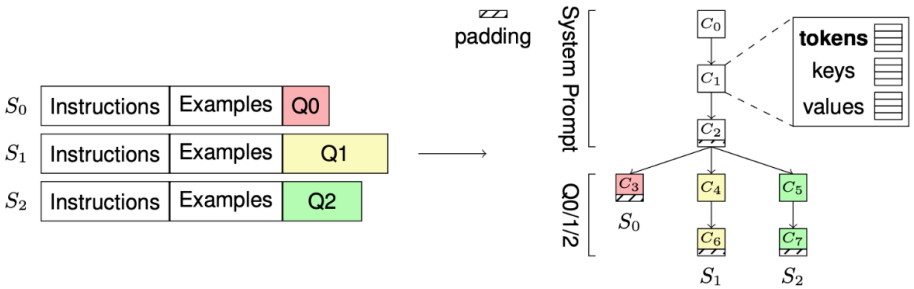
Prefix Caching

```
[Instructions]
You are an AI chatbot. You are having a conversation
with a human by following rules:
- You do not have a name.
- You are helpful, creative, clever, and friendy.
[Examples]
Human: Hello, who are you?
Al: I am an AI chatbot. How can I help you?

[Q1]
Human: Tell me about the second world war.

[Q2]
Human: What is the capital of France?

[Q3]
Human: What is the weather like in Paris today?
```



3.2 数据管理

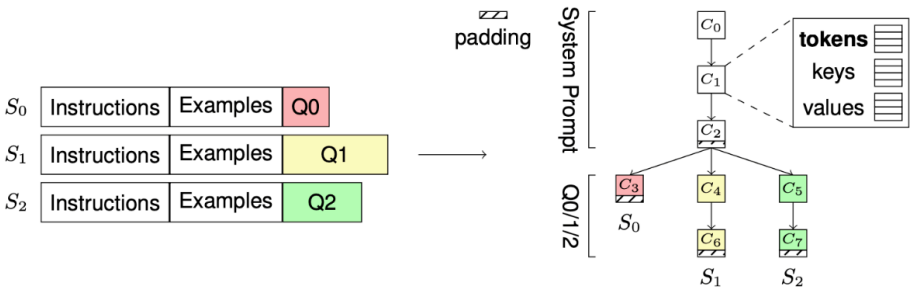
Prefix Caching

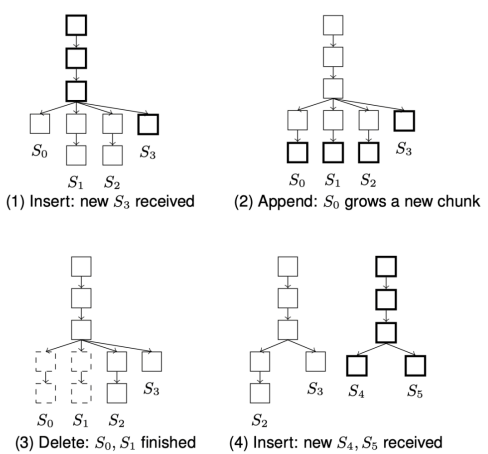
```
[Instructions]
You are an AI chatbot. You are having a conversation
with a human by following rules:
- You do not have a name.
- You are helpful, creative, clever, and friendy.
[Examples]
Human: Hello, who are you?
Al: I am an AI chatbot. How can I help you?

[Q1]
Human: Tell me about the second world war.

[Q2]
Human: What is the capital of France?

[Q3]
Human: What is the weather like in Paris today?
```





3.2 数据管理

Prefix Caching & 无状态计算

Prefix Caching 通过存储之前生成的键（keys）和值（values）来提高自回归模型的推理效率，但在这一过程中，可能会引入一些状态，这些状态的存在可能会影响无状态计算的原则。

数据泄露

如果缓存中的数据没有适时更新或删除，过期的信息可能会在意外情况下被泄露。例如，某个用户的敏感查询结果未能及时清除，可能会被后续请求访问到。

信息推断

由于 Prefix Caching 将历史生成信息与当前查询关联。在推理过程中，如果前缀缓存被重复使用或共享给不同的推理任务，攻击者可能通过比较不同任务的输出，推断出前一个输入的信息。例如，在生成式任务中，攻击者可以通过观察缓存的相似性，来推测用户的部分输入。

访问模式分析

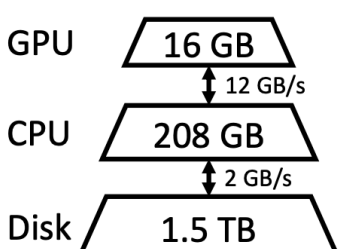
Prefix Caching 可能导致某些查询或生成过程频繁访问缓存，这种访问模式的可见性可能使攻击者能够分析并推测出缓存内容。通过访问模式，攻击者可能会获取到敏感信息的特征或上下文。

3.3 传输

KV Cache Offloading

KV cache 保存了每个生成步骤的键（Key）和值（Value），避免重复计算。然而，随着生成的标记数增加，KV cache 会不断变大，最终占用大量的 GPU 内存。

KV Cache Offloading 通过将这些缓存数据从高性能但有限的 GPU 内存中转移到较为廉价但速度较慢的内存设备，以释放 GPU 内存，供其他计算操作使用。当模型需要访问这些转移出去的 KV 缓存时，再从 CPU 内存或存储中按需重新加载到 GPU 内存进行计算。

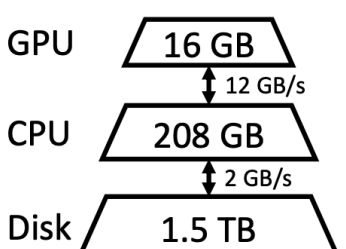


3.3 传输

KV Cache Offloading

KV cache 保存了每个生成步骤的键（Key）和值（Value），避免重复计算。然而，随着生成的标记数增加，KV cache 会不断变大，最终占用大量的 GPU 内存。

KV Cache Offloading 通过将这些缓存数据从高性能但有限的 GPU 内存中转移到较为廉价但速度较慢的内存设备，以释放 GPU 内存，供其他计算操作使用。当模型需要访问这些转移出去的 KV 缓存时，再从 CPU 内存或存储中按需重新加载到 GPU 内存进行计算。



KV Cache Offloading & 无状态计算

数据泄露

在无状态计算中，理想情况下每个请求结束后所有状态都应被清除，而且状态不应该在请求之间进行任何形式的存储或复用。如果 KV Cache 被 offload 到 CPU 或硬盘，可能会导致状态数据被存储在较长期的存储介质上。尤其是当缓存的数据保留在硬盘上时，它在多个请求之间的清除可能会不如内存中的状态那么及时，从而违反无状态计算的要求。

3.3 传输

Data Duplication

在多服务器环境中，KV Cache 可能会被复制到不同的节点上，以便在多个节点上并行执行推理任务。

Data Duplication 由 master 负责，它将 KV Cache 复制到多个节点上，以便在多个节点上并行执行推理任务。

3.3 传输

Data Duplication

在多服务器环境中，KV Cache 可能会被复制到不同的节点上，以便在多个节点上并行执行推理任务。

Data Duplication 由 master 负责，它将 KV Cache 复制到多个节点上，以便在多个节点上并行执行推理任务。

Data Duplication & 无状态计算

状态残留

当多个节点拥有 KV Cache 的副本时，执行结束后，由于同步或者过期策略的问题，这些副本可能仍然存在于内存或存储中。如果 KV Cache 中存储了敏感信息（如用户输入、生成的内容等），数据重复可能导致多个节点持有这些敏感数据的副本。在执行结束后，这些副本仍可能被访问或分析，增加了隐私泄露的风险。

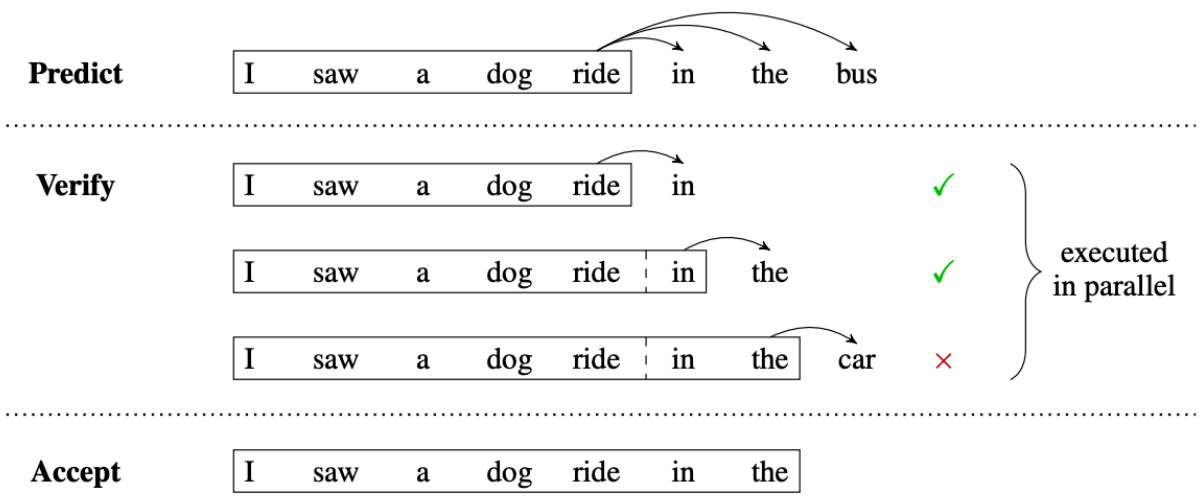
差分分析

如果 KV Cache 的状态在多个节点上存在多份副本，攻击者可能会通过分析这些副本之间的差异，推测出某些用户的输入或敏感信息。

3.4 推测执行

Speculative Inference

在这种方法中，系统会基于当前的输入和模型状态，提前生成多个可能的输出或结果。模型会对这些结果评估，选择最有可能的结果返回给用户。这种方法可以提高系统的响应速度。



3.4 推测执行

Speculative Inference & 无状态计算

预测性推理本身并不违反无状态计算的原则，因为它只是提前生成可能的结果，而不会保留这些结果。然而，预测性推理可能会导致一些状态残留，例如生成的结果可能会被缓存，这可能会增加隐私泄露的风险。

缓存泄露

如果生成的结果被缓存，这些结果可能会在执行结束后仍然存在于缓存中，增加了隐私泄露的风险。

信息推断

预测用的数据库可能会包含敏感信息，攻击者可能会通过分析这些数据库，推测出某些用户的输入或敏感信息。

3.5 状态残留

优化	残留的状态	储存位置	缓解策略	攻击			
				数据泄露	信息推断	访问模式分析	差分分析
Prefix Caching	Token KV Cache	GPU 内存 CPU 内存	过期机制 隔离	✓	✓	✓	
Disk Offloading	KV Cache	外置存储器 (SSD, Hard Drive)	加密	✓			
Data Duplication	KV Cache	GPU 内存 CPU 内存	过期同步机制	✓			✓
Database-based Speculative Inference	Token	GPU 内存 CPU 内存	差分隐私 使用小模型预测	✓	✓		

Outline

1 大模型推理 2
1.1 服务系统 3
1.2 阶段 5
1.3 优化的挑战 7
2 隐私云计算 8
2.1 背景 9
2.2 要求 10
3 无状态计算 11
3.1 定义 12
3.2 数据管理 13
3.3 传输 19
3.4 推测执行 21
3.5 状态残留 23
4 不可针对性 24
4.1 定义 25
4.2 提高针对性 27
5 总结 28
5.1 大模型推理优化的发展方向 29
5.2 相关的论文 30
5.3 工业界的推理系统 32
5.4 大模型推理与可信计算 33
6 附录 35

4.1 定义

不可针对性

An attacker should not be able to attempt to compromise personal data that belongs to specific, targeted Private Cloud Compute users without attempting a broad compromise of the entire PCC system. This must hold true even for exceptionally sophisticated attackers who can attempt physical attacks on PCC nodes in the supply chain or attempt to obtain malicious access to PCC data centers. In other words, a limited PCC compromise must not allow the attacker to **steer requests from specific users to compromised nodes**; targeting users should require a wide attack that's likely to be detected. To understand this more intuitively, contrast it with a traditional cloud service design where every application server is provisioned with database credentials for the entire application database, so a compromise of a single application server is sufficient to access any user's data, even if that user doesn't have any active sessions with the compromised application server.

4.1 定义

不可针对性

An attacker should not be able to attempt to compromise personal data that belongs to specific, targeted Private Cloud Compute users without attempting a broad compromise of the entire PCC system. This must hold true even for exceptionally sophisticated attackers who can attempt physical attacks on PCC nodes in the supply chain or attempt to obtain malicious access to PCC data centers. In other words, a limited PCC compromise must not allow the attacker to **steer requests from specific users to compromised nodes**; targeting users should require a wide attack that's likely to be detected. To understand this more intuitively, contrast it with a traditional cloud service design where every application server is provisioned with database credentials for the entire application database, so a compromise of a single application server is sufficient to access any user's data, even if that user doesn't have any active sessions with the compromised application server.

PCC 要求不能像传统设计中，攻陷一个服务器就可以访问所有用户的数据。如果仅仅攻陷部分节点，攻击者不应该能够访问到特定用户的数据。

4.1 定义

设 $S = \{S_1, S_2, ..., S_n\}$ 为系统中所有服务器的集合，每个服务器 S_i 的能力用 $C(S_i)$ 表示。这些服务器处理的请求集合表示为 $R(S) = \{R(S_1), R(S_2), ..., R(S_n)\}$。如果对于任何服务器子集 $T = \{T_1, T_2, ..., T_m\} \subseteq S$，攻陷特定用户 u 的数据被攻陷的概率如下所示，则该系统被认为是不可针对的:

$$P(u \in R(T)) = \frac{\sum_{i=1}^{m} C(T_i)}{\sum_{i=1}^{n} C(S_i)}$$

-
-

4.1 定义

设 $S = \{S_1, S_2, ..., S_n\}$ 为系统中所有服务器的集合，每个服务器 S_i 的能力用 $C(S_i)$ 表示。这些服务器处理的请求集合表示为 $R(S) = \{R(S_1), R(S_2), ..., R(S_n)\}$。如果对于任何服务器子集 $T = \{T_1, T_2, ..., T_m\} \subseteq S$，攻陷特定用户 u 的数据被攻陷的概率如下所示，则该系统被认为是不可针对的:

$$P(u \in R(T)) = \frac{\sum_{i=1}^{m} C(T_i)}{\sum_{i=1}^{n} C(S_i)}$$

目标

- 增加 Hit: $\sum C(T_i)$
- 减少 Miss: $\sum C(S_i) - \sum C(T_i)$

4.2 提高针对性

优化	增加 Hit	减少 Miss
Duplication	✓ 复制目标的缓存到其他受害者	✓ 复制非目标的缓存到非受害者
Pulling	✓ 从目标的缓存中拉取数据	
Request Migration	✓ 迁入目标的请求	✓ 迁出非目标的请求
Priority-Based Scheduling	✓ 提高目标的请求的优先级	✓ 降低非目标的请求的优先级
Instance Flip	✓ 让受害者节点转变成预填充节点	

Outline

1 大模型推理 2
1.1 服务系统 3
1.2 阶段 5
1.3 优化的挑战 7
2 隐私云计算 8
2.1 背景 9
2.2 要求 10
3 无状态计算 11
3.1 定义 12
3.2 数据管理 13
3.3 传输 19
3.4 推测执行 21
3.5 状态残留 23
4 不可针对性 24
4.1 定义 25
4.2 提高针对性 27
5 总结 28
5.1 大模型推理优化的发展方向 29
5.2 相关的论文 30
5.3 工业界的推理系统 32
5.4 大模型推理与可信计算 33
6 附录 35

5.1 大模型推理优化的发展方向

vLLM 版本的路线图包括以下功能更新:

版本	日期	内存	传输	批处理	并行	调度	模型
v0.1	2306	Paging		Continuous Batching			MQA, GQA
v0.2	2309				Better TP & EP Support		AWQ
v0.3	2401	Prefix Caching					GPTQ
v0.4	2404		Optimize Distributed Communication	Chunked Prefill	Speculative Inference		
v0.5	2407	CPU Offloading			Support PP	Schedule multiple GPU steps in advances	FP8
v0.6	2409					Asynchronous output processor	

发展方向

- 支持大规模的并行化和负载均衡
- 细致的调度和资源复用
- 自动化和自适应的优化策略
- 针对不同场景进行定制化的优化策略

5.1 大模型推理优化的发展方向

vLLM 版本的路线图包括以下功能更新:

版本	日期	内存	传输	批处理	并行	调度	模型
v0.1	2306	Paging		Continuous Batching			MQA, GQA
v0.2	2309				Better TP & EP Support		AWQ
v0.3	2401	Prefix Caching					GPTQ
v0.4	2404		Optimize Distributed Communication	Chunked Prefill	Speculative Inference		
v0.5	2407	CPU Offloading			Support PP	Schedule multiple GPU steps in advances	FP8
v0.6	2409					Asynchronous output processor	

发展方向

- 支持大规模的并行化和负载均衡
- 细致的调度和资源复用
- 自动化和自适应的优化策略
- 针对不同场景进行定制化的优化策略

合规的挑战

- 需要大量的状态管理来实现并行化和负载均衡
- 精致的调度可能会引入定向攻击和隐私泄露
- 自适应的优化策略可能会导致优化用数据被持续泄露
- 攻击者可以通过推测用户的使用场景来推断用户的隐私信息

5.2 相关的论文

S: Stateless computation E: Enforceable guarantees T: Non-targetability P: No privileged runtime access V: Verifiable transparency

类别	优化	威胁	Orca 2206	FlexGen 2303	FastServe 2305	SpecInfer 2305	vLLM 2309	REST 2311	Splitwise 2311	SGLang 2312	Lookahead 2312	Sarathi 23-24	InfiniteLLM 2401	DistServe 2401	Medusa 2401	TetriInfer 2401	AttentionStore 2403	LoongServe 2404	Andes 2405	Llumnix 2406	Preble 2407	MInference 2407	TR 2408
内存	Paging						Initial			✓		✓				✓							
	Prefix Caching	SE								✓											✓		
	Disk Offloading	SE		✓													✓						
传输	Duplication	ST																					
	Pulling	T												✓									
	Request Migration																	✓		✓			
	Disaggregated Arch								✓					✓		✓							
批处理	Continuous Batch		Initial		✓	✓	✓					✓		✓		✓							
	Chunked Prefill											Initial				✓					✓		
	Prepack Prefill													✓		✓							
并行	Speculation					✓		✓		✓	✓				✓								
	Context-Based Speculation	S						✓															
	Database-Based Speculation	S									✓												✓
	Tensor Parallelism					✓																	
	Sequence Parallelism																	✓					
调度	Priority-Based	T			✓					✓		✓				✓			✓	✓	✓		
	Request-Level Prediction	T			✓	✓										✓							
	Machine-level Scheduler	E			✓				✓				✓			✓		✓			✓		
	Instance Flip	T							✓							✓							
	Global Profiling	P		✓					✓					✓								✓	
审计	Non Open-Source	V														✓							✓

5.2 相关的论文

Misc

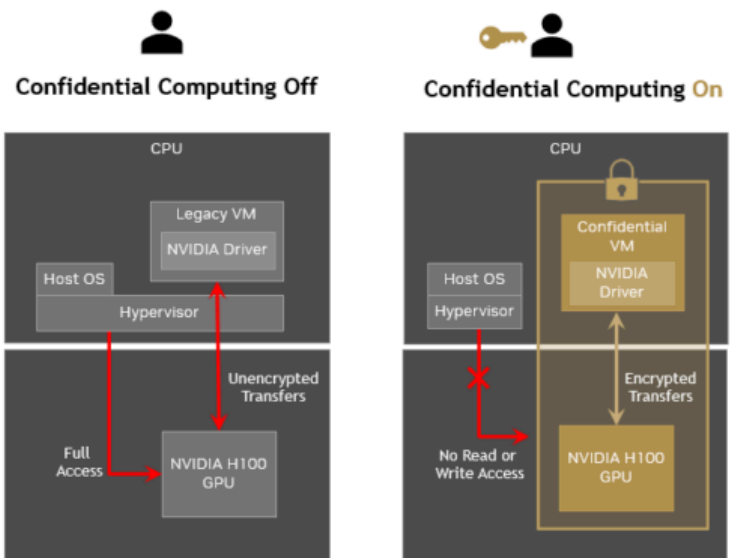
Title	Keywords	Contributions
Prompt Cache	Prefill, Memory	Reuse attention states across different LLM prompts. Parse the prompt and use reusable text segments(snippet)
Layer-wise Transmission	Transmission	Transmit each layer's output to the next layer in the pipeline, instead of transmitting the entire model's output
LightLLM	Interface	Use http as the interface to the system
SkyPilot	Cross Region & Cloud	Given a job and its resource requirements (CPU/GPU/TPU), SkyPilot automatically figures out which locations (zone/region/cloud) have the compute to run the job, then sends it to the cheapest one to execute
MLC LLM	Efficient Execution	Enable efficient execution of large language models across a wide range of hardware platforms, including mobile devices, edge devices, and even web browsers
vAttention	Virtual Memory	stores KV-cache in contiguous virtual memory and leverages OS support for on-demand allocation of physical memory
MemServe	API, Framework	an elastic memory pool API managing distributed memory and KV caches across serving instances
CacheGen	Network, Streaming	CacheGen uses a custom tensor encoder, leveraging KV cache's distributional properties to encode a KV cache into more compact bitstream representations
DynamoLLM	Energy	It exploits heterogeneity in inference compute properties and fluctuations in inference workloads to save energy
MInference	Prefill, Long Context	Addresses the expensive computational cost and the unacceptable latency of the attention calculations in the pre-filling stage of long-context LLMs by leveraging dynamic sparse attention with spatial aggregation patterns
Shared Attention	Attention	directly sharing pre-computed attention weights across multiple layers in LLMs
SnapKV	Compression	Observing that specific tokens within prompts gain consistent attention from each head during generation, our methodology not only retrieve crucial information but also enhances processing efficiency

5.3 工业界的推理系统

类别	优化	威胁	vLLM Open Source	LightLLM Open Source	FlexFlow Open Source	SGLang Open Source	Mooncake Moonshot	DeepSpeed Microsoft	TensorRT NVIDIA	TGI Hugging Face	Llama Intel	LMDeploy Shanghai AI lab	fastllm Open Source	rtp-llm Alibaba	MindIE Huawei
内存	Paging		✓			✓	✓	✓		✓				✓	
	Token Attention			✓											
	Prefix Caching	S	✓						✓						
	Disk Offloading	SE				✓	✓		✓						✓
	Multi-Query Attention								✓						
传输	Duplication	ST					✓								
	Pulling	T													
	Request Migration														
	Disaggregated Arch		✓				✓								✓
批处理	Continuous Batch		✓		✓		✓	✓	✓	✓	✓	✓	✓	✓	
	Chunked Prefill		✓				✓	✓							
	Prepack Prefill														
并行	Speculation		✓		✓	✓			✓	✓				✓	✓
	Tensor Parallelism									✓					✓
	Sequence Parallelism														
调度	Priority-Based	T				✓	✓	✓							
	Request-Level Prediction	T		✓		✓									
	Machine-level Scheduler	E				✓	✓								
	Instance Flip	T													
	Global Profiling	P					✓								
审计	Non Open-Source	V					✓		✓						

5.4 大模型推理与可信计算

NVIDIA H100 GPU



硬件隔离: H100 使用硬件级的安全隔离机制，例如通过虚拟化技术或受信执行环境（Trusted Execution Environment, TEE），来确保多个工作负载可以在同一台机器上运行时相互隔离，防止数据被未授权的实体访问。它利用专门的硬件模块，如安全内存加密（SME）和安全加密虚拟化（SEV），来保护内存中的数据。

数据加密: 在数据的存储和传输过程中，H100 通过硬件支持的加密机制（例如 AES 加密）来确保数据在 GPU 内部、内存中以及网络上传输时都是加密状态，防止数据在使用过程中被泄露。H100 的加密引擎可以对内存中的数据进行实时加密和解密。

硬件根信任（Root of Trust）: H100 采用硬件根信任机制，它通过引导安全启动链，确保设备从硬件层开始处于可信状态。这意味着从设备启动到运行机密工作负载的整个过程中，只有经过验证的、未篡改的软件和固件才可以执行。

远程证明（Remote Attestation）: H100 支持远程证明功能，这可以让外部用户验证 GPU 是否处于可信的状态，从而确保数据仅在安全的环境中运行。通过远程证明机制，用户可以验证系统的配置、固件版本以及运行的工作负载的完整性。

5.4 大模型推理与可信计算

Confidential Computing on nVIDIA H100 GPU: A Performance Benchmark Study

由专注于 Web3 可验证计算的初创公司 Phala Network 于 2024 年 9 月 6 日发布在 ArXiv 上

观察

- 对于吞吐量指标，平均的性能损失小于 7%
  - 随着 token 数量的增加，性能损失逐渐减小
  - 对于 70B 参数的模型，性能损失可以忽略不计
- 预填充阶段受到的影响相对更大
  - 可能是由于初始加载和加密开销
  - 对于小模型，第一个 token 的生成平均延迟增加了接近 20%

Outline

1 大模型推理 2
1.1 服务系统 3
1.2 阶段 5
1.3 优化的挑战 7
2 隐私云计算 8
2.1 背景 9
2.2 要求 10
3 无状态计算 11
3.1 定义 12
3.2 数据管理 13
3.3 传输 19
3.4 推测执行 21
3.5 状态残留 23
4 不可针对性 24
4.1 定义 25
4.2 提高针对性 27
5 总结 28
5.1 大模型推理优化的发展方向 29
5.2 相关的论文 30
5.3 工业界的推理系统 32
5.4 大模型推理与可信计算 33
6 附录 35

Thanks

6 附录

吞吐量损失 –7%

Confidential Computing on nVIDIA H100 GPU: A Performance Benchmark Study

Model	TPS (tokens/s)			QPS (req/s)		
	TEE-on	TEE-off	Overhead	TEE-on	TEE-off	Overhead
LLama-3.1-8B	123.2985	132.3618	6.85%	18.2141	18.8208	3.22%
Phi3-14B-128k	66.5845	69.7787	4.58%	7.1760	7.3456	2.31%
Llama-3.1-70B	2.4822	2.4789	-0.13%[1]	0.8325	0.8295	-0.36%[2]

Table 1: Performance comparison of TEE-on and TEE-off modes for various models in terms of TPS (tokens per second) and QPS (queries per second).

计算资源的利用效率：大模型通常需要更强的计算资源，如更多的内存和更复杂的计算。这意味着当启用TEE时，虽然加密和保护数据的开销增加，但相较于整体的计算量，这些开销占比可能较小。而小模型本身计算量较少，因而加密和保护数据的开销在总计算量中的占比相对较大，导致表现出的性能损耗较明显。

TEE 的处理机制：TEE 会对模型的推理过程进行加密和数据隔离，这些操作对小模型的相对简单的计算任务来说，可能产生更多干扰。而大模型本身就具有大量的并发处理、计算层次复杂等特性，因此 TEE 的影响在复杂的计算流程中被“稀释”了。

I/O 密集度：小模型可能对 I/O 更敏感（如输入输出数据流），而 TEE 的机制通常会增加 I/O 开销，导致小模型的性能损失比大模型更明显。

6 附录

Confidential Computing on nVIDIA H100 GPU: A Performance Benchmark Study

Model	TTFT (s)			ITL (s)		
	TEE-on	TEE-off	Overhead	TEE-on	TEE-off	Overhead
LLama-3.1-8B	0.0288	0.0242	19.03%	1.6743	1.5549	7.67%
Phi3-14B-128k	0.0546	0.0463	18.02%	3.7676	3.5784	5.29%
Llama-3.1-70B	0.5108	0.5129	-0.41%[3]	94.8714	95.2395	-0.39%[4]

Table 2: Comparison of TTFT (Time to First Token) and ITL (Inter Output Token Latency) for TEE-on and TEE-off modes across models.

Prefill 阶段受到的影响相对更大

初始加载和加密开销: 在 prefill 阶段，模型需要加载大量的初始上下文、参数等数据。而 TEE 开启时，会有数据加密和保护的额外开销，这会导致在 prefill 阶段的额外延迟。这部分工作在 TTFT 上表现得更明显，而在 ITL 阶段（生成后续 tokens）相对稳定，因为后续 tokens 的生成依赖于已经缓存或预处理的结果。

初始加载和加密开销: 在 prefill 阶段，模型需要加载大量的初始上下文、参数等数据。而 TEE 开启时，会有数据加密和保护的额外开销，这会导致在 prefill 阶段的额外延迟。这部分工作在 TTFT 上表现得更明显，而在 ITL 阶段（生成后续 tokens）相对稳定，因为后续 tokens 的生成依赖于已经缓存或预处理的结果。

上下文窗口的处理: 在 prefill 阶段，模型需要处理输入的完整上下文（或是大部分上下文），对于每个 token 的生成，TEE 的安全处理会增加上下文的开销，这在 TTFT 上表现为显著的影响。而 ITL 通常只涉及每次单个 token 的生成，计算量较小，影响较弱。

并行与串行处理的差异: prefill 阶段可能涉及更多的并行计算和预加载数据，在此过程中，TEE 可能会限制部分并行化的效率，使得整体 prefill 时间增加。而在 ITL 阶段，由于生成每个 token 的过程是逐步串行的，受 TEE 影响较小。

缓存与推理优化: prefill 过程中可能无法充分利用缓存机制，尤其是对于较小的模型，整个序列的推理需要在初始阶段完成较多的计算。而在生成后的每个 token 推理时，缓存的 KV 机制可以提高效率，抵消一部分 TEE 带来的开销。因此，ITL 阶段受影响较小。

6 附录

Confidential Computing on nVIDIA H100 GPU: A Performance Benchmark Study

Model	TPS - short (tokens/s)			TPS - medium (tokens/s)			TPS - long (tokens/s)		
	TEE-on	TEE-off	Overhead	TEE-on	TEE-off	Overhead	TEE-on	TEE-off	Overhead
LLama-3.1-8B	127.0310	136.8282	7.16%	122.9356	132.0464	6.90%	122.9705	131.7333	6.65%
Phi3-14B-128k	70.9799	74.7556	5.05%	66.1690	69.3104	4.53%	66.2987	69.4176	4.49%
Llama-3.1-70B	2.5983	2.6073	0.34%	2.4413	2.4374	-0.16%[5]	2.5245	2.5168	-0.30%[6]

Table 3: Performance comparison of TEE-on and TEE-off modes across different sequence lengths in terms of TPS (tokens per second). Short sequences are no longer than 100 tokens. Medium sequences are no longer than 500 tokens. Long sequences are between 501 and 1500 tokens.

固定开销的摊薄效应：TEE 引入的加密、数据保护等固定开销对于较短的序列（如 100 tokens 以内）来说，影响较大，因为短序列的计算时间本身较短，额外的保护开销占比更高。然而，随着 token 数量增加，模型的整体推理时间增加，固定的加密和保护开销在总计算中的占比减少，表现为 overhead 逐渐降低。

并行化处理效率提升：随着 token 数量的增加，模型在生成过程中可能会更多依赖并行处理和 GPU 的并发能力，而 TEE 对并行处理的影响较小，因此在处理更长的序列时，整体的计算效率提升，加密的开销变得更为可忽略。sv

缓存和推理优化的效果：长序列的推理过程中，模型更容易利用 KV cache 等缓存机制来减少重复计算。尤其是在生成后续 tokens 时，缓存能显著提高效率，进一步抵消了 TEE 带来的额外开销。而短序列则在生成第一个 token 时受缓存的作用较小，overhead 显得更高。

I/O 操作的相对减少：对于短序列，I/O 操作（如加载输入、预处理等）所占比例较大，而 TEE 的加密机制会显著增加 I/O 的时间成本。随着序列长度增加，计算的主要瓶颈转向模型推理本身，而 I/O 的比例相对降低，TEE 对 I/O 的影响也逐渐变得不那么显著。

计算复杂度的稀释：较长的序列意味着更多的推理步骤和计算，而每一步的加密和保护开销被这些额外的计算步骤稀释掉了。换句话说，随着推理步骤的增加，TEE 的影响不再占据推理时间的主要部分，模型的计算复杂度逐渐主导整个过程。

6 附录

Attention

https://zh.d2l.ai/chapter_attention-mechanisms/attention-cues.html

Inference Optimization

https://github.com/DefTruth/Awesome-LLM-Inference

https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

Parallelism

https://developer.nvidia.com/blog/demystifying-ai-inference-deployments-for-trillion-parameter-large-language-models/

Utilities

https://github.com/Trusted-AI/adversarial-robustness-toolbox

6 附录

Stateful Inference Systems

Static state States in traditional systems can be modified after creation and require various consistency and coherence mechanisms to support parallelism. In LLM inference, once KVs are computed for a sequence, their values do not change.

Regular computation patterns LLMs' transformer computation is regular. Its computing and memory consumption is determined by the model size, the prompt length, and the output generation length. The model size and a request's prompt length are known before execution, and output is generated one token per iteration. Thus, we can estimate the computing and memory consumption for every iteration.

6 附录

Quantization

Quantization is the process of reducing the precision of a model's weights and activations.

Sparsity

Sparsity is the process of setting a portion of the model's weights to zero. Then the model can be expressed as a sparse matrix.

Distillation

Distillation is the process of training a smaller model to mimic the behavior of a larger model.