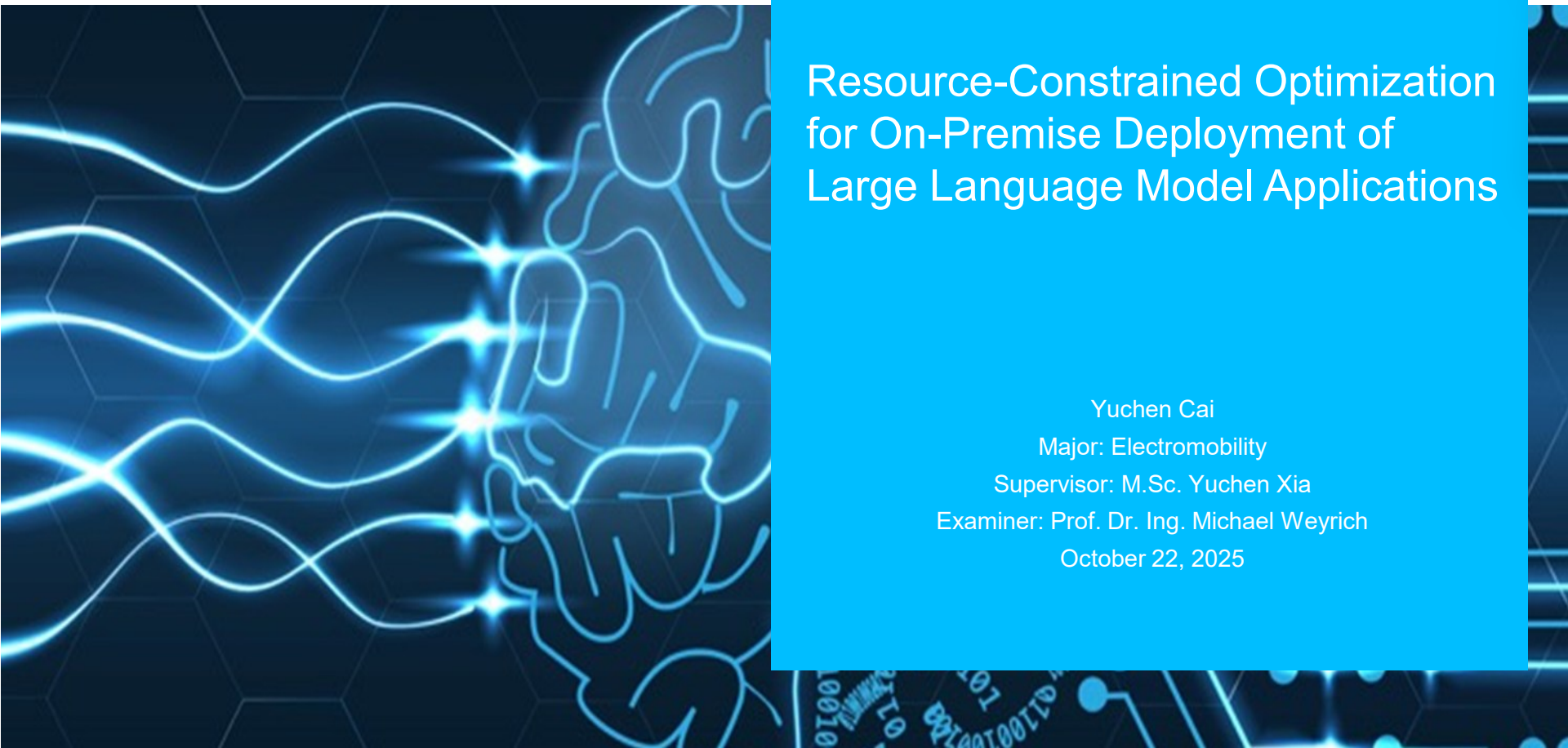
University of Stuttgart
Institute of Industrial Automation
and Software Engineering



Resource-Constrained Optimization for On-Premise Deployment of Large Language Model Applications

Yuchen Cai
Major: Electromobility
Supervisor: M.Sc. Yuchen Xia
Examiner: Prof. Dr. Ing. Michael Weyrich
October 22, 2025

Agenda

1. Introduction
2. Background
3. Important Metrics
4. Experimentation Setup
5. Optimization methods
6. Test and evaluation
7. Conclusion and outlook

Introduction

Introduction

The LLM Revolution & The Deployment Choice

Large Language Models (LLMs) are transforming industries — enabling automation, decision support, and knowledge discovery.

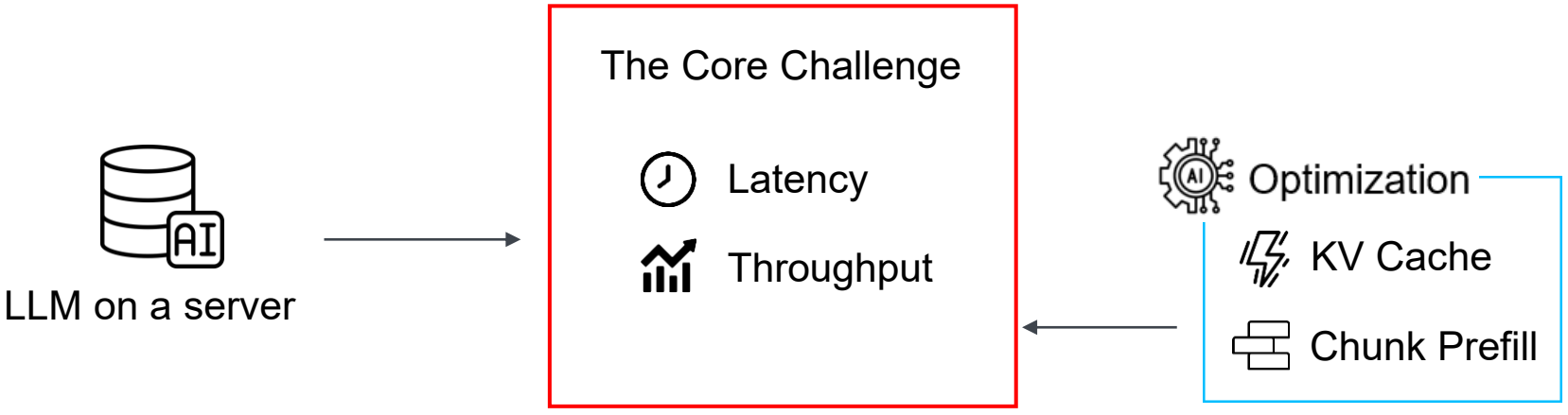
Two main deployment approaches: 1. Cloud-Based

2. On-Premise

Aspect	Cloud-Based	On-Premise
Latency	**Internet-dependent**	**Requires local GPUs and servers**
Throughput	**Elastic / Scalable on-demand**	**Limited by local hardware**
Data Security	Depends on provider's policy	Full control; no external data transfer
Infrastructure	No local investment	Requires local GPUs and servers
Scalability	Virtually unlimited	Limited by GPU and VRAM
Customization	Restricted by APIs	Full model access and tuning
Cost	High CAPEX, low OPEX	Pay-as-you-go (high OPEX)

Introduction

Challenge and Optimization



Background

Background

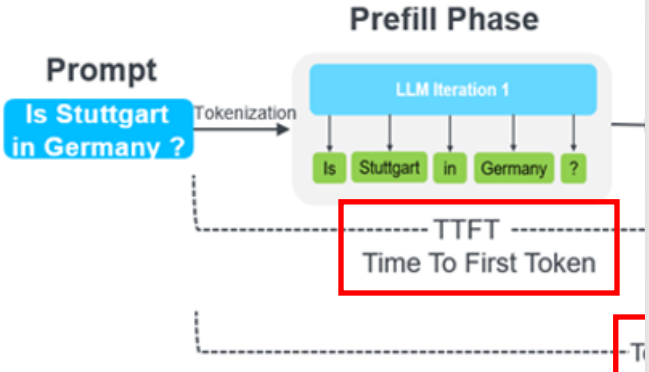
Classification of Existing Optimization Methods

Level	Technique	Advantage	Disadvantage
Inference	**KV Caching** [1]	**Avoids recomputation**	**High VRAM**
	Chunk Prefill [2]	**Reduces prefill load**	**Adds scheduling cost**
	Group-Query Attention [3]	Faster decoding	Slight accuracy loss
Model	Quantization [4]	Smaller model	Accuracy loss
	Pruning [5]	Less compute	Retraining needed
System	Flash Attention[6]	Speeds up attention, saves memory	Limited to supported GPUs
	Paged Attention [7]	Dynamic cache management	Complex implementation

Basics

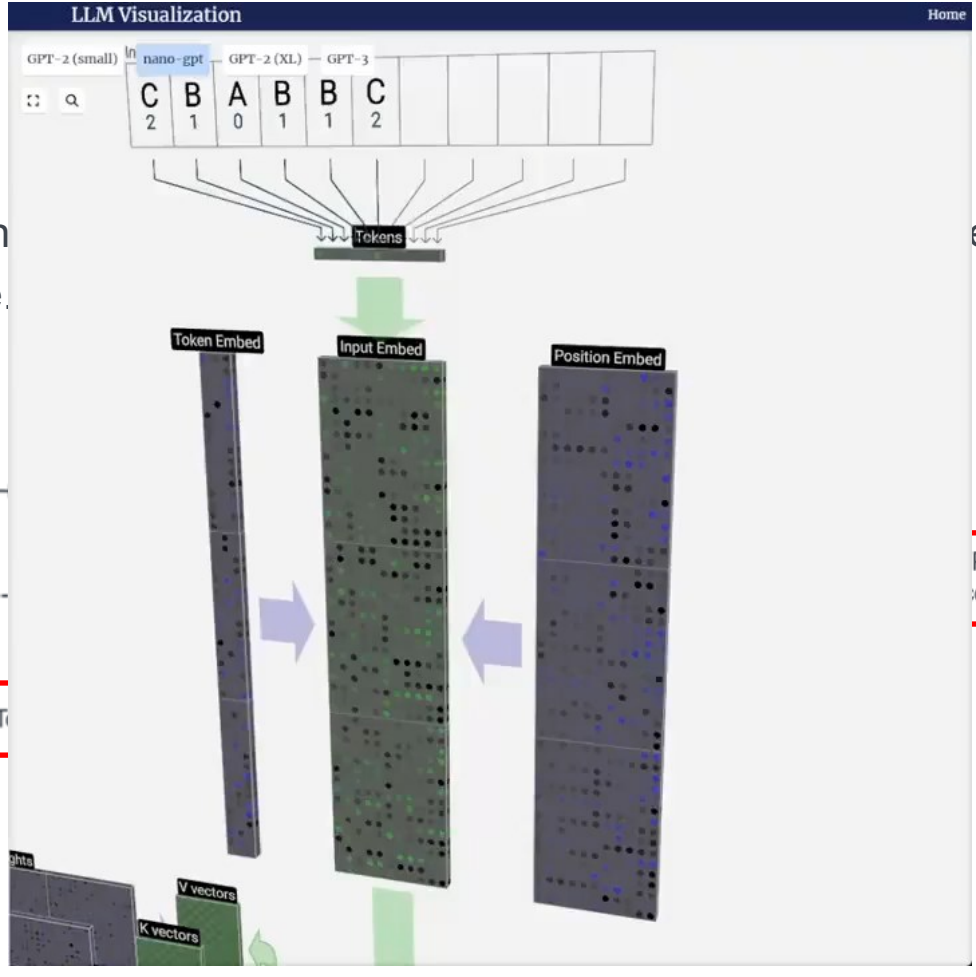
LLM Inference Process

The inference process of Large Lan e
Prefill Stage and the Decode Stage.



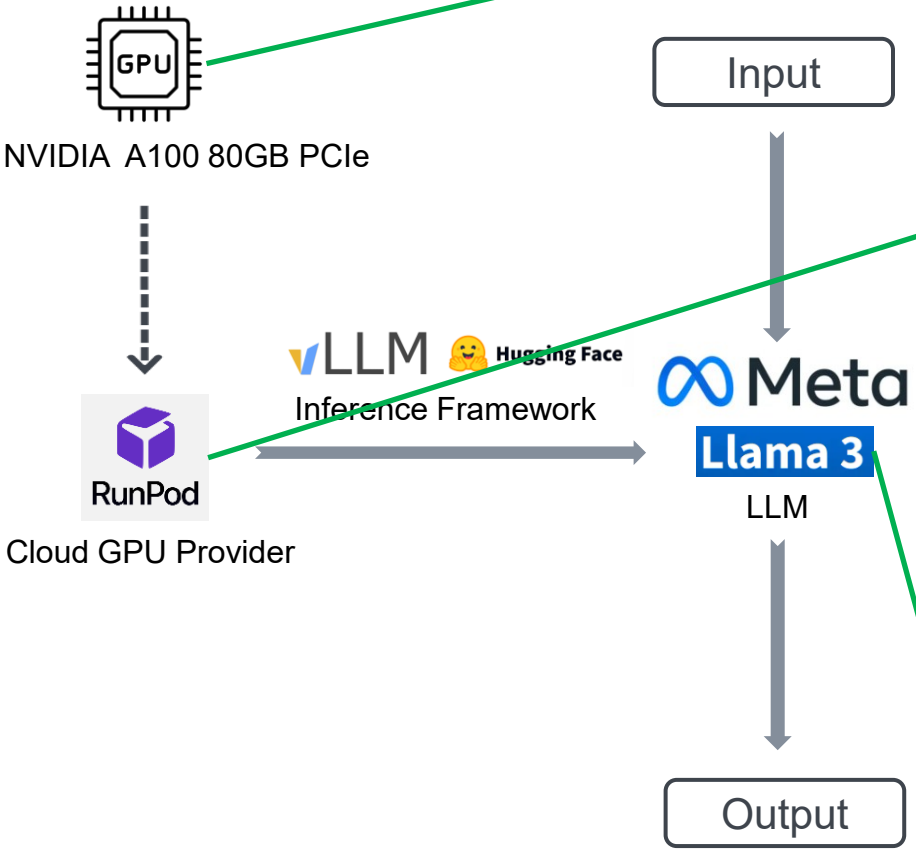
PS)
ond

[9] LLM Inference



[8] LLM Inference Process visualization

Deployment of LLM



GPU	NIVIDIA A100 80GB PCIe
FP16 Tensor	312 TFLOPS
Memory	80GB HBM2e
Memory Bandwidth	1935GB/s

Cloud Platform	RunPod
CUDA Version	12.8
Operating System	Ubuntu 24.04 LTS
PyTorch Version	2.8.0
Container Disk	100 GB
Cost	$1.64 / hr

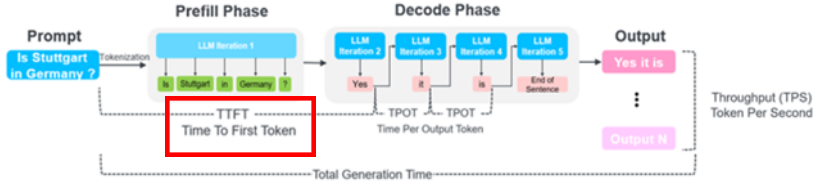
Model	Meta Llama3-8B
Model parameter size	8B
Architecture	Decoder-only Transformer
Attention Heads	32 (GQA enabled)
Hidden Size	4096
Context Length	8019
Precision	FP16

Important Metrics

- Latency
- Throughput

Important Metrics: Latency

Prefill: Time To First Token(TTFT)



Definition: The amount of time between sending an input prompt and receiving the first output token.

Symbol	Description	Value
L	Number of Transformer Layers	32
d_{model}	Hidden Size	4096
B	Batch Size	1
k_{FLOPS}	The per-token FLOPs coefficient.	26
$FLOPS_{peak}$	Peak theoretical performance of the hardware	312TFLOPS
η_{MFU}	Model FLOPS Utilization	95.57%
s_{in}	Input Sequence Length	513
c_0	Constant Overhead	14.83ms

$$TTFT = \frac{L \cdot k_{FLOPS} \cdot B \cdot d_{model}^2}{FLOPS_{peak} \cdot \eta_{MFU}} \cdot s_{in} + \frac{L \cdot 4 \cdot B \cdot d_{model}}{FLOPS_{peak} \cdot \eta_{MFU}} \cdot s_{in}^2 + c_0$$

$$TTFT_{Theoretical} = 54.13\ ms$$

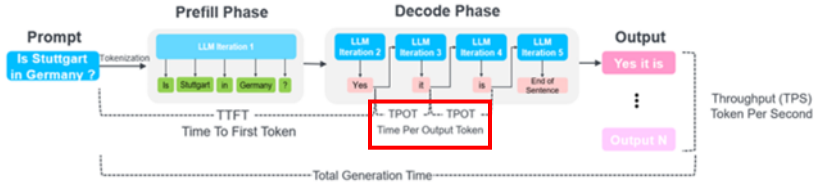
$$TTFT_{Experimental} = 54.81\ ms$$

$$\text{Error} = 1.24\%$$

Important Metrics: Latency

Decode: Time Per Output Token(TPOT)

Definition: The average time required to generate each output token after the first token.



Symbol	Description	Value
L	Number of Transformer Layers	32
d_{model}	Hidden Size	4096
B	Batch Size	1
b_{para}	Bytes per parameter, FP16	2
b_{kv}	Bytes per KV Cache element, FP16	2
d_{kv}	Total dimension of the K/V heads	1024
BW_{peak}	Peak theoretical memory bandwidth	1935 GB/s
η_{BW}	Memory Bandwidth Utilization	57.95%
s_{ctx}	Current context length	1025

$$TPOT = \frac{12 \cdot L \cdot d_{model}^2 \cdot b_{para}}{BW_{peak} \cdot \eta_{BW}} + \frac{2 \cdot L \cdot B \cdot d_{kv} \cdot b_{kv}}{BW_{peak} \cdot \eta_{BW}} \cdot s_{ctx}$$

$$TPOT_{Theoretical} = 11.36\ ms/token$$

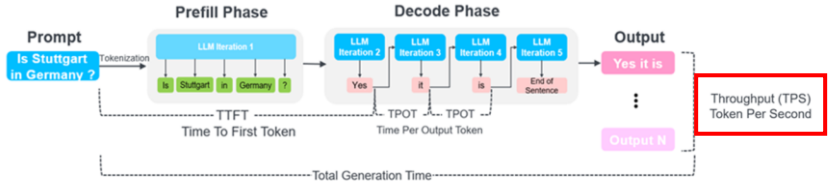
$$TPOT_{Experimental} = 16.16\ ms/token$$

$$Error = 30\%$$

Important Metrics: Throughput

Tokens Per Second(TPS)

Definition: The number of output tokens generated by the model per second.



Symbol	Description	Value
L	Number of Transformer Layers	32
d_{model}	Hidden Size	4096
B	Batch Size	1
b_{para}	Bytes per parameter, FP16	2
b_{kv}	Bytes per KV Cache element, FP16	2
d_{kv}	Total dimension of the K/V heads	1024
BW_{peak}	Peak theoretical memory bandwidth	1935 GB/s
η_{BW}	Memory Bandwidth Utilization	57.95%
s_{ctx}	Current context length	1025

$$TPS_{Theoretical} = 81.6\ tokens/s$$

$$TPS_{Experimental} = 57.89\ tokens/s$$

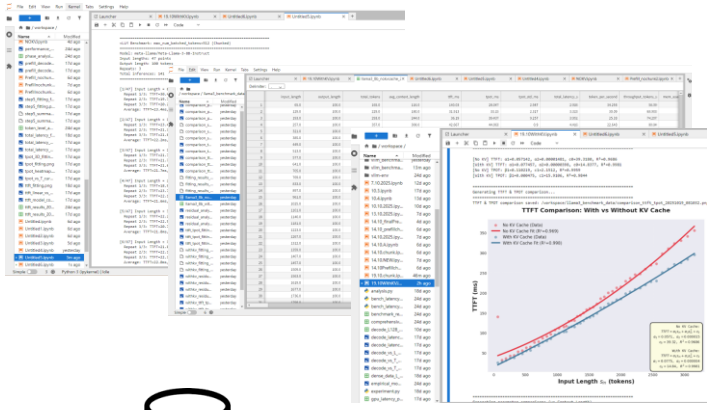
$$\text{Error} = 45\%$$

$$TPS = \frac{1}{TPOT} = \frac{BW_{peak} \cdot \eta_{BW}}{12 \cdot L \cdot d_{model}^2 \cdot b_{para} + 2 \cdot L \cdot B \cdot d_{model} \cdot b_{kv} \cdot s_{ctx}}$$

Experimentation Setup

Experimentation Setup

Deployment and Measurement Workflow

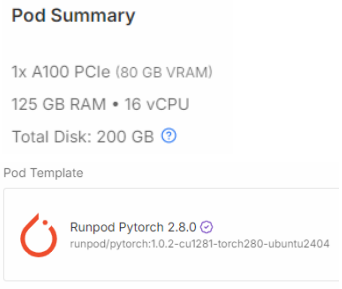


GPU

NVIDIA A100

RunPod

Cloud GPU Provider



vLLM

Inference Framework

Hugging Face

Input

Meta

Llama 3

LLM

Output

Metrics Input Monitor

Metrics Output Monitor

Fitting Formula
(to predict the throughput and latency)

Optimization methods

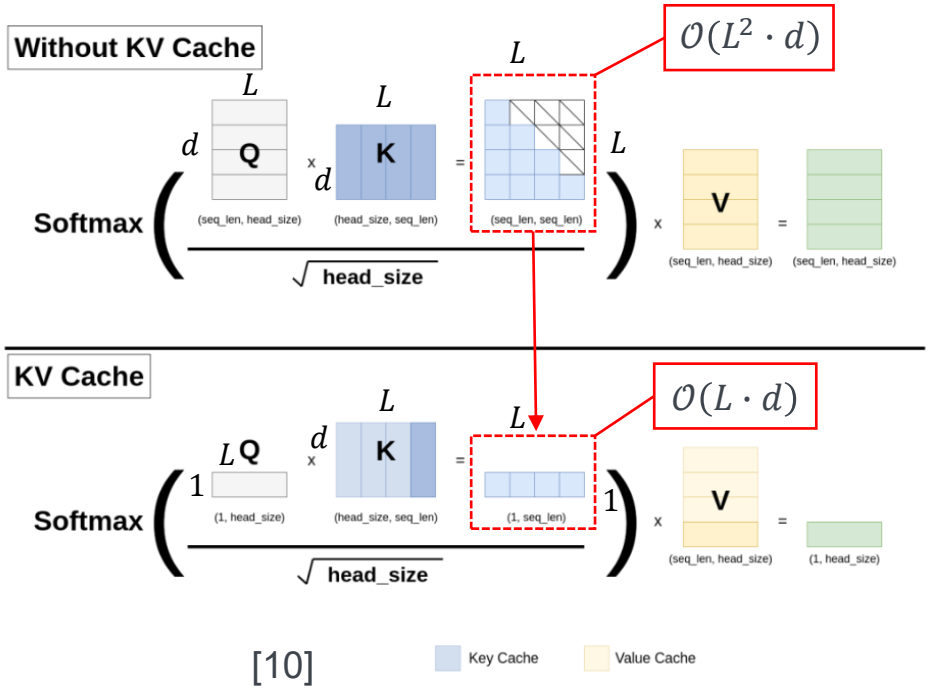
- KV Cache
- Chunk Prefill

Optimization Method 1

KV Cache

Definition: KV Cache is a mechanism that stores previously computed **Key** and **Value** tensors during decoding to reuse past attention, thereby reducing per-step computation complexity and speeding up generation.

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$



[10]

Symbol	Description
Q	Query
K	Key
V	Vaule
L	Sequence length
d	Head size

Computational Complexity

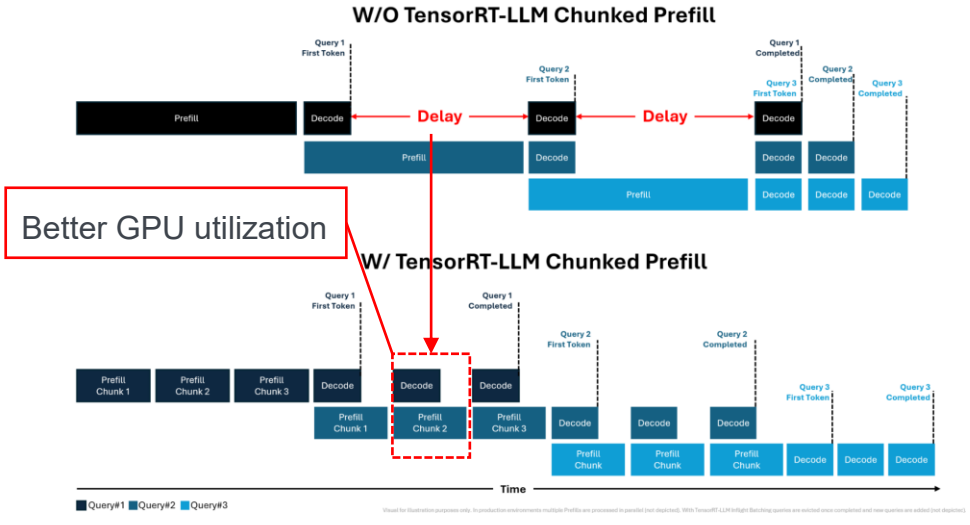
$\mathcal{O}(L^2 \cdot d) \longrightarrow \mathcal{O}(L \cdot d)$

Optimization Method 2

Chunk Prefill

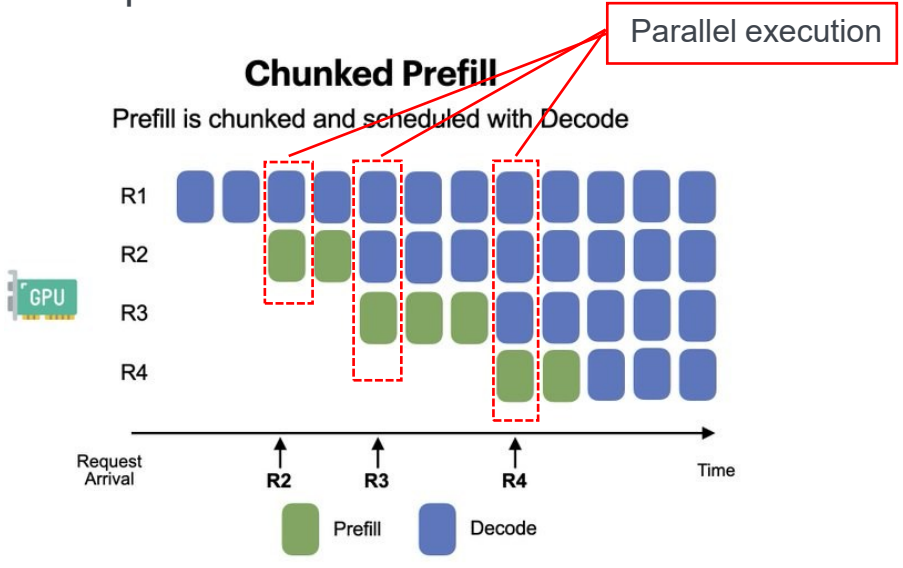
Definition: Chunk Prefill is a technique that splits large input prompts into smaller chunks and batches them together with decode requests to reduce latency and improve throughput.

Efficient Scheduling



[11]

Computation Parallelization



[12]

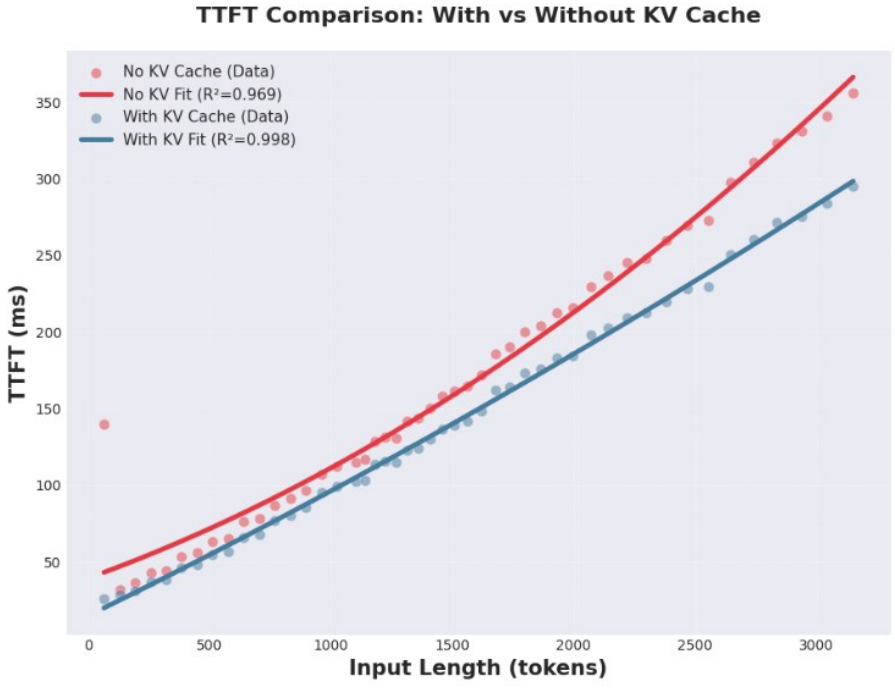
Test and evaluation

- Without / With KV Cache

Optimization Method 1: KV Cache

Prefill: Time To First Token Fitted Formula

$$TTFT = \alpha_1 \cdot s_{in} + \alpha_2 \cdot s_{in}^2 + c_0$$



Without KV cache

$$TTFT_{NoKV} = 5.714 \cdot 10^{-2} \cdot s_{in} + 1.481 \cdot 10^{-5} \cdot s_{in}^2 + 39.381$$

$$R_{NoKV}^2 = 0.9686$$

With KV cache

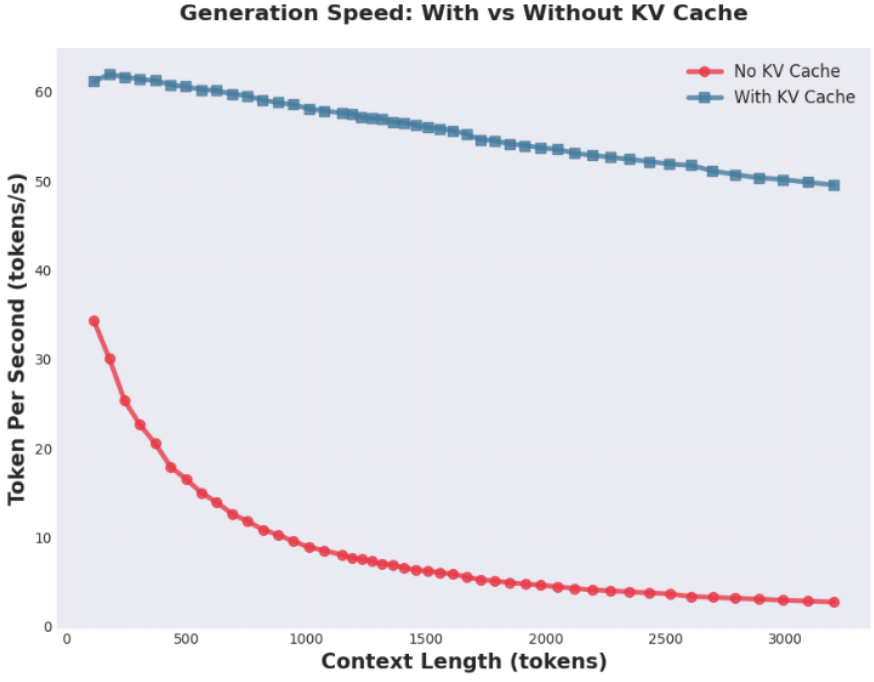
$$TTFT_{KV} = 7.745 \cdot 10^{-2} \cdot s_{in} + 3.98 \cdot 10^{-5} \cdot s_{in}^2 + 14.837$$

$$R_{KV}^2 = 0.9686$$

For Latency, Lower is Better!
Enabling KV Cache reduces the Time To First Token (TTFT) by approximately **25%–35%**

Optimization Method 1: KV Cache

Tokens Per Second(TPS)



	Without KV Cache	With KV Cache	Speedup
Average TPS	9.02 tokens/s	56.03 tokens/s	6.2×
s_{ctx}>2900 tokens	2.84 tokens/s	49.86 tokens/s	17.6×

For Throughput, Higher is Better!

.

Conclusion: KV Cache improves generation throughput by **500–1600%**, significantly accelerating inferencing.

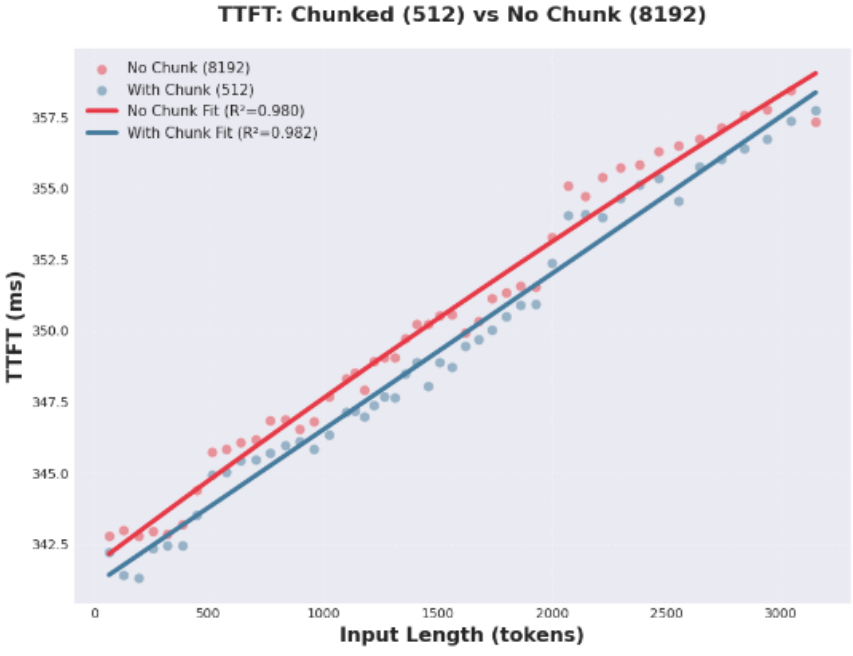
Test and evaluation

- Without / With Chunk Prefill

Optimization Method 2: Chunk Prefill

Prefill: Time To First Token Fitted Formula

$$TTFT = \alpha_1 \cdot s_{in} + \alpha_2 \cdot s_{in}^2 + c_0$$



Without Chunk Prefill

$$TTFT_{NoChunk} = 6.008 \cdot 10^{-3} \cdot s_{in} + 1.7 \cdot 10^{-7} \cdot s_{in}^2 + 341.408$$

$$R^2_{NoChunk} = 0.980$$

With Chunk Prefill

$$TTFT_{Chunk} = 5.435 \cdot 10^{-3} \cdot s_{in} + 2 \cdot 10^{-8} \cdot s_{in}^2 + 341.06$$

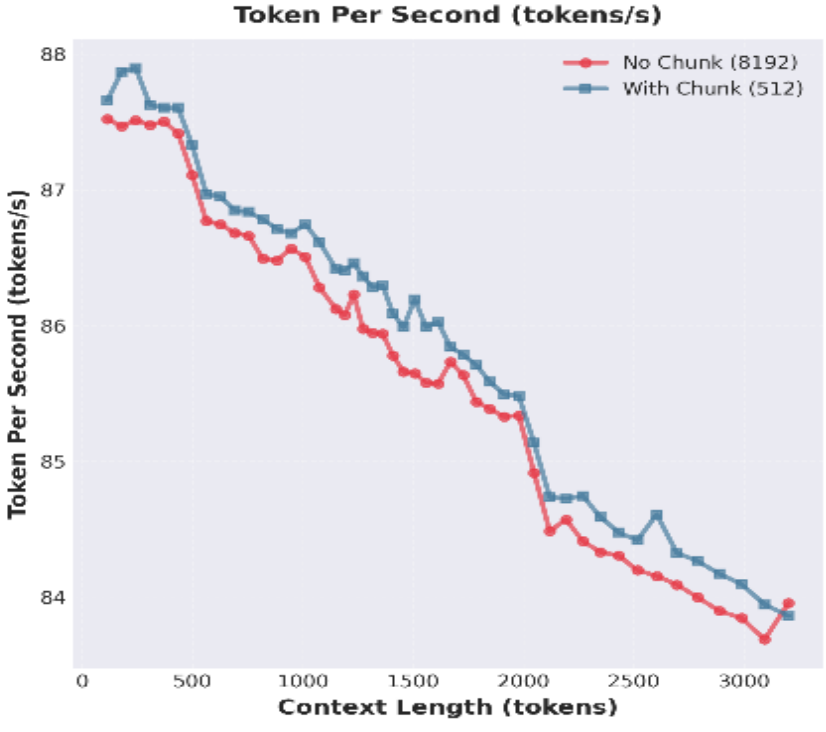
$$R^2_{Chunk} = 0.982$$

For Latency, Lower is Better!

Chunk Prefill reduces Time To First Token by **0.3–1%** for small-batch inference.

Optimization Method 2: Chunk Prefill

Tokens Per Second(TPS)



For Throughput, Higher is Better!

With Chunk Prefill improves throughput by about **1.5–2%**, meaning **higher TPS → faster token generation** during prefill.

Conclusion: Chunk Prefill marginally improve the LLM inference performance

Conclusion and outlook

Conclusion and Outlook

Conclusion

- KV Cache and Chunk Prefill effectively improve inference efficiency.
- System-level optimization is key for on-premise LLM deployment.

Outlook

- Further optimize resource scheduling and memory allocation for constrained environments.
- Develop adaptive inference pipelines to balance latency, throughput in real time.

University of Stuttgart
Institut of Industrial Automation
and Software Engineering

Thank you!

Yuchen Cai

e-mail st178192@stud.uni-stuttgart.de
phone +49 (0) 711 685-
fax +49 (0) 711 685-

University of Stuttgart
Institut of Industrial Automation and Software Engineering
Pfaffenwaldring 47, 70550 Stuttgart, Germany

References

[1] Shi, Luohe, et al. "Keep the cost down: A review on methods to optimize LLM's KV-cache consumption." arXiv preprint arXiv:2407.18003 (2024).

[2] Agrawal, Amey, et al. "Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills." arXiv preprint arXiv:2308.16369 (2023).

[3] Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebr'on, F., & Sanghai, S.K. (2023). GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. ArXiv, abs/2305.13245.

[4] Gong, Ruihao, et al. "A survey of low-bit large language models: Basics, systems, and algorithms." arXiv preprint arXiv:2409.16694 (2024).

[5] Kurtić, Eldar, Elias Frantar, and Dan Alistarh. "Ziplm: Inference-aware structured pruning of language models." Advances in Neural Information Processing Systems 36 (2023): 65597-65617.

[6] Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." Advances in neural information processing systems 35 (2022): 16344-16359.

[7] Kwon, Woosuk, et al. "Efficient memory management for large language model serving with pagedattention." Proceedings of the 29th symposium on operating systems principles. 2023.

[8] S. Park, S. Jeon, C. Lee, S. Jeon, B.-S. Kim, and J. Lee, “A Survey on Inference Engines for Large Language Models: Perspectives on Optimization and Efficiency,” arXiv preprint arXiv:2505.01658, May 2025.

[9] B. Bycroft, “LLM Visualization,” bbycroft.net, Aug. 03, 2025. [Online]. Available: https://bbycroft.net/llm.

References

[10] [1] U. Jamil, “PyTorch LLaMA Notes,” GitHub repository, Jun. 2024. [Online]. Available: https://github.com/hkproj/pytorch-llama-notes.

[11] Clay Atlas, “KV Cache: A Caching Mechanism To Accelerate Transformer Generation,” Clay-Atlas.com, Nov. 1, 2024. [Online]. Available: https://clay-atlas.com/us/blog/2024/11/01/en-transformer-kv-cache-accelerate/

[12] A. Elmeleegy, N. Comly, and S. Chetlur, “Streamlining AI Inference Performance and Deployment with NVIDIA TensorRT-LLM Chunked Prefill,” NVIDIA Developer Blog, Nov. 15, 2024. [Online]. Available: https://developer.nvidia.com/blog/streamlining-ai-inference-performance-and-deployment-with-nvidia-tensorrt-llm-chunked-prefill/

[13] Anyscale, “chunked prefill – Anyscale contribution to vLLM breaks prefill requests into multiple chunks and batch them with decoding requests,” X, May 12, 2025. [Online]. Available: https://x.com/anyscalecompute/status/1795485248563839074