大模型推理中的隐私云计算

Private Cloud Compute 中的无状态计算

管佳乐

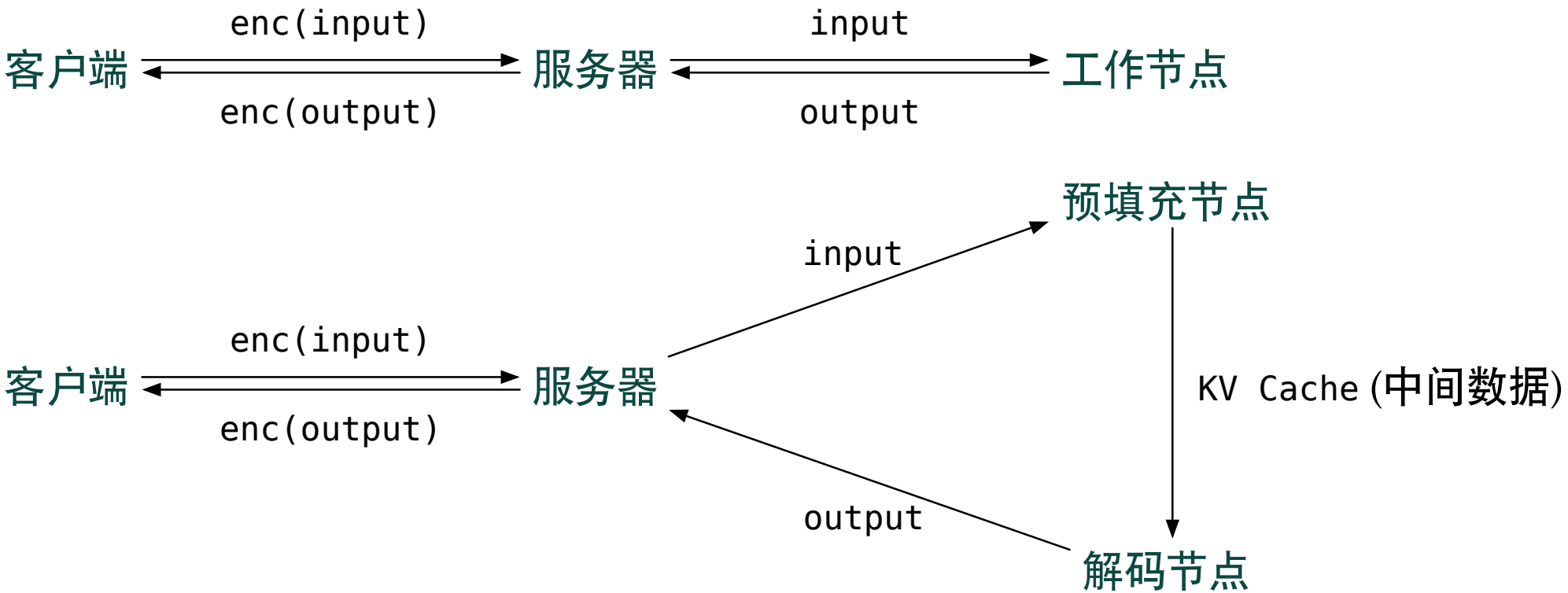
2024-10-09

Outline

Outline

1.1 服务系统

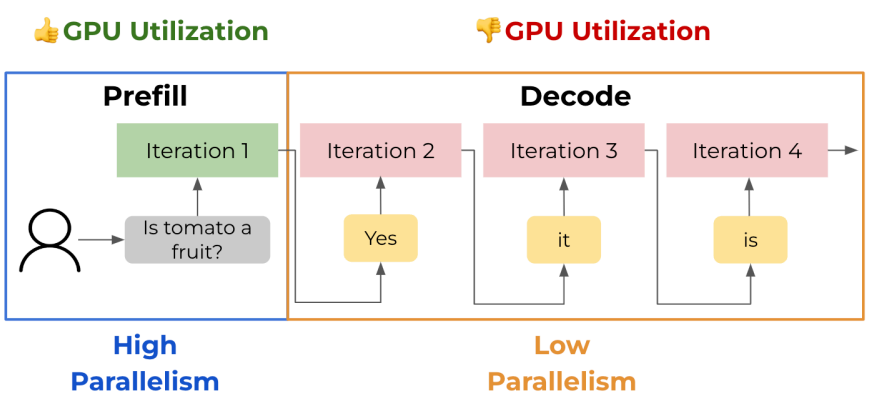
1.1 服务系统


enc(input)
客户端
服务器
input
工作节点
enc(output)
output
预填充节点
input
enc(input)
客户端
服务器
KV Cache (中间数据)
enc(output)
output
解码节点

1.2 阶段

预填充: Processing the input

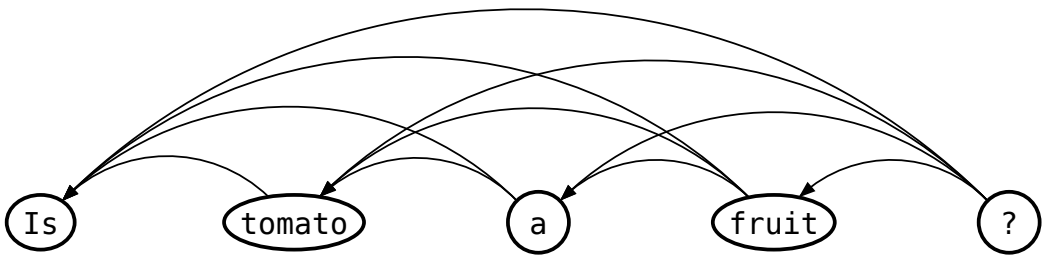
在预填充阶段，LLM 处理输入的 token 以计算中间状态 (KV Cache)，这些状态用于生成“第一个”新 token。每个 token 都依赖于之前的所有 token，但由于输入的完整内容是已知的，从宏观来看，这是一个 Matrix-Matrix 的操作，且高度并行化。这一过程有效地使 GPU 达到最大利用率。

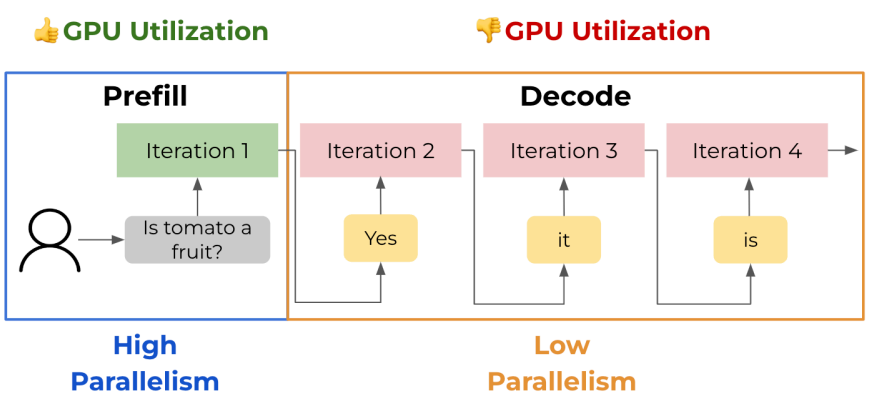


1.2 阶段

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。

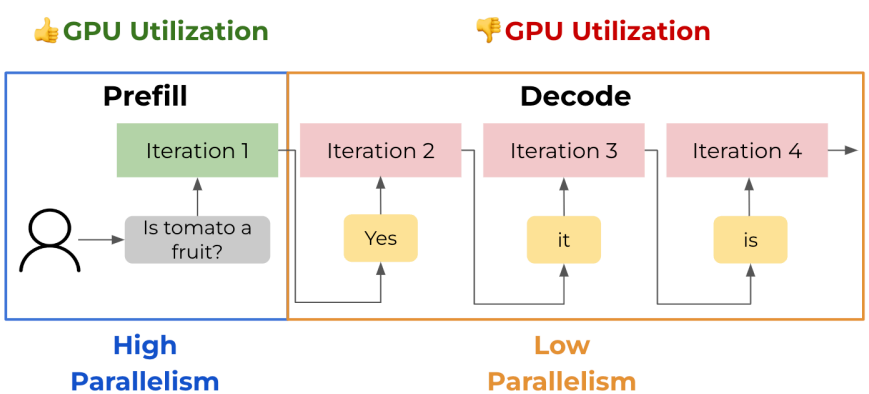


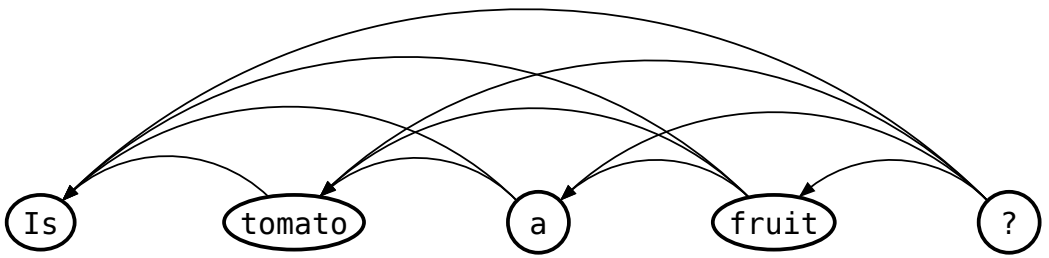


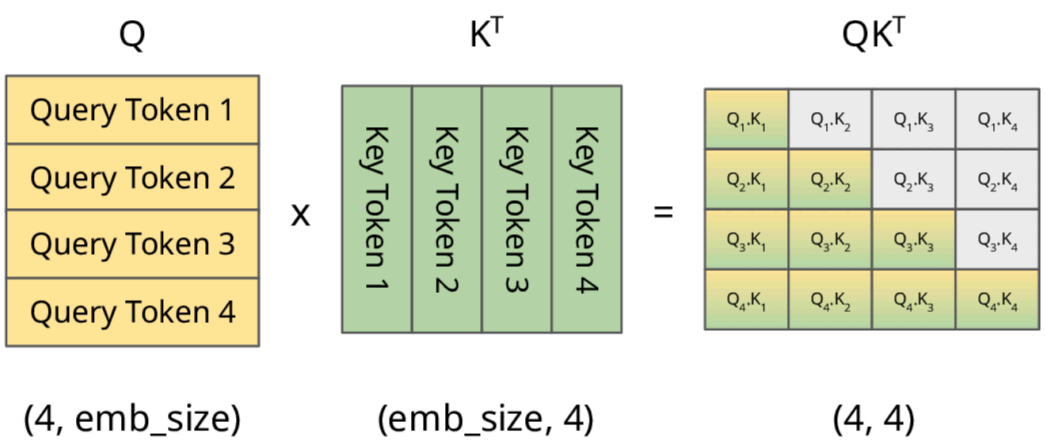
1.2 阶段

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。





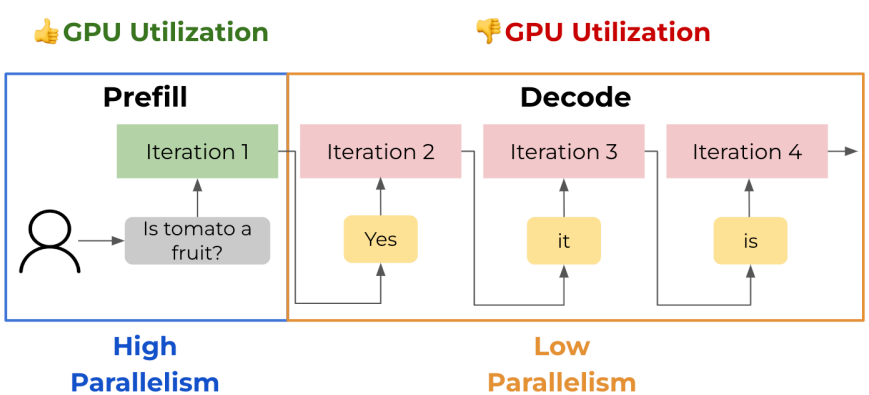


，

1.2 阶段

解码: Generating the output

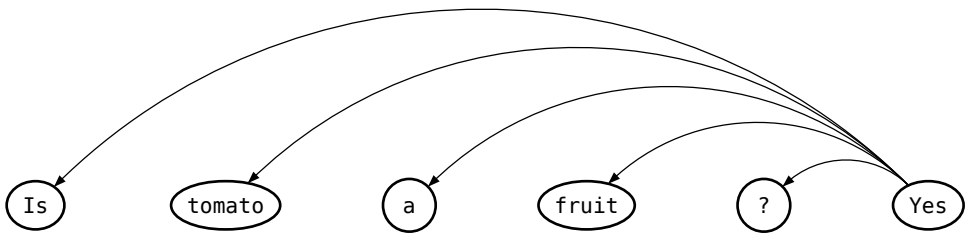
在解码阶段，LLM 以自回归的方式逐个生成输出标记，直到满足停止条件。每一个连续的输出标记都需要了解之前所有迭代的输出状态（KV Cache）。这一过程类似于 Matrix-Vector 操作，相较于预填充阶段，GPU 计算能力未被充分利用。延迟主要受数据（权重、键、值、激活）从内存传输到 GPU 的速度影响，而不是计算的速度。换句话说，这是一个受限于内存的操作。

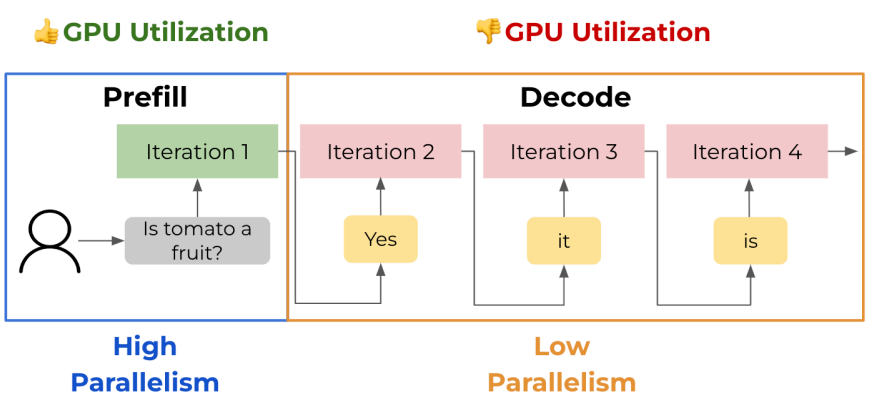


1.2 阶段

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。

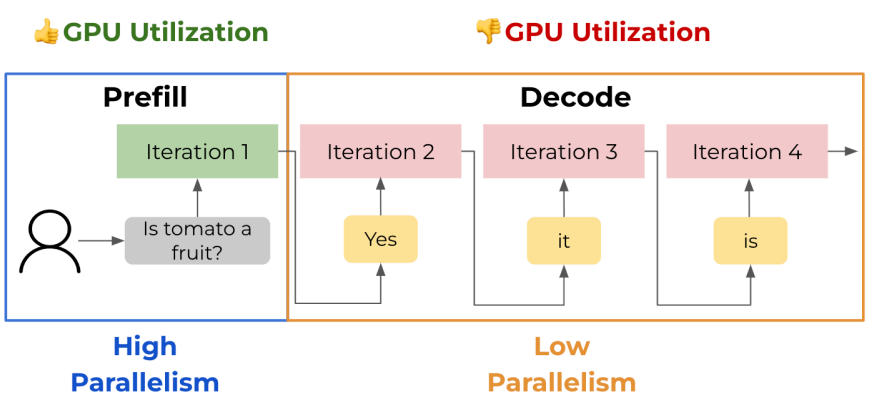


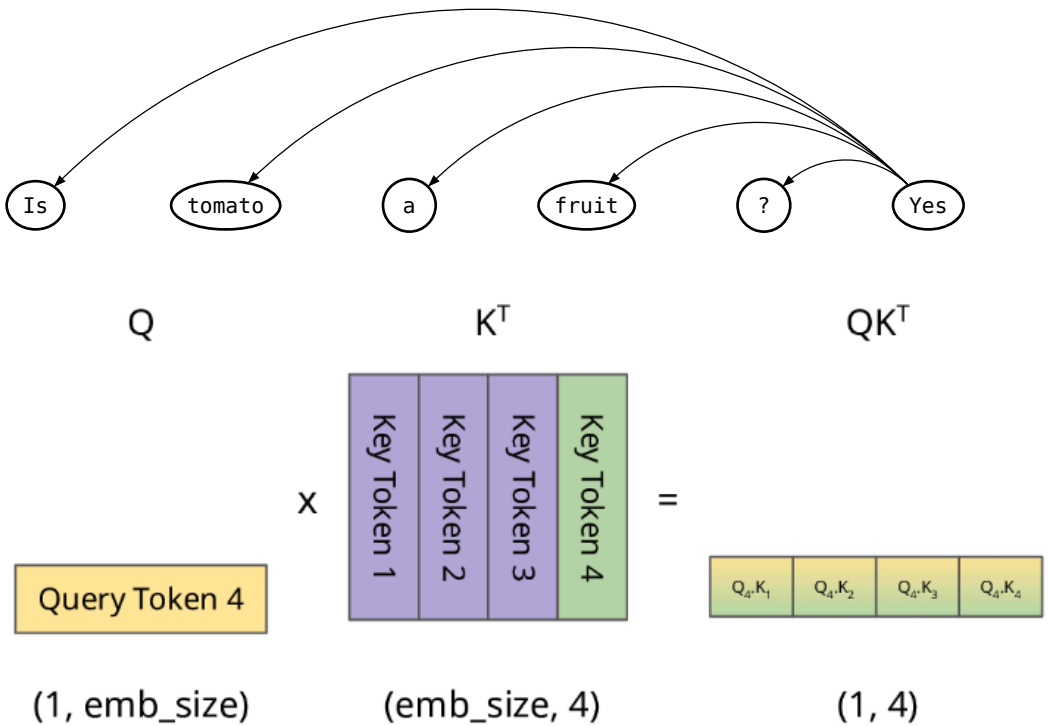


1.2 阶段

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

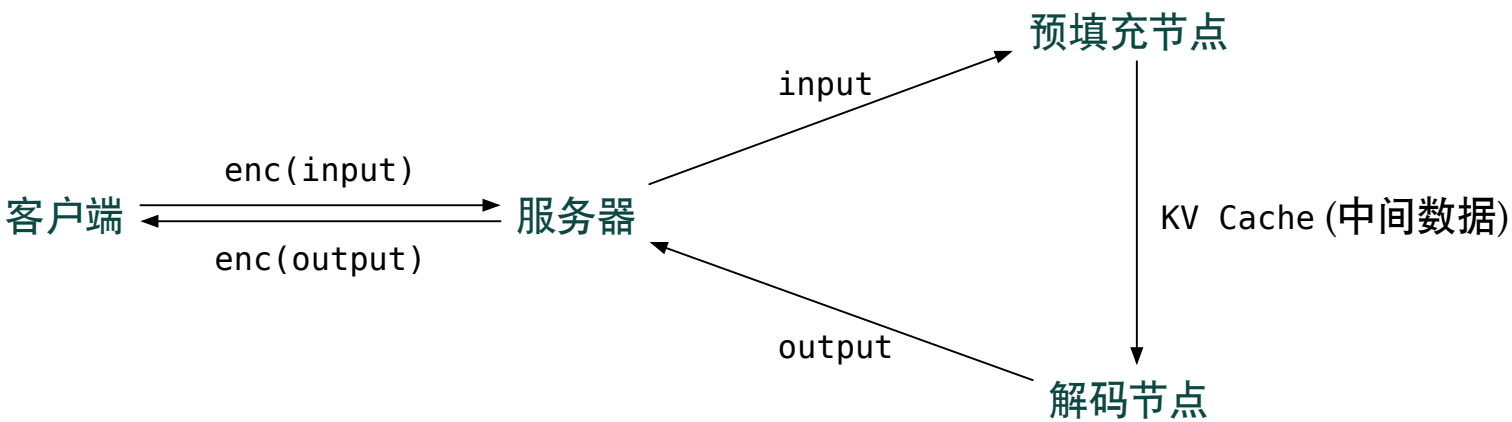
在输出结束之前，无法事先知道将生成多少个 token。因此，请求的执行时间和资源需求都是不可预测的。这种不确定性使得难以为每个请求分配合适的资源，需要动态调整资源分配策略。

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

2.1 背景



挑战:

- 无法使用端到端加密
- 无法在输入中使用匿名化

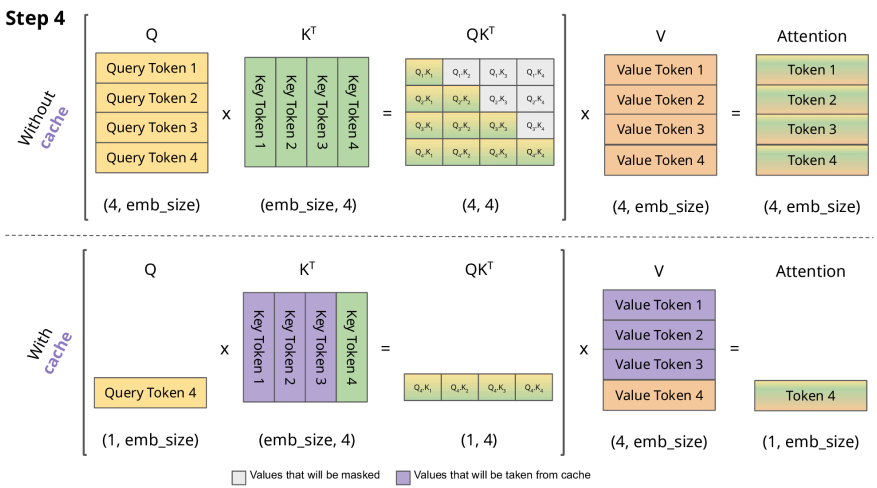
2.2 要求

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	Trace of data after processing 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
技术	不可针对性 Non-targetability	Targeted attack 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	Technical enforceability 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
生态	无特权运行时访问 No privileged runtime access	Privileged interfaces 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
生态	可验证的透明性 Verifiable transparency	Uninspected code	Every production build of PCC publicly available

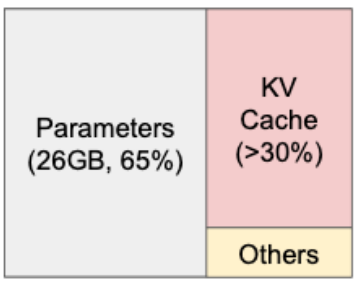
Outline

3.1 数据管理

KV Cache



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.



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.

3.1 数据管理

PagedAttention

PagedAttention* 是一种将注意力矩阵划分为较小页面的技术。这种方法几乎完美地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。

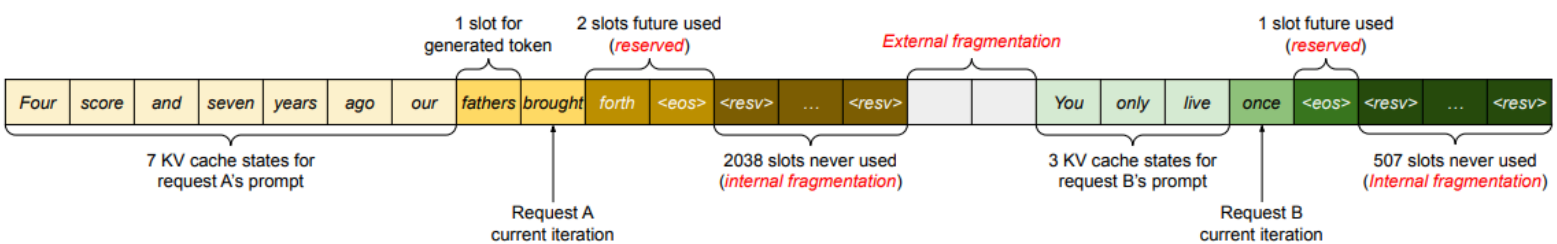


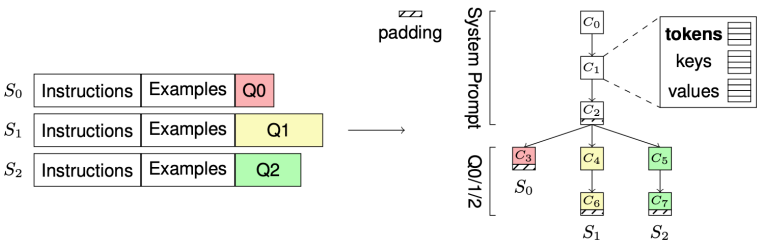
Figure 3. KV cache memory management in existing systems. Three types of memory wastes – reserved, internal fragmentation, and external fragmentation – exist that prevent other requests from fitting into the memory. The token in each memory slot represents its KV cache. Note the same tokens can have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

* Efficient Memory Management for Large Language Model Serving with PagedAttention

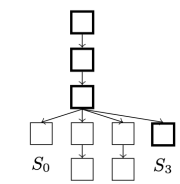
3.1 数据管理

Prefix Caching

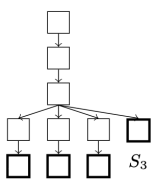
Prefix Caching* 是一种用于加速大模型推理(特别是语言模型推理)的技术。其主要目的是通过缓存和重用推理过程中已经计算过的部分，减少重复计算，从而提高效率。



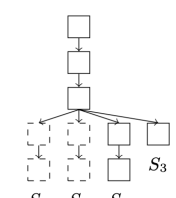
Prompt:
[Instructions]
You are an AI chatbot. You are having a conversation with a human by following rules:
- You do not have a name.
- You are helpful, creative, clever, and friendly
...
[Examples]
Human: Hello, who are you?
AI: I am an AI chatbot. How can I help you?
...
[Question]
Human: Tell me about the second world war.



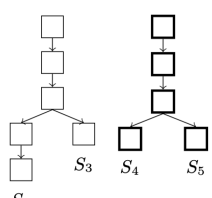
(1) Insert: new S_3 received



(2) Append: S_0 grows a new chunk



(3) Delete: S_0, S_1 finished



(4) Insert: new S_4, S_5 received

* ChunkAttention: Efficient Self-Attention with Prefix-Aware KV Cache and Two-Phase Partition. 2024

3.1 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。

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。

缓存泄露

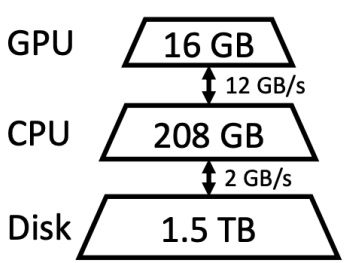
如果缓存中的数据没有适时更新或删除，过期的信息可能会在意外情况下被泄露。例如，某个用户的敏感查询结果未能及时清除，可能会被后续请求访问到。

3.2 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。

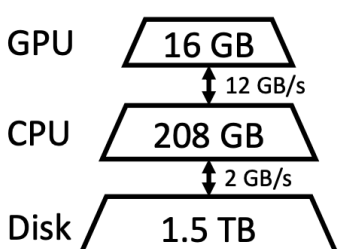


3.2 传输

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

如果 KV Cache 中存储了敏感信息（如用户输入、生成的内容等），KV Cache Offloading 可能会导致这些敏感信息被转移到不安全的存储介质中，增加了隐私泄露的风险。

3.2 传输

Pull, 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。

Pull 由执行的节点负责，它将 KV Cache 从其他节点拉取到本地节点上，以便在本地节点上执行推理任务。

Data Duplication 由调度器负责，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。

3.2 传输

Pull, 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。

Pull 由执行的节点负责，它将 KV Cache 从其他节点拉取到本地节点上，以便在本地节点上执行推理任务。

Data Duplication 由调度器负责，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，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。

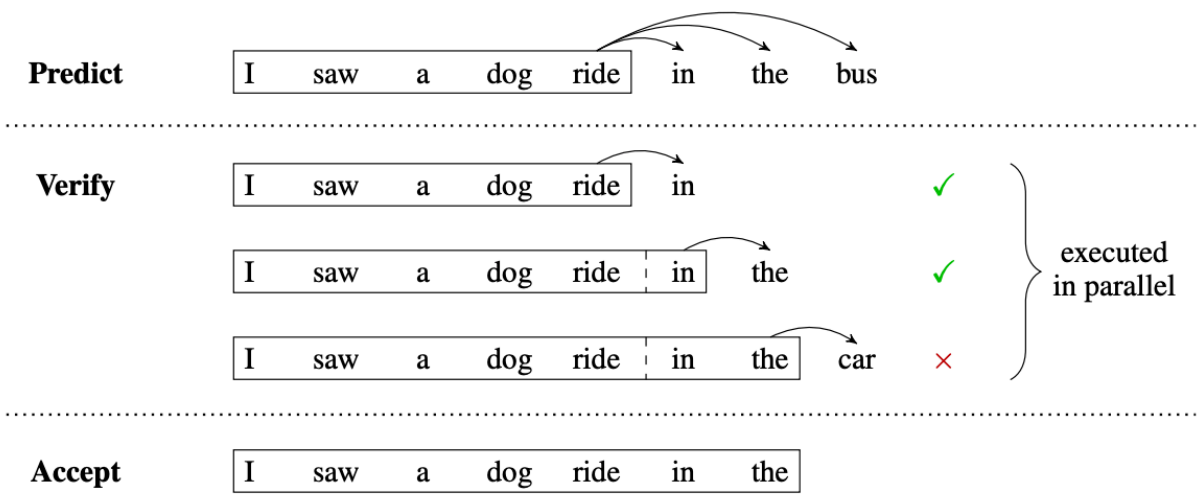
差分分析

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。

3.3 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。



3.3 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。

3.4 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	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

4.1 优化的趋势

领域	趋势	例子	PCC 的要求
内存管理	更精细的内存管理，更高的复用度	Paging Token-Level Optimization	**S**
传输	更精细的传输，使得资源能够按需到达	Data Duplication Prefetching/Pulling PD Disaggregation	**T**
调度	根据场景进行优化	Request-level Predictions Machine-Level Scheduling Global profiling	**STP**
并行	更高的并行度	Pipeline Parallelism Tensor Parallelism Sequence Parallelism Speculative Inference	**S**

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

4.2 论文

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	T																					
	Pulling	SET												✓									
	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														✓							✓

Miscellaneous

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

4.3 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	T					✗								
	Pulling	SET													
	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					✗		✗						

4.3 工业界的推理系统

vLLM 版本的路线图包括以下功能更新:

版本	日前	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

Thanks

Outline

5 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

5 附录

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).

5 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.

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.

5 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.

5 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.