Private Cloud Computing

Jiale Guan

2024-09-12

Outline

Outline

1.1 Taxonomy

Requirements	Threat	Guarantees
Stateless computation	Trace of data after processing e.g. Logging, debugging	(Purpose) Only use user data to perform requested operations (Transient) Delete the data after fulfilling the request (Scope) Not available to even Apple staff
Enforceable guarantees	Technical enforceability e.g. External TLS-terminating load balancer	(Hardware) Secure Enclave, Secure Boot (System) Signed System Volume, Swift on Server (Software) Code Signing, Sandboxing
No privileged runtime access	Privileged interfaces e.g. 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
Non-targetability	Targeted attack e.g. Steer request to compromised nodes	(Hardware) Hardened supply chain (Scheduler) Requests cannot be user/content-specific routed (Anonymity) OHTTP Relay, RSA Blind Signature (Scope) No system-wide encryption
Verifiable transparency	Uninspected code	Every production build of PCC publicly available

1.2 Requirements

Stateless computation

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.

Enforceable guarantees

Security and privacy guarantees are strongest when they are entirely technically enforceable, which means it must be possible to **constrain and analyze all the components** that critically contribute to the guarantees of the overall Private Cloud Compute system. To use our example from earlier, it's very difficult to reason about what a TLS-terminating load balancer may do with user data during a debugging session. Therefore, PCC must not depend on such external components for its core security and privacy guarantees. Similarly, operational requirements such as collecting server metrics and error logs must be supported with mechanisms that do not undermine privacy protections.

No privileged runtime access

Private Cloud Compute must not contain privileged interfaces that would enable Apple's site reliability staff to bypass PCC privacy guarantees, even when working to resolve an outage or other severe incident. This also means that PCC must not support a mechanism by which the privileged access envelope could be enlarged at runtime, such as by loading additional software.

Non-targetability

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.

Verifiable transparency

Security researchers need to be able to verify, with a high degree of confidence, that our privacy and security guarantees for Private Cloud Compute match our public promises. We already have an earlier requirement for our guarantees to be enforceable. Hypothetically, then, if security researchers had sufficient access to the system, they would be able to verify the guarantees. But this last requirement, verifiable transparency, goes one step further and does away with the hypothetical: security researchers must be able to verify the security and privacy guarantees of Private Cloud Compute, and they must be able to verify that the software that's running in the PCC production environment is the same as the software they inspected when verifying the guarantees.

Outline

LLM Inference

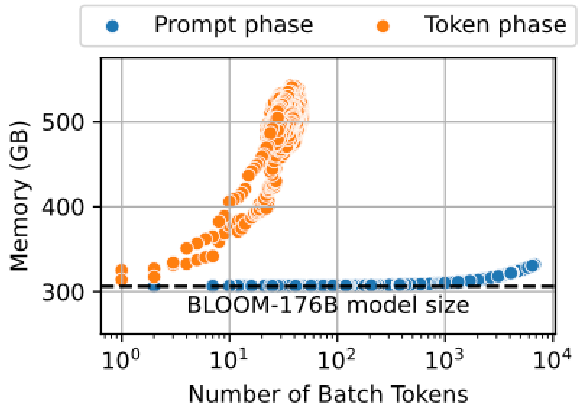
Most of the popular decoder-only LLMs (GPT-3, for example) are pretrained on the causal modeling objective, essentially as next-word predictors. These LLMs take a series of tokens as inputs, and generate subsequent tokens autoregressively until they meet a stopping criteria.

2.2 Prefill Phase: Processing the input

In the prefill phase, the LLM processes the input tokens to compute the intermediate states (keys and values), which are used to generate the “first” new token. Each new token depends on all the previous tokens, but because the full extent of the input is known, at a high level this is a matrix-matrix operation that’s **highly parallelized**. It effectively **saturates GPU utilization**.

2.3 Decode Phase: Generating the output

In the decode phase, the LLM generates output tokens autoregressively one at a time, until a stopping criteria is met. Each sequential output token needs to know all the previous iterations' output states (keys and values). This is like a matrix-vector operation that underutilizes the GPU compute ability compared to the prefill phase. The speed at which the data (weights, keys, values, activations) is **transferred to the GPU from memory** dominates the latency, not how fast the computation actually happens. In other words, this is a **memory-bound operation**.



Prefill (Prompt) Phase

The graph indicates that most of the memory usage in this phase may come from loading and initializing the model parameters, and the number of input tokens has little impact on memory usage.

Decode (Token) Phase

In contrast, memory usage during the decode phase increases, as the model needs to retain the context of all previously generated tokens to infer the next token. The memory requirement increases significantly with the number of generated tokens.

Outline

3.1 Concurrency

Batch Processing

Batch: A group of requests that are processed together.

Continous Batch: A batch that is continuously processed, leveraging the opportunity by batching new requests once some old requests are finished

Parallel Processing

Pipeline Parallelism

PP involves sharding the model (vertically) into chunks, where each chunk comprises a subset of layers that is executed on a separate device.

Tensor Parallelism

TP involves sharding the model (horizontally) into chunks, where each chunk comprises a subset of the model's parameters.

Sequence Parallelism

SP involves sharding the input sequence into chunks, where each chunk is processed by a separate device.

Speculative Inference[*]

Standard inference processes

Sequence generation is strictly sequential. Each token must be generated based on the previously generated token, which leads to high latency, especially for long-sequence tasks.

Speculative inference processes

- **Predict multiple tokens ahead**: When generating the first token, the model simultaneously makes speculative predictions about the next several tokens.
- **Parallel processing**: These speculative predictions allow the model to process multiple possible outcomes in parallel, speeding up the inference.
- **Validate predicted paths**: If the speculative predictions are correct, the model can continue with these results, avoiding the need to recalculate. If the predictions are incorrect, the model adjusts and corrects the path.

[*] Blockwise Parallel Decoding for Deep Autoregressive Models

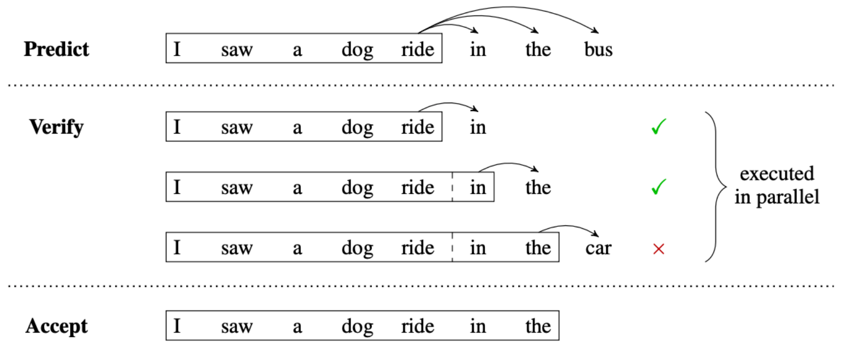
Speculative Inference (cont.)

Algorithm* is as follows:

- p is the smaller draft model, q is the larger target model.

Algorithm 2 Speculative Sampling (SpS) with Auto-Regressive Target and Draft Models

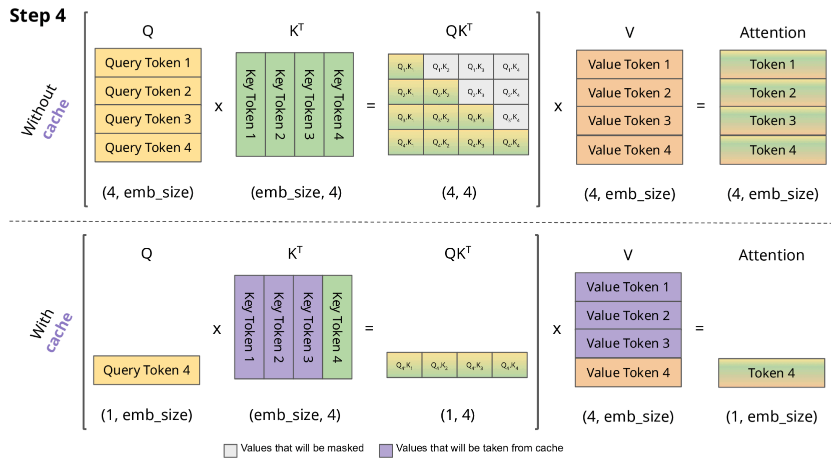
```
Given lookahead $K$ and minimum target sequence length $T$.
Given auto-regressive target model $q(.|.)$, and auto-regressive draft model $p(.|.)$, initial prompt sequence $x_0, \dots, x_t$.
Initialise $n \leftarrow t$.
while $n < T$ do
  for $t = 1 : K$ do
    Sample draft auto-regressively $\tilde{x}_t \sim p(x|, x_1, \dots, x_n, \tilde{x}_1, \dots, \tilde{x}_{t-1})$
  end for
  In parallel, compute $K + 1$ sets of logits from drafts $\tilde{x}_1, \dots, \tilde{x}_K$ :
        $q(x|, x_1, \dots, x_n),\ q(x|, x_1, \dots, x_n, \tilde{x}_1),\ \dots,\ q(x|, x_1, \dots, x_n, \tilde{x}_1, \dots, \tilde{x}_K)$
  for $t = 1 : K$ do
    Sample $r \sim U[0, 1]$ from a uniform distribution.
    if $r < \min\left(1, \frac{q(x|x_1,\dots,x_{n+t-1})}{p(x|x_1,\dots,x_{n+t-1})}\right)$, then
      Set $x_{n+t} \leftarrow \tilde{x}_t$ and $n \leftarrow n + 1$.
    else
      sample $x_{n+t} \sim (q(x|x_1, \dots, x_{n+t-1}) - p(x|x_1, \dots, x_{n+t-1}))_+$ and exit for loop.
    end if
  end for
  If all tokens $x_{n+1}, \dots, x_{n+K}$ are accepted, sample extra token $x_{n+K+1} \sim q(x|, x_1, \dots, x_n, x_{n+K})$ and set $n \leftarrow n + 1$.
end while
```



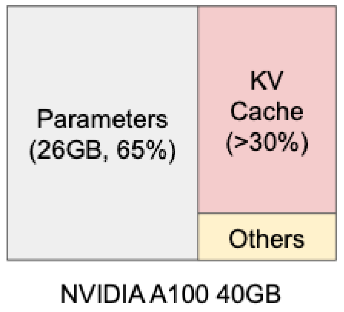
* Accelerating Large Language Model Decoding with Speculative Sampling, 2023

3.2 Resource Reuse

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.

Paged Attention[*]

Paged Attention is a technique that divides the attention matrix into smaller pages, which are processed sequentially. This allows the model to process large attention matrices that do not fit in GPU memory.

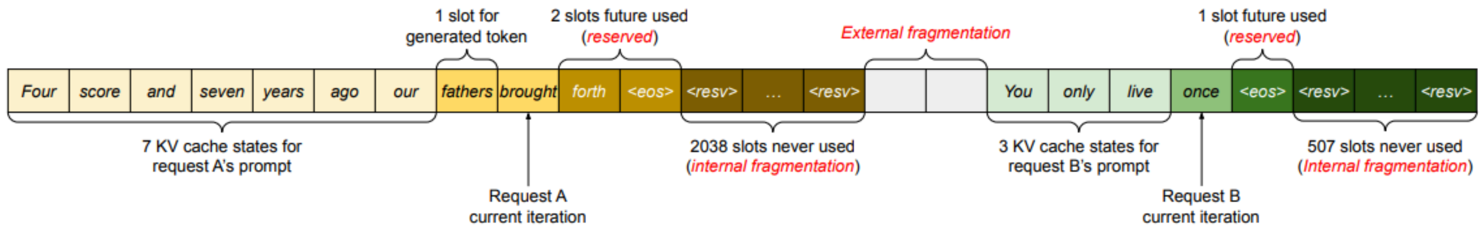

1 slot for generated token
2 slots future used (reserved)
External fragmentation
1 slot future used (reserved)
Four
score
and
seven
years
ago
our
fathers
brought
forth
<eos>
<resv>
...
<resv>
You
only
live
once
<eos>
<resv>
...
<resv>
7 KV cache states for request A's prompt
2038 slots never used (internal fragmentation)
3 KV cache states for request B's prompt
507 slots never used (Internal fragmentation)
Request A current iteration
Request B current iteration

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

Paged Attention (cont.)

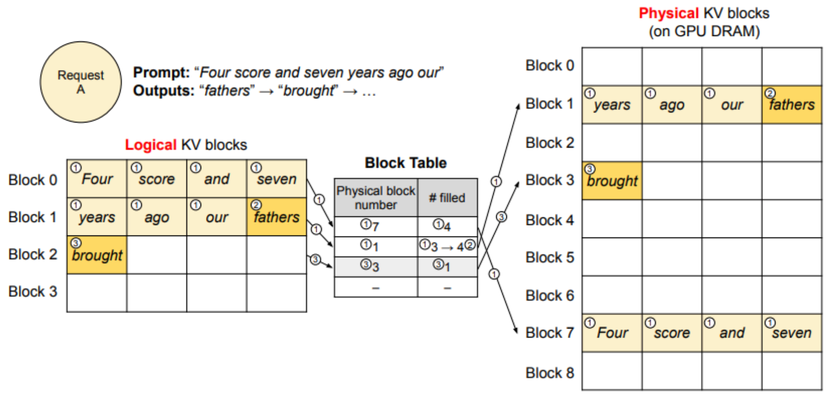

Request A
Prompt: "Four score and seven years ago our"
Outputs: "fathers" → "brought" → …
Logical KV blocks
Block Table
Physical block number
# filled
Physical KV blocks (on GPU DRAM)

Figure 6. Block table translation in vLLM.

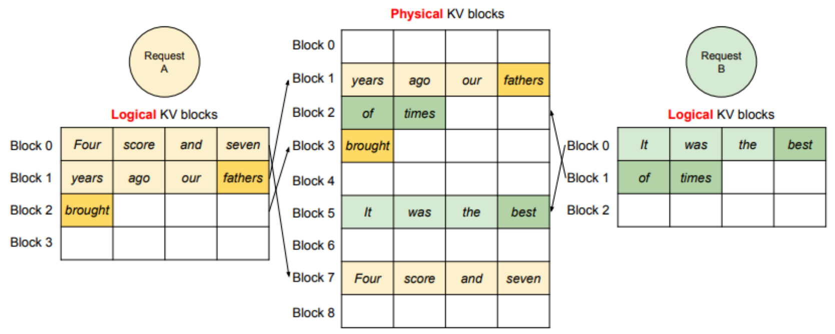


Figure 7. Storing the KV cache of two requests at the same time in vLLM.

Group-Query Attention[*]

- **Standard Attention**: Compute attention for each query separately. Complexity is $O(n^2)$.
- **Multi-Query Attention**: Reuse the same attention matrix for multiple queries. Queries are similar enough to share the same attention distribution.
- **Group-Query Attention**: Divide queries into groups and compute attention for each group separately.

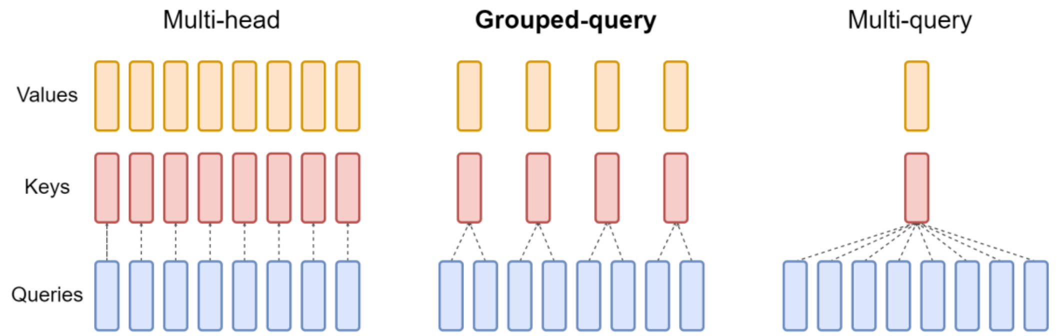


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

[*] GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

Prefix Caching

Prefix Caching is a technique that caches the intermediate states of the model during the prefill phase. These states are then reused during the decode phase to speed up inference.

Flash Attention*

GPU: One kind of computation done on the input data at a time in sequence

Fusing: Fusing multiple layers together during the actual computation can enable minimizing the data access by GPUs.

FlashAttention uses **tiling** to fully compute and write out a small part of the final matrix at once

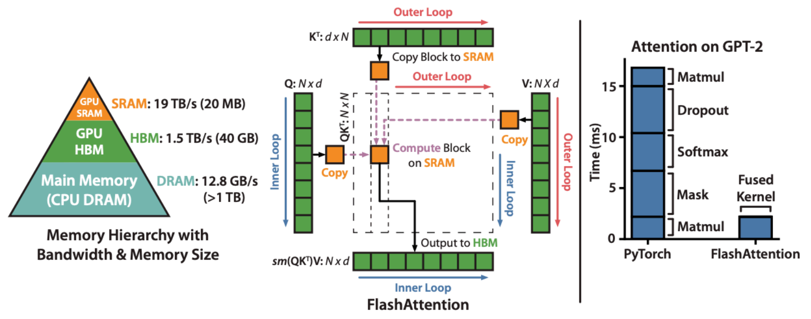


Figure 1: **Left:** FlashAttention uses tiling to prevent materialization of the large $N \times N$ attention matrix (dotted box) on (relatively) slow GPU HBM. In the outer loop (red arrows), FlashAttention loops through blocks of the **K** and **V** matrices and loads them to fast on-chip SRAM. In each block, FlashAttention loops over blocks of **Q** matrix (blue arrows), loading them to SRAM, and writing the output of the attention computation back to HBM. **Right:** Speedup over the PyTorch implementation of attention on GPT-2. FlashAttention does not read and write the large $N \times N$ attention matrix to HBM, resulting in an 7.6× speedup on the attention computation.

* FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

KV Cache Offloading

The KV Cache Offloading technique moves the KV cache from the GPU to the CPU to free up GPU memory for other tasks.

3.3 Miscellaneous

Compression

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.

Cross Region & Cloud

Cross Region: Distributing the model across multiple regions to reduce latency and improve availability.

Cloud: Using cloud services to offload the model computation to reduce the load on the on-premises servers.

SkyPilot

3.4 Summary

Category	Optimization	Compute	Memory	Transmission	Throughput	TTFT	TBT
Batch	Iteration-Level Batch	+	+	+	+	-	-
	Chunked Prefill	+	+		+	+	+
	Prepack Prefill	+			+	-	
Parallelism	Pipeline Parallelism	+		-	+	-	?
	Tensor Parallelism	+		-	+	-	+
	Sequence Parallelism	+		-	+	+	?
	Speculative Inference	+	-	-	+	+	+
Memory	Paging		+		+		
	Disk Offloading		+	-	+		
	Prefix Caching		+		+		
	Multi-Query Attention	+	+		+	+	+
	Group-Query Attention	+	+		+	+	+
Tranmission	Duplication	+	-	+	+	+	+
	Pulling	+	-	+	+	+	+
	Request Migration	+	+	-	+	+	+
	Disaggregated Arch	+	+	-	+	-	-

Outline

4.1 Academic Systems

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

Category	Optimization	Threat	FT 22	Orca 22	vLLM 23	FlexGen 23	FastServe 23	Some 23	Sarathi 23	SGLang 23	Preble 24	Lookahead 24	REST 24	SpecInfer 24	Medusa 24	DistServe 24	Splitwise 24	LoongServe 24	AttentionStore 24	TetriInfer 24	InfiniteLLM 24
Batch	Iteration-Level Batch			Initial	✓		✓		✓							✓				✓	
	Chunked Prefill								Initial		✓									✓	
	Prepack Prefill													✓		✓				✓	
Parallelism	Speculation	S								✓		✓	✓	✓	✓						
	Prompt-Based Speculation	S										✓									
	Context-Based Speculation	S											✓								
	Tensor Parallelism		✓																		
	SafeTensors																				
	Sequence Parallelism																	✓			
Memory	Paging				Initial				✓	✓										✓	
	Disk Offloading	SE				✓	✓												✓		
	Prefix Caching	SE								✓	✓										
	Multi-Query Attention																				
	Group-Query Attention	T																			
Tranmission	Duplication	T																			
	Pulling	SET														✓					
	Request Migration																	✓			
	Disaggregated Arch															✓	✓			✓	

Scheduling	Priority-Based	**T**		✓	✓	✓	✓					✓	
	Request-Level Prediction	**T**		✓				✓				✓	
	Machine-level Scheduler	**ET**		✓			✓			✓	✓	✓	✓
	Instance Flip									✓		✓	
	Global Profiling	**P**	✓						✓	✓			
Verification	Open Source	**V**										✓	

TorchServe - PyTorch

Preble - more read

AlpaServe

Pollux

Attmemo:Accelerating transformers with memoization on big memory systems

Ring Attention

Virtual Token Counter

FastGen

Refer to DistServe Related Work

4.2 Industrial Systems

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

Category	Optimization	Threat	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
Batch	Iteration-Level Batch		✓		✓		✓	✓	✓	✓	✓	✓	✓	✓	
	Chunked Prefill		✓				✓	✓							
	Prepack Prefill														
Parallelism	Speculation	**S**	✓		✓	✓			✓	✓				✓	✓
	Medusa									✓				✓	
	Tensor Parallelism									✓					✓
	SafeTensors									✓					
	Sequence Parallelism														
Memory	Paging		✓			✓	✓	✓		✓				✓	
	Token Attention			✓											
	Disk Offloading	**SE**				✓	✓		✓						✓
	Prefix Caching	**S**	✓						✓						
	Multi-Query Attention								✓						
	Group-Query Attention	**T**							✓						
Tranmission	Duplication	**T**					✓								
	Pulling	**SET**													
	Request Migration														
	Disaggregated Arch		✓				✓								✓

Scheduling	Priority-Based	**T**		✓	✓	✓	
	Request-Level Prediction	**T**	✓	✓			
	Machine-level Scheduler	**ET**		✓	✓		
	Instance Flip						
	Global Profiling	**P**			✓		
Verification	Open Source	**V**			✓		✓

Other Optimizations

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: 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: enable efficient execution of large language models across a wide range of hardware platforms, including mobile devices, edge devices, and even web browsers.

4.3 Threats

Paging & Offloading

Definition:

- Paging: 使用类似于虚拟内存的机制，将模型参数分页存储在磁盘上，根据需要加载到内存中。
- Offloading: 将模型参数从 GPU 内存中移动到 CPU 内存或磁盘中，以释放 GPU 内存供其他模型使用。

Threats:

- 分页处理过程中可能会产生包含敏感信息的日志，这些日志如果没有妥善管理，可能会泄露隐私数据。
- 分页数据可能会被意外持久化到不安全的存储介质中，从而暴露隐私数据。

Duplication & Pulling

Definition:

- Duplication: 在不同的节点之间复制模型参数，以便在多个节点上并行执行推理任务。
- Pulling: 从远程节点拉取模型参数，以便在本地节点上执行推理任务。

Threats:

- 模型参数的复制和拉取过程中可能会泄露隐私数据。
- 模型参数的复制和拉取过程中可能会定向到恶意节点，从而导致隐私数据泄露。如果其中任何一个节点被攻破，攻击者可能获得整个模型的敏感信息。
- 拉取模型参数可能导致数据不同步，尤其在多次拉取操作之间，可能出现数据不一致的情况，影响模型的准确性和隐私保护。

Priority-based Scheduler & Local Scheduler & Instance Flip

Definition:

- Priority-based Scheduler: 根据任务的优先级调度任务，以确保高优先级任务能够及时完成。
- Local Scheduler: 在本地节点上调度任务，以减少任务调度的延迟。

Threats:

（优先级调度）

- 可能通过观察任务的优先级来推断任务的重要性和敏感性，从而有针对性地进行攻击。
- 在任务调度过程中，任务的调度信息（如任务类型、数据类型等）可能被泄露，导致隐私数据暴露。’

（本地调度）

- 在本地节点上调度任务时，所有任务和数据都集中在本地节点，如果本地节点被攻破，所有数据和任务信息都可能被泄露。
- 本地节点可能会缓存大量的任务数据，如果这些缓存数据未妥善处理，可能会导致隐私泄露。
- 为了减少调度延迟，可能会牺牲一些数据同步和一致性机制，导致数据不一致。

（节点翻转）

- 攻击者可能修改恶意节点的数据，来让恶意节点被选中执行任务，从而获取敏感信息。
- 攻击者可能通过控制节点翻转的时机，来获取敏感信息-。

Disaggregated Architecture & Online/Offline Profiling

Definition:

- Disaggregated Architecture: 将 Prefill 和 Decode 的过程通过实例(instance)分离，以提高资源利用率和灵活性。
- Online/Offline Profiling: 在线/离线性能分析，以优化模型推理性能。

Threats:

- 在进行用户画像时，会收集和存储大量的用户数据，包括在线行为数据和离线数据，这些数据一旦被泄露，可能对用户隐私造成严重威胁。

Iteration-Level Batch & Chunked Prefill & Prepack Prefill

Definition:

- Iteration-Level Batch: 在迭代级别上进行批处理，以提高模型推理性能。
- Chunked Prefill: 将 Prefill 过程分块，以减少 Prefill 的延迟。
- Prepack Prefill: 预先打包 Prefill 数据，以减少 Prefill 的延迟。

Threats:

- N/A.

Thanks

Outline

5.1 References

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

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

Tools

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