UNIVERSITY OF CENTRAL FLORIDA

ALISA: Accelerating Large Language Model Inference via Sparsity-aware KV Caching

Youpeng Zhao, Di Wu, Jun Wang

Computer Systems and Data Science (CASS) Laboratory

University of Central Florida

ISCA 2024

Background

- Emergence of Large Language Models (LLMs)

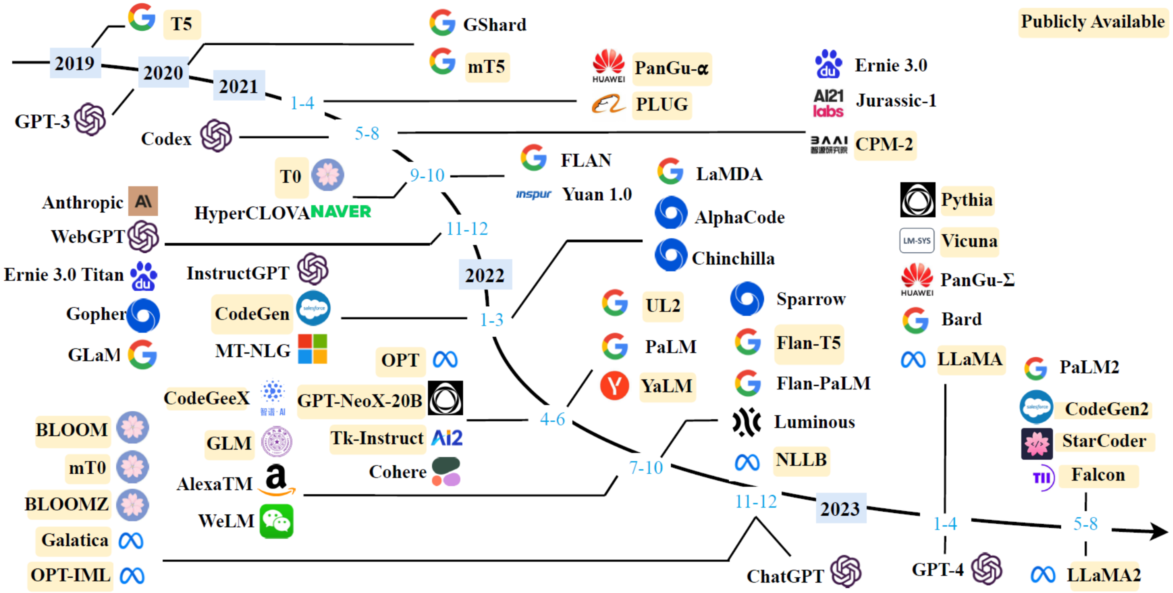
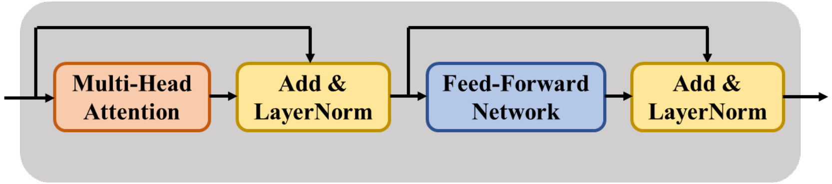


Fig. 1: A timeline of existing large language models (>10 B) in recent years [1].

- Inference serving accounts for most LLM-based application scenarios. Accelerating LLM inference has become an increasingly important research problem.

Background

- Transformer Architecture



$$AW(Q,K) = \sigma\left(\frac{QK^T}{\sqrt{d}}\right)$$

Quadratic Complexity

$$Attn(Q,K,V) = AW(Q,K) \cdot V$$

- LLM Inference with KV Caching

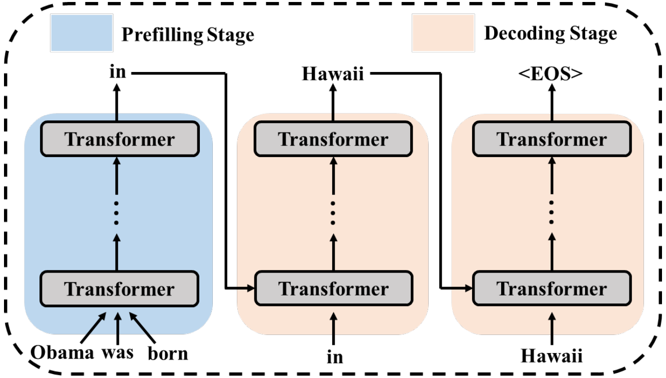


Fig. 2: Autoregressive inference of LLMs

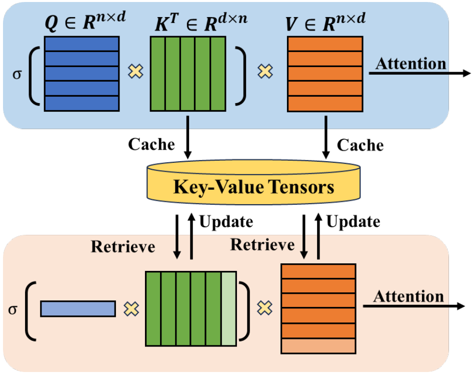


Fig. 3: KV Caching Mechanism [2].

Original Attention Computation $O(n^2)$

Attention Computation $O(n)$
+
KV Memory Access $O(n)$

Motivation

- KV caching increases memory overhead

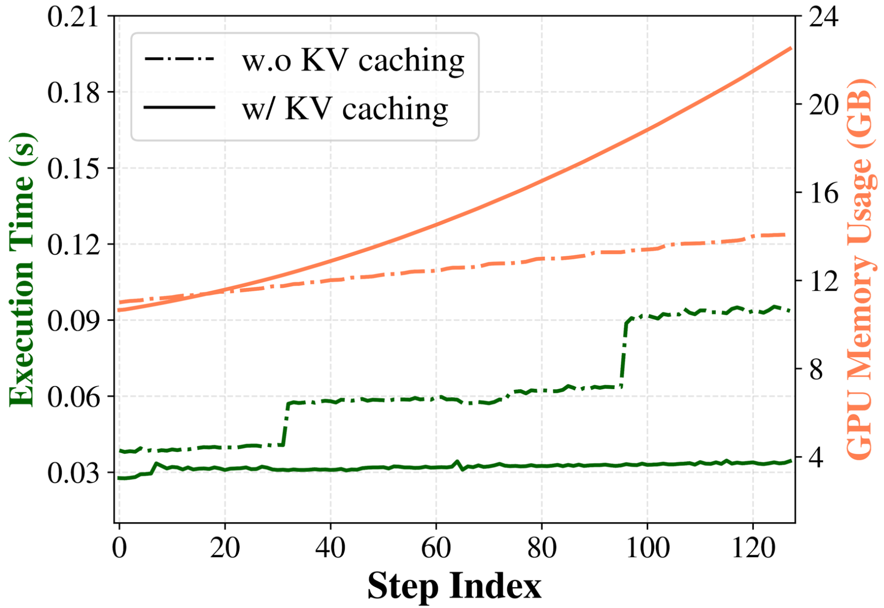
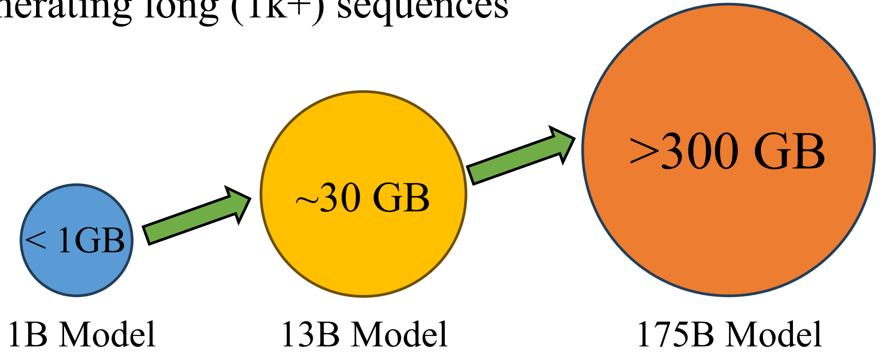


Fig. 4: Execution time and memory comparison for OPT-6.7B.

$$KV\ Tensor\ Size = B \cdot h \cdot n$$

- For small LLMs, the KV tensor size for one token is around dozens of MB; for larger models, the size can be as big as dozens of GB

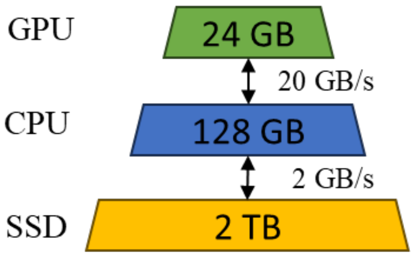
- The memory footprint of KV tensors can be a potential problem when serving large models or generating long (1k+) sequences



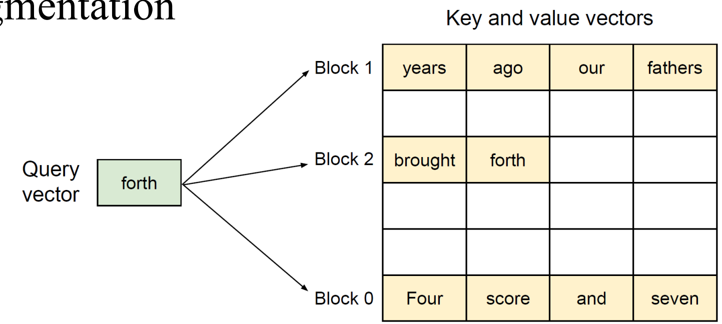
Motivation

- Optimizing KV Caching (Previous works)

- FlexGen [3]: Utilize CPU DRAM, and secondary storage (SSD) to hold the intermediate KV tensors and model weights



- vLLM [4]: Employ non-contiguous paged memory to store KV tensors at block-level, where each block contains a fixed group of tokens to reduce memory fragmentation



- Other Works for accelerating LLM inference:

- Attention computation acceleration [5,6], pruning [7,8] (Algorithm)
- Hardware acceleration [9,10] (Accelerator co-design)

} **Cannot scale to large LLMs**

- Quantization [11,12] **Focus on model weight reduction**

Main Problem

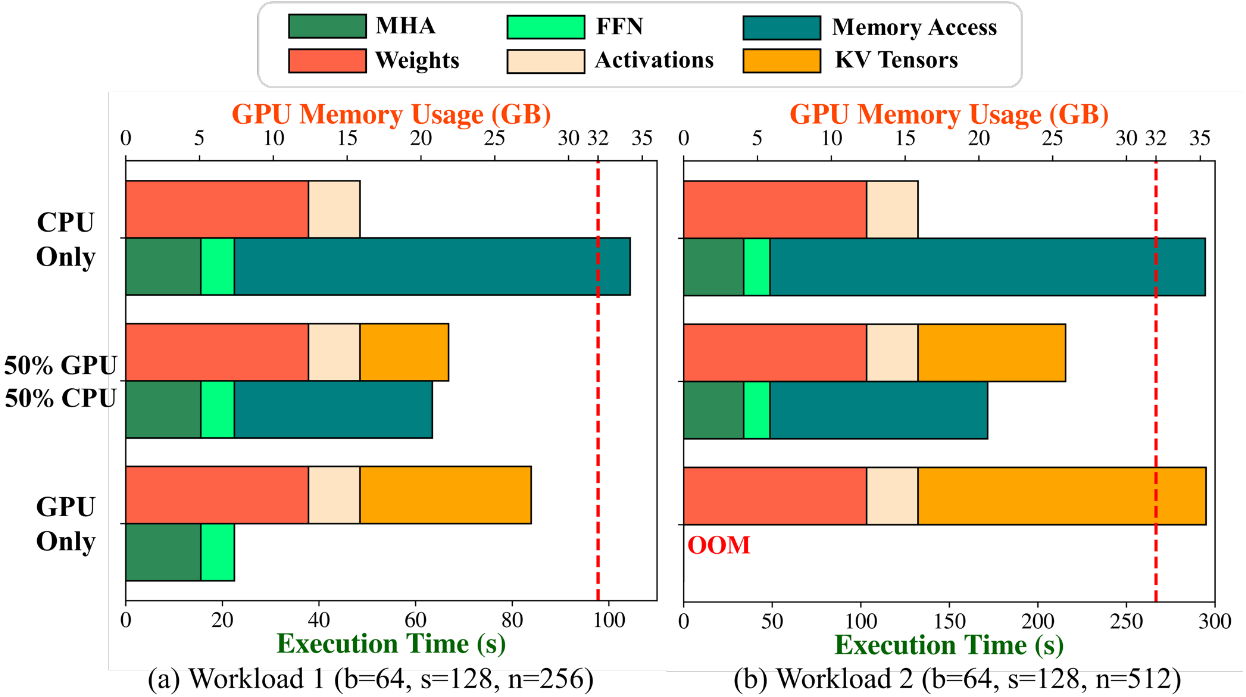


Fig. 5: Execution breakdown of OPT-6.7B in a single NVIDIA V100 (32GB) GPU.

Frequent offloading/reloading incurs significant I/O latency, creating new bottleneck for LLM inference

Proposal

How do we innovate KV caching to alleviate I/O bottleneck of LLM inference?

Key Observation: Not all tokens are created equal!

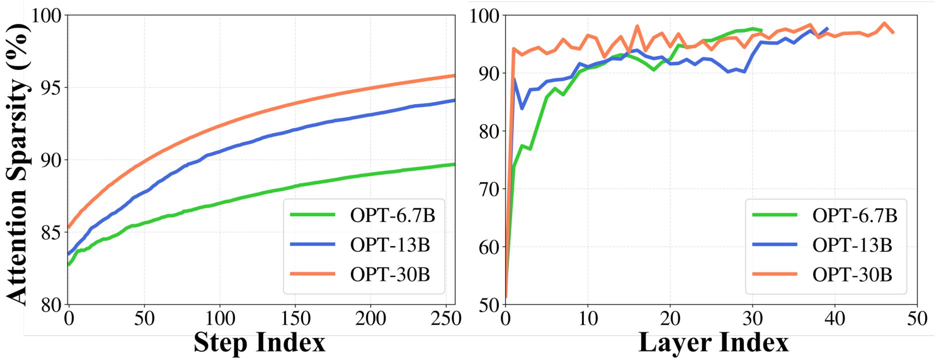
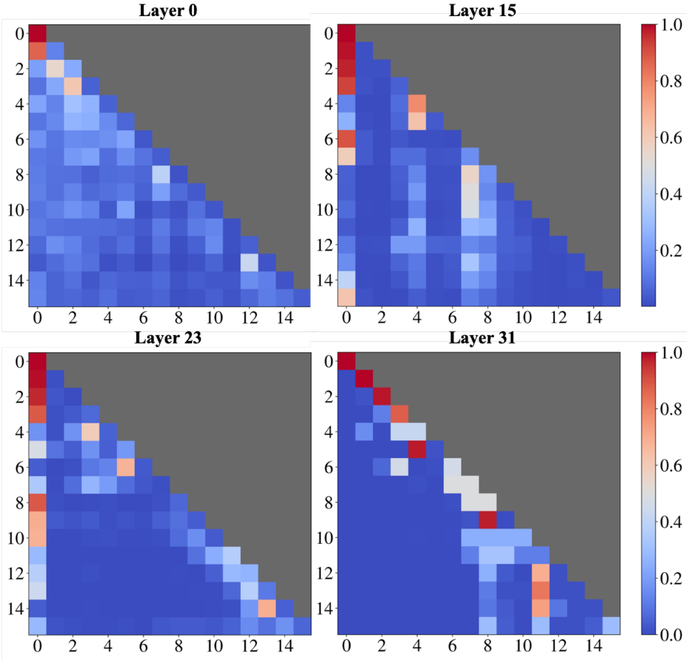


Fig. 6: Attention sparsity observed across different steps and layers of OPT model inference.

Fig. 7: Attention weight distribution for OPT-6.7B.



$$A = \begin{bmatrix} A_{11} & 0 & 0 & 0 \\ A_{21} & A_{22} & 0 & 0 \\ A_{31} & A_{32} & A_{33} & 0 \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

$\text{Attn}_4(Q,K,V) = A_{41}V_1 + A_{42}V_2 + A_{43}V_3 + A_{44}V_4$

Assume A_{42} is close to zero, we have

$\text{Attn}_4(Q,K,V) \approx A_{41}V_1 + A_{43}V_3 + A_{44}V_4$

Therefore, we do not need to calculate A_{42}, and do not need KV tensors for the second token

Proposal

UCF

How do we leverage sparsity for KV caching?

1. Identifying important tokens (Algorithm): we need a low-cost mechanism to distinguish important tokens without hurting LLM accuracy

2. Sparsity-aware Caching Policy (System): when GPU cannot hold all the KV tensors, we need to design a suitable low-overhead caching policy to allocate KV tensors between CPU and GPU and ensure a low miss rate;

3. Caching vs. Recomputation (System): for longer sequences, the benefits of KV caching diminishes, we need to consider recomputation of partial KV tensors instead of caching.

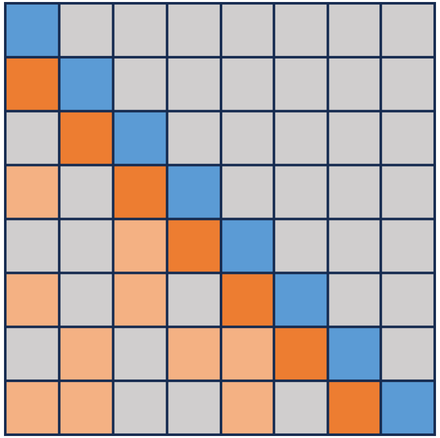
Algorithm Design

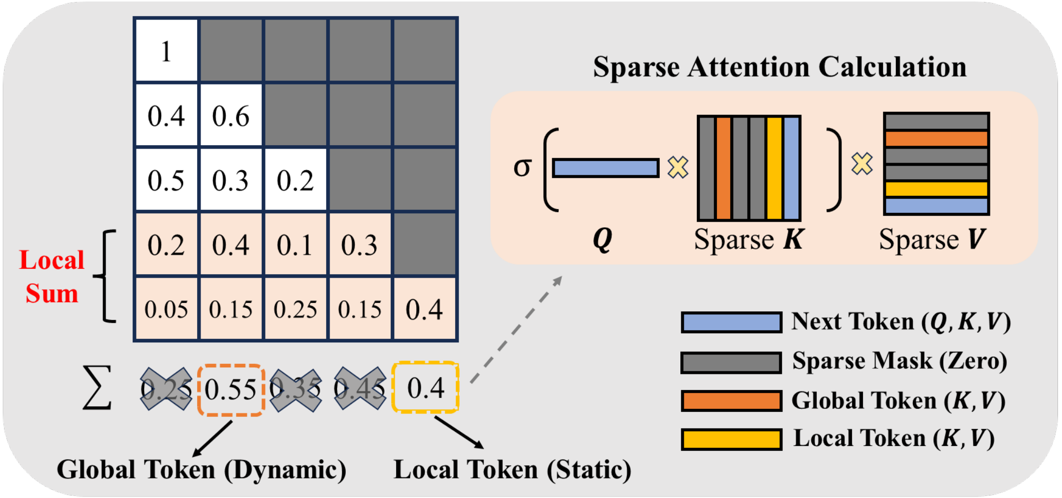
Sparse Window Attention (SWA):

- Global Dynamic Sparse Patterns: determined by local attention weight sum (light orange color)

- Local Static Sparse Patterns : determined by recency (dark orange color)

SWA Generated Patterns





Algorithm Design

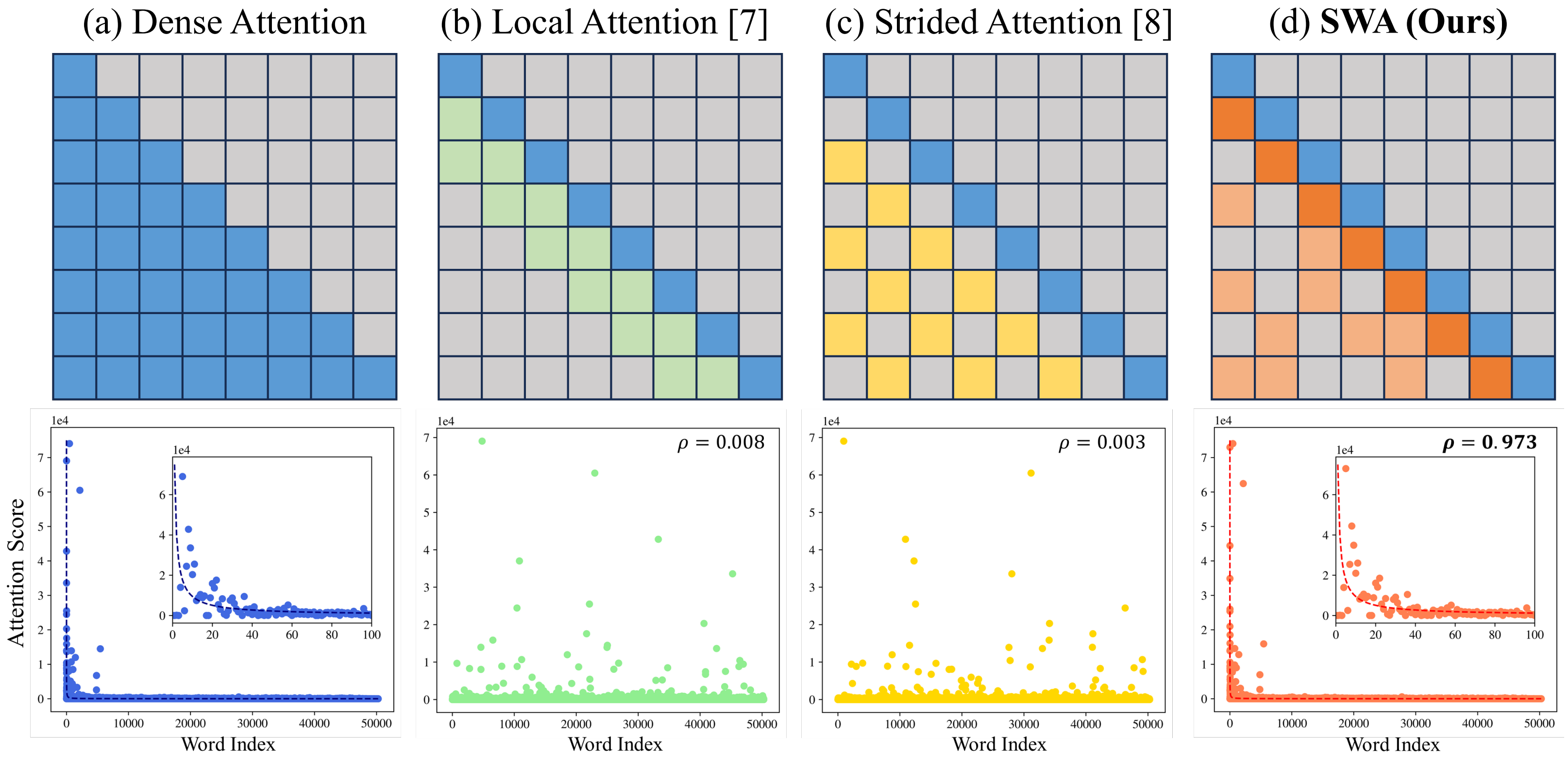


Fig. 8: Comparison of different attention methods. Top row illustrates the attention patterns, and the bottom row compares the final attention score distribution

System Design

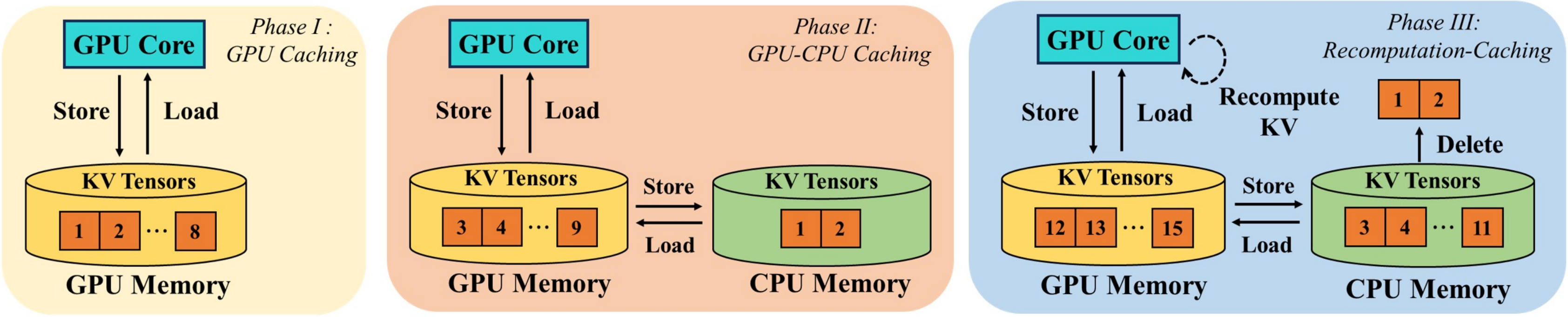
Three-phase Dynamic Scheduling:

We divide the LLM inference into three phases to balance the caching and Recomputation

- Phase I: *GPU Caching*. All KV tensors can fit in GPU memory.

- Phase II: *GPU-CPU Caching*. Split KV tensors at token-level on both GPU and CPU.

- Phase III: *Caching-Recomputation*. Delete partial KV tensors and perform recomputation instead if needed



System Design

UCF

How do we determine the phase switch step and offload/recomputation ratio?

We formulate this question into an optimization problem to minimize total execution time.

- Size of KV tensors: $4 \cdot b \cdot l \cdot h$
- Number of tokens moved from GPU to CPU: $\theta_j^c = \alpha(j + s)$
- Number of tokens moved from GPU to CPU: θ_g^c

For each step, the execution time is:

$$T_j^m(\alpha) = \frac{4 \cdot b \cdot l \cdot h \cdot (\theta_j^c + \theta_g^c)}{B}$$

The total execution time is:

$$\min_{\{\alpha,\beta,p_1,p_2\}} \sum_{j=1}^{p_2} T_j^c + \sum_{j=p_1}^{p_2} T_j^m(\alpha) + \sum_{j=p_2}^{n} T_j^r(\beta)$$

TABLE II: Notations.

h, l, b	hidden dimension, layer count, batch size
s, n	input length, output length
r, B	KV caching ratio, CPU-GPU bandwidth
α, β, p_1, p_2	offload/recompute ratio, phase switch step
T^c, T^r	Time for compute and recompute
T^m	Time for KV caching (CPU-GPU)

Solution:

- Divide the problem into two sub-problems, including an I/O problem and a computation problem.
- For I/O problem, we can use greedy search
- For computation, we can use profiling

Algorithm-System Co-Design

Additional System Optimization

- KV Compression (quantizing FP16 KV tensors to INT8)

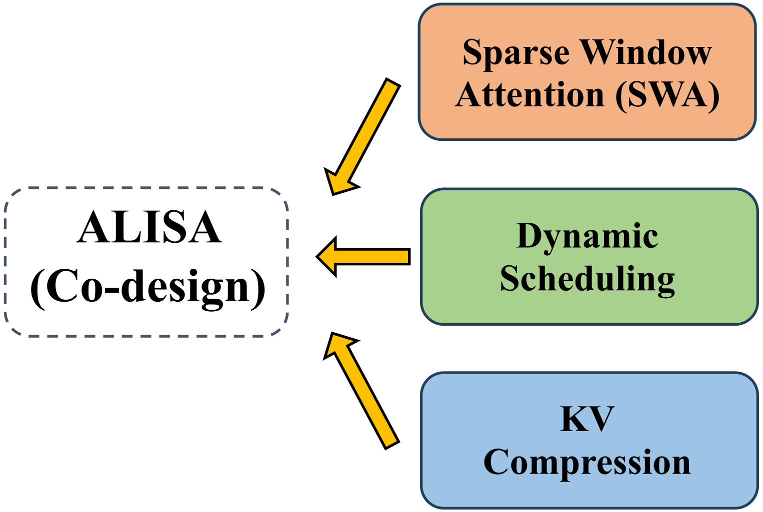


Table I: Comparison of our ALISA and prior works.

Design	vLLM [19]	FlexGen [29]	ALISA (Ours)
Sparse Attn.	✗	✗	✓
Caching Granularity	Block-level (Static)	Head-level (Static)	Token-level (Dynamic)
Recomputation	✓	✗	✓
Scenario	Online (Multi-GPU)	Offline (Single-GPU)	Offline (Single-GPU)
Co-Design	✗	✗	✓

Evaluation

UCF

Experimental Settings:

- Models: OPT (6.7B, 13B, 30B), LLaMA (7B, 13B, 33B), Pythia (6.9B, 12B)

- Algorithm Baselines: Dense attention, Local attention [7], Strided attention [8]

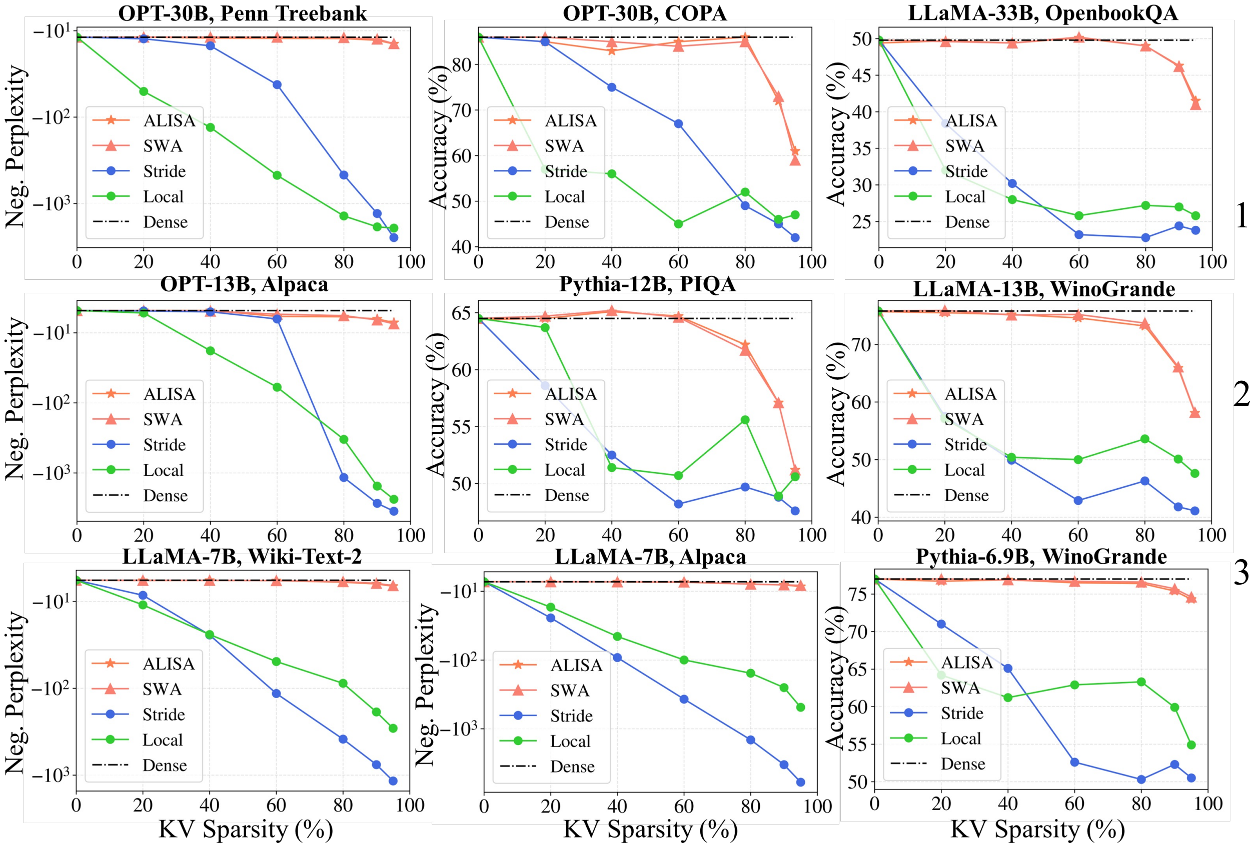
- System Baselines: DeepSpeed [13], Accelerate [14], FlexGen [3], vLLM [4]

- Datasets: Alpaca, Penn Treebank, Wiki-Text-2 (language modeling)
  OpenBookQA, PIQA, COPA, WinoGrande (question answering)

- Metrics: Perplexity, Accuracy (Algorithm), Throughput (system)

- Hardware Platforms: V100 16/32 GB, H100 80 GB, 128 GB DRAM (single GPU-CPU sys)

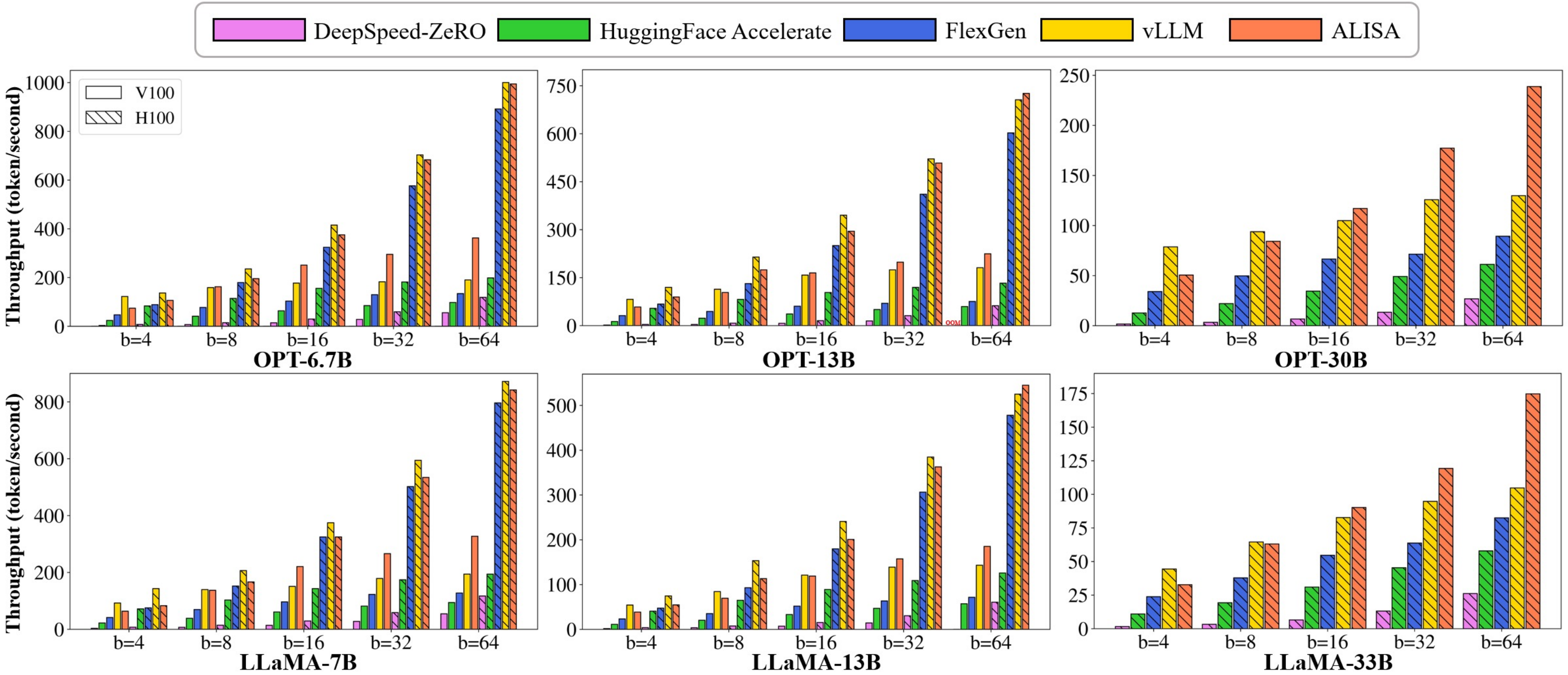
Algorithm Results



1. ALISA consistently outperforms local and strided attention **across different model types and scales**
2. ALISA can maintain **identical** performance as dense attention to up to **80% KV sparsity**
3. KV compression has almost **no effect** on accuracy

Full results can be found in the paper

System Results

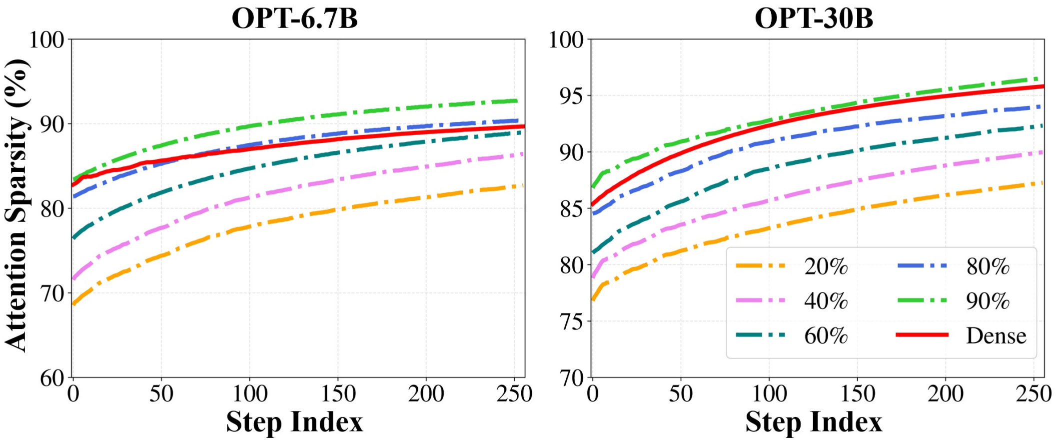


1. Compared to FlexGen, ALISA achieves 1.4~3.0× throughput improvement, showing much better scalability across different model sizes and batch sizes

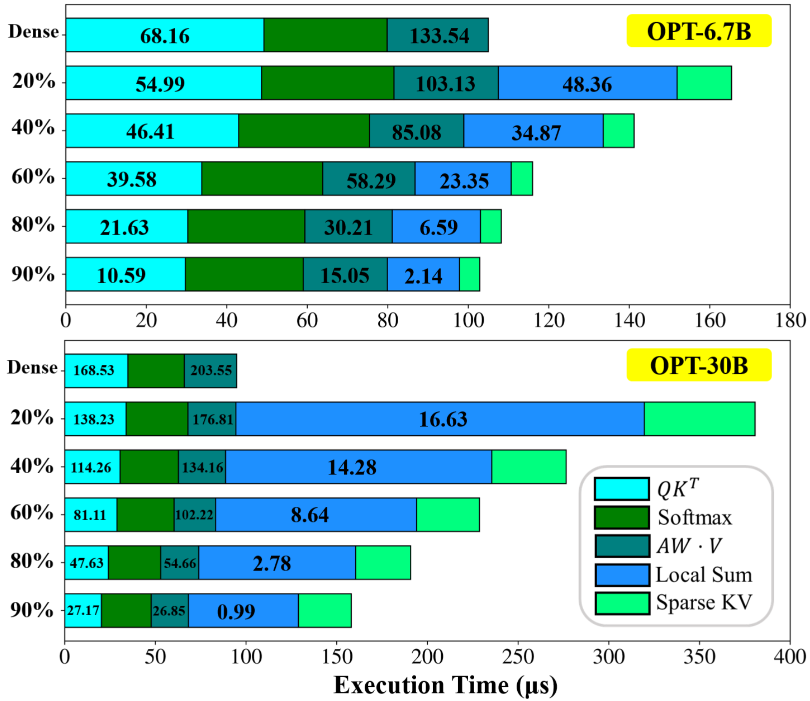
2. Compared to vLLM, under large batch sizes, ALISA can sustain up to 1.9× higher throughput

Performance Analysis

Attainable Sparsity



Kernel-level Breakdown

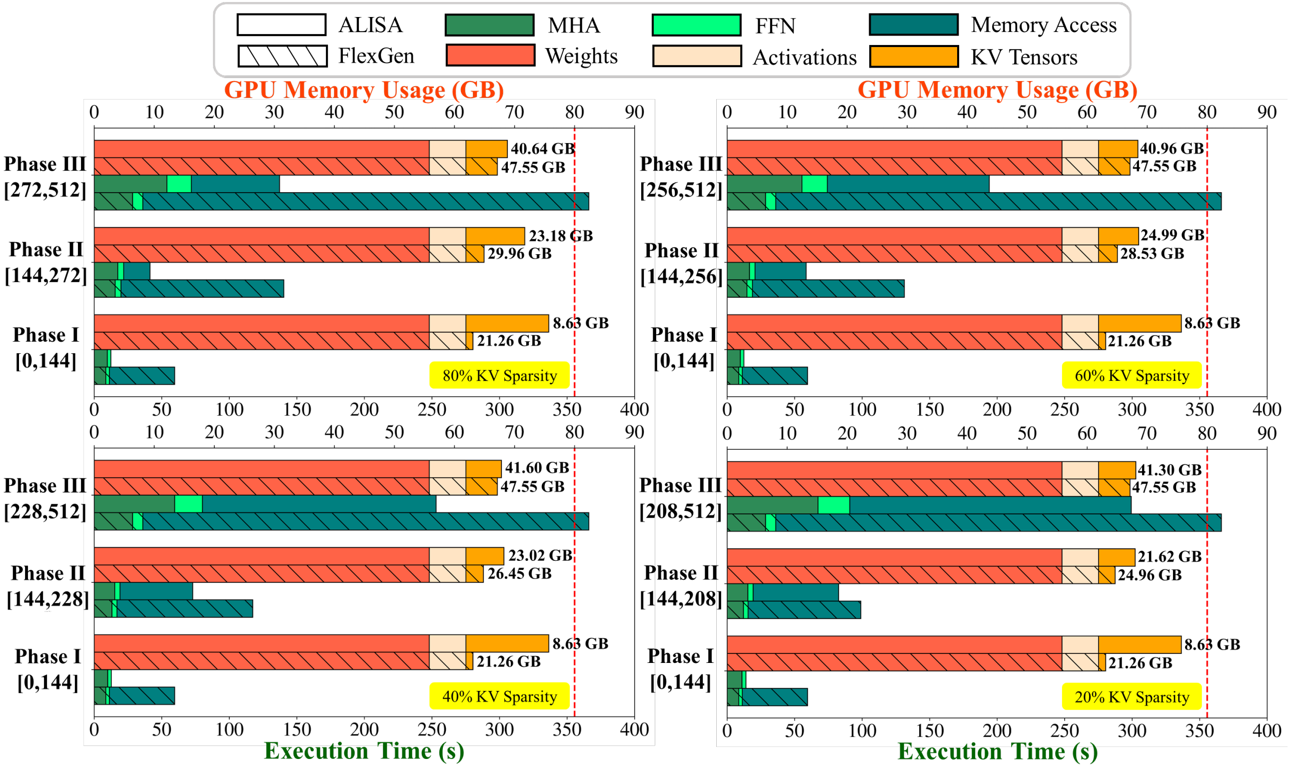


1. KV Sparsity increases the sparsity by creating sparse KV tensors, which is close to dense attention

2. There exists compute under-utilization in SWA calculation, but the overall overhead is relatively small against dense attention

Performance Analysis

LLM Inference Breakdown

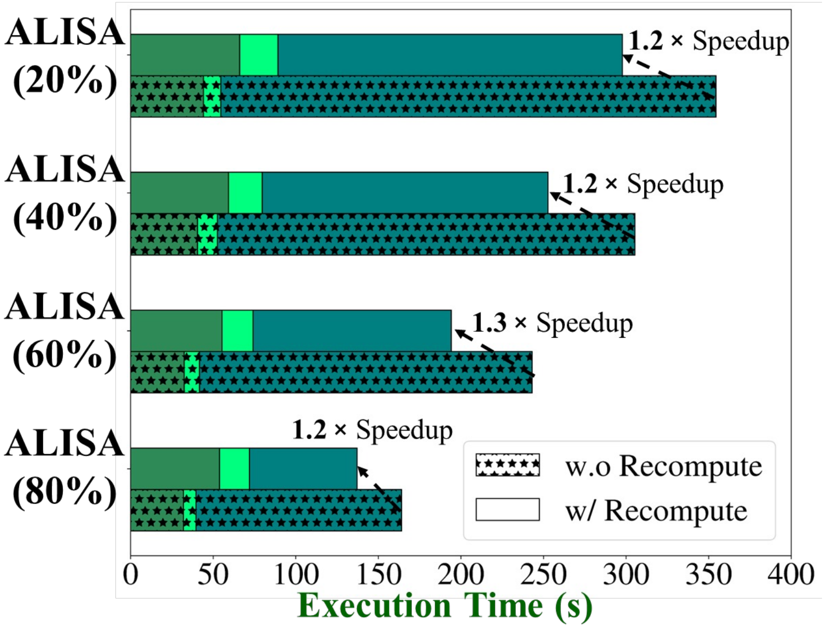


1. ALISA improves upon FlexGen across **different phases**
2. With **higher KV sparsity**, the speedup of ALISA over FlexGen is **more significant**
3. ALISA makes **better use of the GPU memory**

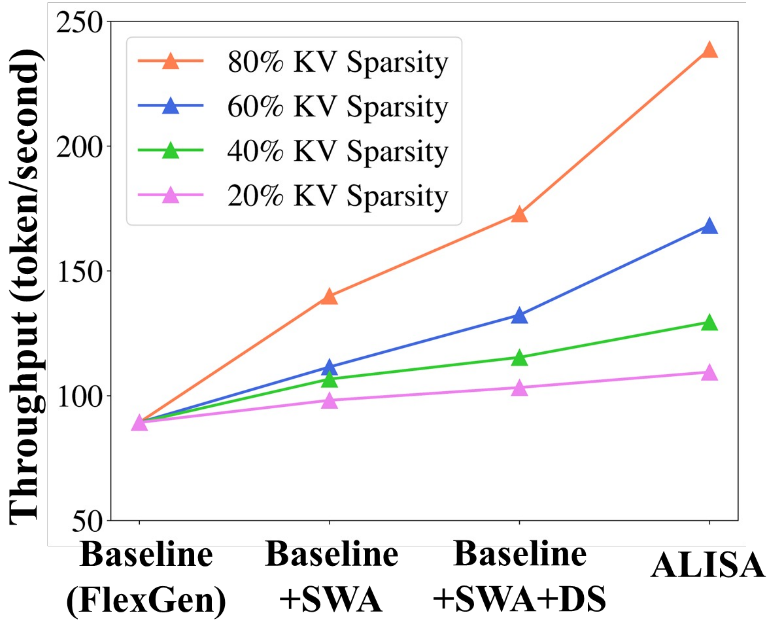
Performance Analysis

Ablation Study

- Impact of Recomputation



- Impact of each technique



Conclusion

UCF

- We identify the challenges in KV caching for LLM inference and propose an algorithm-system co-design solution, ALISA, for efficient LLM inference

- On the algorithm level, we propose sparse window attention (SWA) that creates a mixture of globally dynamic and locally static sparse patterns in KV tensors to reduce the memory footprint while maintaining high accuracy.

- On the system level, we design a three-phase scheduler to dynamically allocate KV tensors between GPU and CPU memory to reduce data transfer at the token level.

- Extensive experiments demonstrate that ALISA can significantly reduce the memory footprint of KV tensors and increase the throughput over previous baselines, with negligible accuracy drop

References

[1] B. Wang. Timeline of Open and Proprietary Large Language Models.
[2] M. Ott, et al. "Fairseq: A fast, extensible toolkit for sequence modeling," NACCL, 2019.
[3] Y. Sheng, et al. "High-throughput generative inference of large language models with a single gpu," ICML 2023.
[4] W. Kwon, et al. "Efficient memory management for large language model serving with pagedattention," SOSP, 2023.
[5] N. Kitaev , et al. "Reformer: The efficient transformer," ICLR,.
[6] S. Wang, et al. "Linformer: Selfattention with linear complexity," ArXiv, 2020.
[7] I. Beltagy, et al. "Longformer: The long document transformer," ArXiv, 2020.
[8] R. Child, et al. "Generating long sequences with sparse transformers," ArXiv, 2019.
[9] H. Wang, et al. "Spatten: Efficient sparse attention architecture with cascade token and head pruning," HPCA, 2020.
[10] J. Dass, et al. "Vitality: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention," HPCA, 2022.
[11] J. Lin , et al. "Awq: Activation-aware weight quantization for llm compression and acceleration," ArXiv, 2023.
[12] E. Frantar, et al, "Gptq: Accurate post-training quantization for generative pre-trained transformers," ICLR, 2023.
[13] R. Y. Aminabadi, et al. "Deepspeedinference: Enabling efficient inference of transformer models at unprecedented scale," SC, 2022.
[14] T. Wolf , et al. "Huggingface's transformers: State-of-the-art natural language processing," ArXiv, 2019.

Acknowledgements

- We appreciate the generous support from National Science Foundation 1907765, and 2400014.

- Faculty and Student Investigators from the UCF Computer Systems and Data Science Laboratory
  - Mr. Youpeng Zhao
  - Dr. Di Wu
  - Many others…