Private Cloud Compute

Jiale Guan

2024-09-18

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.

1.2 Requirements

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.

1.2 Requirements

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.

1.2 Requirements

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.

1.2 Requirements

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

2.1 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.

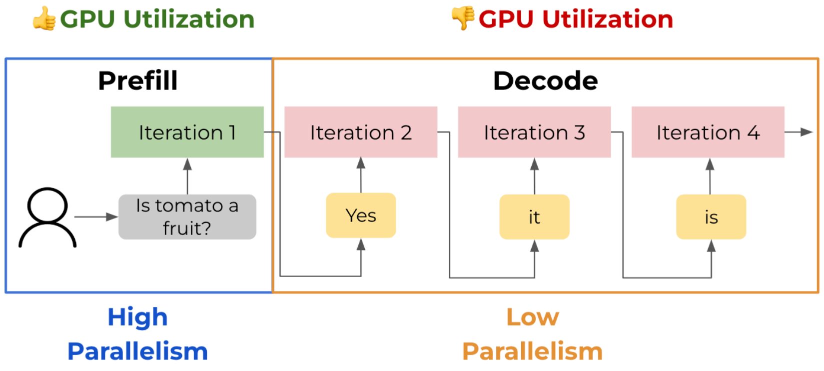
2.2 Phases

Prefill: 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**.

Decode: 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**.



2.3 Challenges

Workload Heterogeneity

Universality and application diversity lead to heterogeneity of the inference requests, in terms of input lengths, output lengths, expected latencies, etc

Execution Unpredictability

Unknown a priori how many tokens will be generated before the stopping criteria is met. As such, the execution time and the resource demand of a request are both unpredictable.

Multi-Tenant and Dynamic Environment

The system must scale to support multiple users and adapt to the dynamic nature of the environment.

2.3 Challenges

Queuing Delays

The system must handle queuing delays, which can be caused by the system being overloaded or by the system waiting for external resources.

Preemptions

The system must handle preemption, which can be caused by the system being overloaded or by the system waiting for external resources.

Interference

Interference between requests can lead to performance degradation.

Outline

3.1 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

Algorithm 1 LLM serving with Continuous batching

Initialize current batch $B \leftarrow \emptyset$, waiting queue $Q \leftarrow \emptyset$
▷ `with monitoring stream:`
while True **do**
    if new request r arrived **then**
        $Q \leftarrow Q + r$
▷ `with execution stream:`
while True **do**
    if can_add_new_request() **then**
        $B_{new} \leftarrow$ **select_new_requests(Q)**
        prefill(B_{new})
        $B \leftarrow B + B_{new}$
    decode(B)
    $B \leftarrow$ filter_finished_requests(B)

3.2 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.

3.3 Speculative Inference

Standard inference

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*

- **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

3.3 Speculative Inference

Algorithm* is as follows:

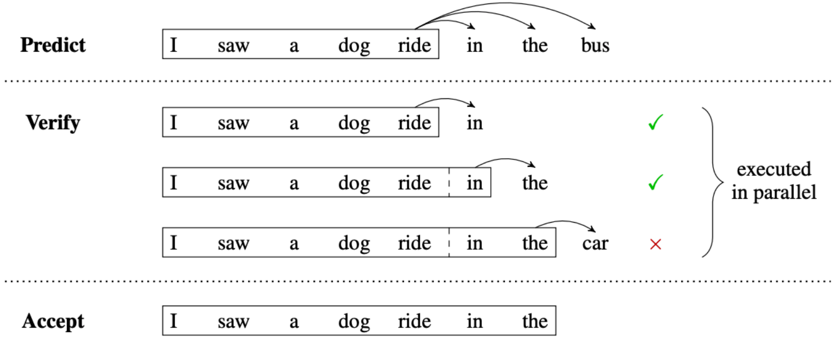
- p is the smaller draft model, q is the larger target model.

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 $x_0, \ldots, x_t$.
Initialise $n \leftarrow t$.
while $n < T$ **do**
  for $t = 1 : K$ **do**
    Sample draft auto-regressively $\tilde{x}_t \sim p(x|, x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_{t-1})$
  end for
  In parallel, compute $K + 1$ sets of logits from drafts $\tilde{x}_1, \ldots, \tilde{x}_K$:

$$q(x|, x_1, \ldots, x_n),\; q(x|, x_1, \ldots, x_n, \tilde{x}_1),\; \ldots,\; q(x|, x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_K)$$

  for $t = 1 : K$ **do**
    Sample $r \sim U[0, 1]$ from a uniform distribution.
    if $r < \min\left(1, \frac{q(x|x_1,\ldots,x_{n+t-1})}{p(x|x_1,\ldots,x_{n+t-1})}\right)$, **then**
      Set $x_{n+t} \leftarrow \tilde{x}_t$ and $n \leftarrow n + 1$.
    else
      sample $x_{n+t} \sim (q(x|x_1, \ldots, x_{n+t-1}) - p(x|x_1, \ldots, x_{n+t-1}))_+$ and exit for loop.
    end if
  end for
  If all tokens $x_{n+1}, \ldots, x_{n+K}$ are accepted, sample extra token $x_{n+K+1} \sim q(x|, x_1, \ldots, x_n, x_{n+K})$ and set $n \leftarrow n + 1$.
end while



* Accelerating Large Language Model Decoding with Speculative Sampling, 2023

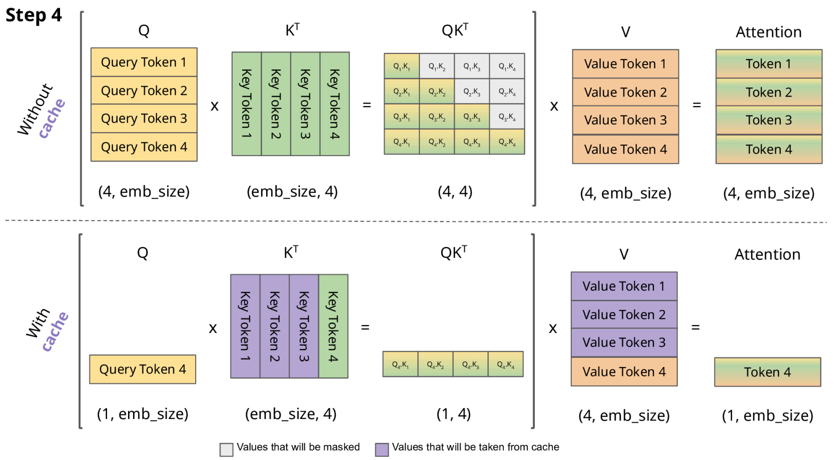
3.3 Speculative Inference

Medusa

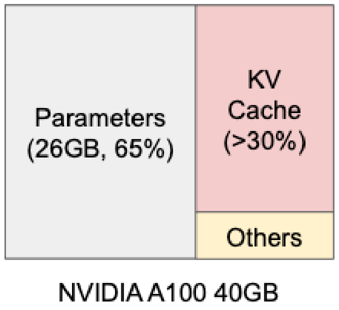
Medusa is a system that uses **speculative inference** to generate multiple tokens in parallel. It uses a **speculative model** to predict multiple tokens ahead and then validates the predicted paths to avoid redundant calculations.

3.4 Memory

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.

3.4 Memory

Paged Attention

Paged Attention* is a technique that divides the attention matrix into smaller pages. This approach provides a near-perfect solution for mitigating fragmentation and hence, PagedAttention has become the de facto standard for dynamic memory allocation in LLM serving systems.

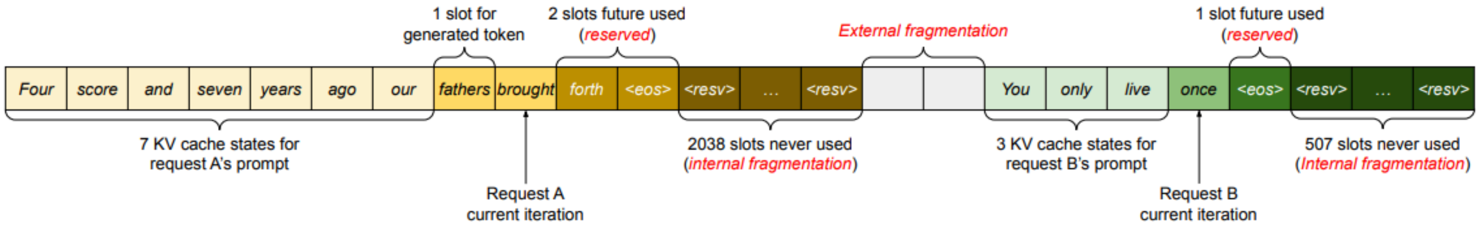


Figure 3. KV cache memory management in existing systems. Three types of memory wastes – reserved, internal fragmentation, and external fragmentation – exist that prevent other requests from fitting into the memory. The token in each memory slot represents its KV cache. Note the same tokens can have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

* Efficient Memory Management for Large Language Model Serving with PagedAttention

3.4 Memory

Paged Attention (cont.)

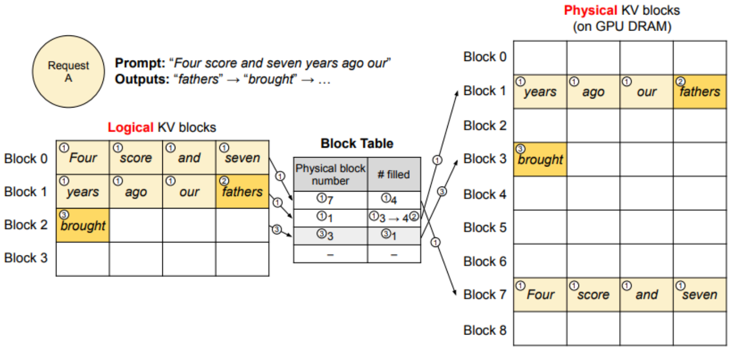


Figure 6. Block table translation in vLLM.

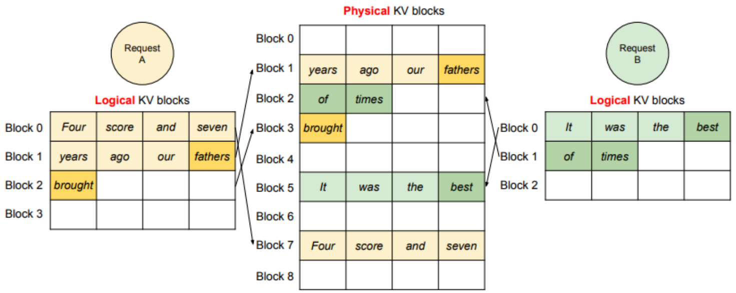


Figure 7. Storing the KV cache of two requests at the same time in vLLM.

Pitfalls*

- Requires re-writing the attention kernel.
- Adds software complexity and redundancy (CPU code), can degrade throughput by 11%.
- Introduces performance overhead. 20-26% slower than original FasterTransformer kernel.

* vAttention: Dynamic Memory Management for Serving LLMs without PagedAttention. 2024

3.4 Memory

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.

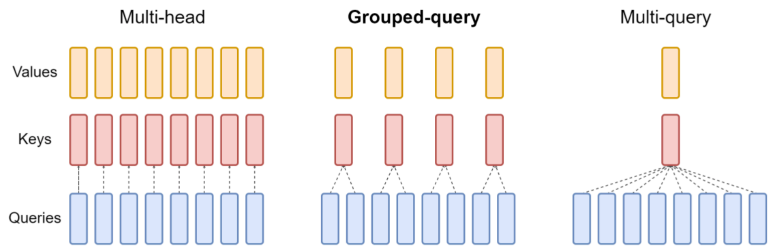


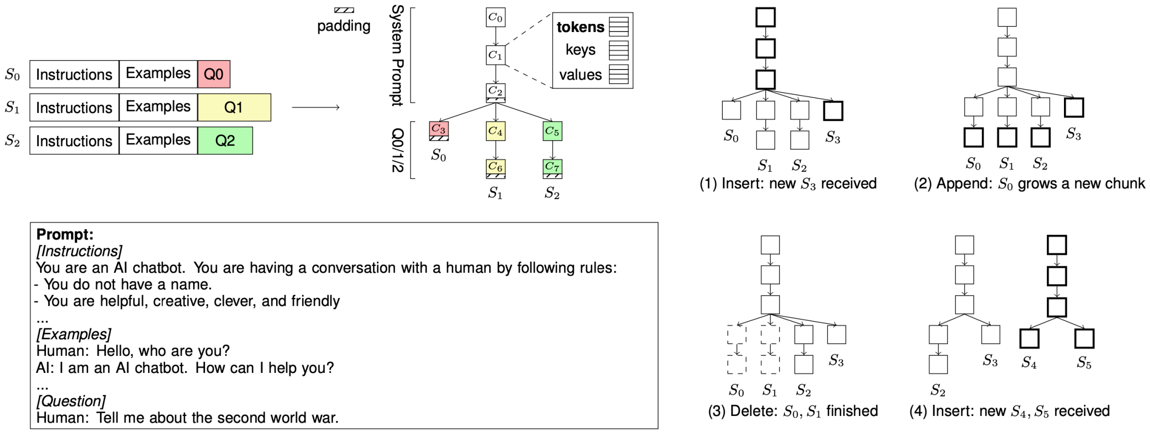
Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between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

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

3.4 Memory

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.



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

3.4 Memory

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

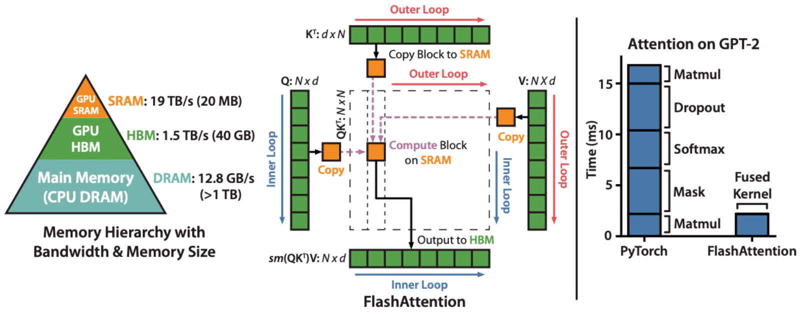


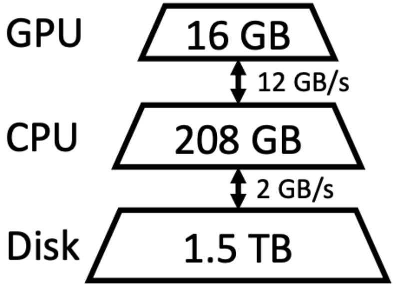
Figure 1: **Left:** FlashAttention uses tiling to prevent materialization of the large $N \times N$ attention matrix (dotted box) on (relatively) slow GPU HBM. In the outer loop (red arrows), FlashAttention loops through blocks of the **K** and **V** matrices and loads them to fast on-chip SRAM. In each 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

3.4 Memory

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.



3.5 Miscellaneous

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.

3.6 Summary

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.

3.6 Summary

Category	Optimization	GPU Resources			Optimization Goal		
		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	-	+		+		
	Prefix Caching		+		+		
	Disk Offloading		+	-	+		
	Multi-Query Attention		+		+	+	+
	Group-Query Attention		+		+	+	+
Tranmission	Duplication	+	-	+	+	+	+
	Pulling	+	-	+	+	+	+
	Request Migration	+	+	-	+	+	+
	Disaggregated Arch	+	+	+	+	-	-

Outline

4.1 Academic Systems

Category	Optimization	Threat	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
Batch	Iteration-Level Batch		Initial		✓	✓	✓					✓		✓		✓					
	Chunked Prefill											Initial				✓					✓
	Prepack Prefill													✓		✓					
Parallelism	Speculation	S				✓		✓		✓	✓				✓						
	Context-Based Speculation	S						✓													
	Prompt-Based Speculation	S									✓										
	Tensor Parallelism					✓															
	SafeTensors																				
	Sequence Parallelism																	✓			
Memory	Paging						Initial			✓		✓				✓					
	Prefix Caching	SE								✓											✓
	Disk Offloading	SE		✓													✓				
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														✓					

Miscellaneous

Title	Keywords	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	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

4.2 Industrial Systems

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	✓		✓	✓			✓	✓				✓	✓
	Tensor Parallelism									✓					✓
	SafeTensors									✓					
	Sequence Parallelism														
Memory	Paging		✓			✓	✓	✓		✓				✓	
	Token Attention			✓											
	Prefix Caching	S	✓						✓						
	Disk Offloading	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					✓		✓						

Thanks

Outline

5.1 References

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

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

5.2 Utilities

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