LLMShare: Optimizing LLM Inference Serving with Hardware Architecture Exploration

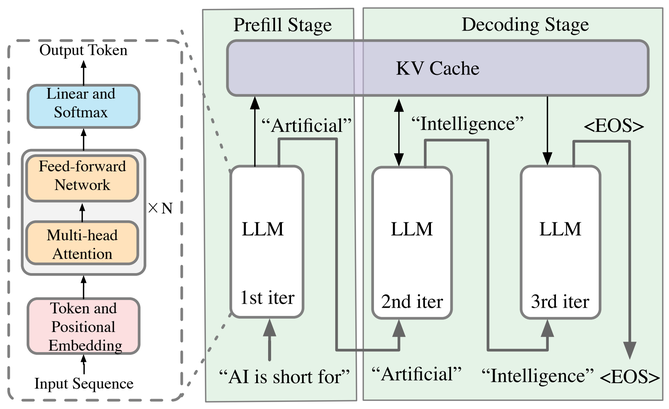
Hongduo Liu[1], Chen Bai[2], Peng Xu[1], Lihao Yin[3], Xianzhi Yu[3] Hui-Ling Zhen[3], Mingxuan Yuan[3], Tsung-Yi Ho[1], Bei Yu[1]
[1]The Chinese University of Hong Kong [2]Hong Kong University of Science and Technology [3]Huawei

Background

- LLMs are getting smarter, but also larger.
- Serving LLMs require substantial hardware resources.
- Serving requires strict service-level objectives.

LLM Inference Process Overview

- **Prefill Phase:** Process the entire input prompt to set up context.
- **Decoding Phase:** Generate output tokens autoregressively using KV cache.
- Prefill and Decoding pose different computation and memory requirements.



Motivation

- PD Disaggregation: prefill and decoding are handled by sperate machines.
- Mismatches between hardware capabilities and P/D requirements still exist.

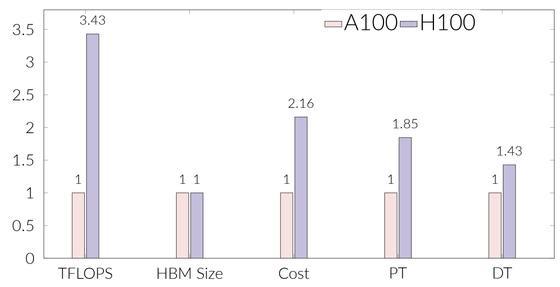


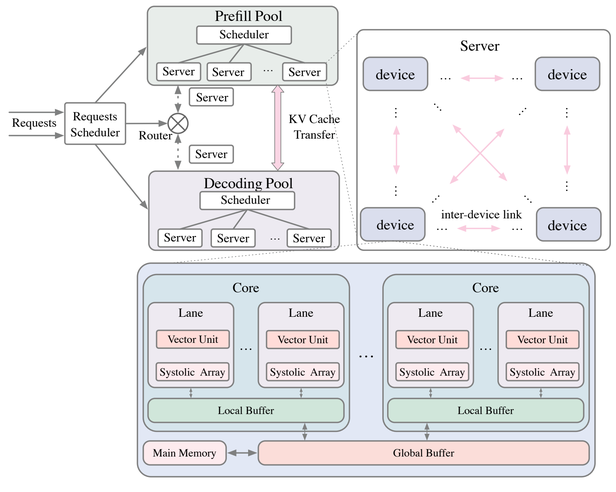
Figure 1. Comparison of NVIDIA A100 and H100 cluster with 8 GPUs on Llama-70b without batching. 'PT' denotes prefill phase throughput, and 'DT' denotes decoding phase throughput.

Research Objectives

`Can we find a hardware configuration to achieve a better performance-cost tradeoff?`

1. Model the performance and cost of different hardware configurations.
   - A simulator.
2. Find a systematic way to explore optimal hardware configurations.
   - A design space exploration algorithm.

LLM Serving System Modeling



Design Space

Parameter	Notation	Value Range	#
Server Count	sc	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	10
Device Count	dc	4, 8, 12, 16	4
Link Count Per Device	lc	6, 12, 18, 24	4
Main Memory (GB)	mm	40, 64, 80, 96, 112, 128	6
Global Buffer (MB)	gb	20, 30, 40, 50, 60, 70, 80, 90, 100, 110	10
Core Count	cc	72, 96, 108, 132, 156, 180	6
Local Buffer (KB)	lb	64, 128, 192, 256, 320, 384, 448, 512	8
Lane Count	lc	1, 2, 4, 8	4
Array Height	ah	16, 32, 64, 128	4
Vector Width	vw	16, 32, 64, 128	4

Table 1. Design space of the prefill and decoding pools. The entire design space of an LLM serving system is nearly 9×10^{14}.

LLMShare Overview

- A LLM serving simulator to get serving time and total cost.
- A Bayesian optimization framework to find optimal serving system configurations.

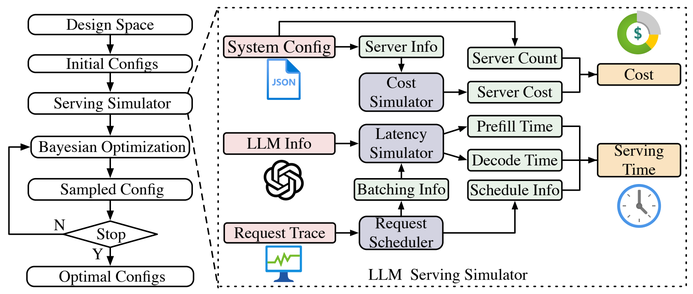
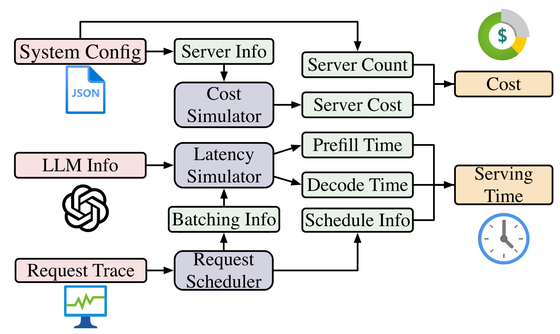


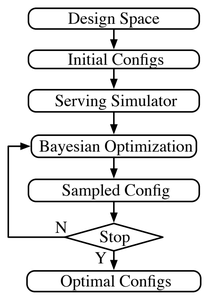
Figure 2. The overview of LLMShare.

Simulator Framework Overview

- Inputs:
  - Serving system configuration (number of servers, device specs, etc.).
  - LLM information (e.g., number of layers, attention heads).
  - Request trace (arrival timings, input/output token sizes).
- Outputs: Serving time and total cost.



Design Space Exploration Framework



1. Initialize with a set of sample designs.
   - Memory-Centric Initialization
2. Get simulated cost and serving time by the simulator.
3. Fit a surrogate model.
   - Deep Tree Kernel
4. Select the most promising design by optimizing the acquisition function.
5. Update the surrogate model using the selected design.

Memory-Centric Initialization (MCI)

Algorithm 1 Memory-Centric Initialization

Input: • $\mathcal{U}$: unsampled design space with n configurations;
• t: total number of initial configurations to select;
• u: number of groups used during sampling.
Output: $\mathcal{D}_x$ with $|\mathcal{D}_x| = t$ ▷ Selected initial designs

Compute the total main memory size for each design:
for $i = 1$ to n **do**

$$c_i = \vec{x}^p_{i,\text{sc}} \cdot \vec{x}^p_{i,\text{dc}} \cdot \vec{x}^p_{i,\text{mm}} + \vec{x}^d_{i,\text{sc}} \cdot \vec{x}^d_{i,\text{dc}} \cdot \vec{x}^d_{i,\text{mm}};$$

end for
Determine percentiles: $P = \left\{ \frac{100 \times j}{u} \,\middle|\, j = 0, 1, \ldots, u \right\}$;
Compute bin edges for the percentiles of $\{c_i\}_{i=1}^n$:

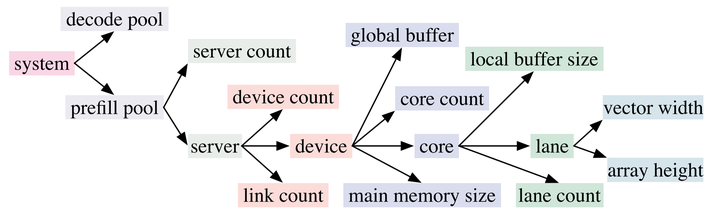
$$B = \left\{ b_j = \text{Percentile}(\{c_i\}_{i=1}^n, p_j) \,\middle|\, j = 0, 1, \ldots, u \right\};$$

Compute base sample count per group: $q \leftarrow \left\lfloor \frac{t}{u} \right\rfloor$;
Compute remainder: $r \leftarrow t \bmod u$;
Initialize $\mathcal{D}_x \leftarrow \emptyset$;
for $j = 1$ to u **do**
  $\mathcal{G}_j = \left\{ i \,\middle|\, b_{j-1} \le c_i < b_j \right\}$;
  if $j \le r$ **then**
    $s_j \leftarrow q + 1$;
  else
    $s_j \leftarrow q$;
  end if
  Select s_j samples from $\mathcal{G}_j$: $\mathcal{S}_j = \textbf{TED}(\mathcal{G}_j, s_j)$;
  $\mathcal{D}_x \leftarrow \mathcal{D}_x \cup \mathcal{S}_j$;
end for
return $\mathcal{D}_x$

- Memory capacity can largely affect throughput and cost.
- Divide the design space into groups based on memory capacity.
- Sample in each group using traditional sampling method like transductive experimental design (TED).

Deep Tree Kernel (DTK)

- Each configuration is represented as a tree
- A hierarchical MLP is used to aggregate features



Multi-Objective Bayesian Optimization

- Surrogate model: Gaussian Process with Deep Tree Kernel
- Acquisition function: Expected Hypervolume Improvement (EHVI)

Experimental Setup

- **Underlying LLM:** GPT-3 175B.
- **Serving Trace:** 2454 requests in 2 mins.
  - The distribution of token sizes is derived from a Microsoft Azure production trace
- **Cost and Prefill/Decoding Time Simulation:** LLMCompass [6], which only has 5% simulation error.
- **Request Scheduler:** Splitwise [4].
- **Design Space:** Subset of 1,055 configurations sampled from the whole design space.
- **DSE Process:** Initialization with 10 samples and perform 20 optimization iterations.

Results: Comparison of different algorithms

Algorithms	Normalized ADRS	Hypervolume (10^8)
SVR [1]	0.1811	4.9593
DAC'16 [3]	0.1718	4.9714
ASPDAC'20 [5]	0.1805	4.9513
ICCAD'21 [2]	0.2059	4.8134
LLMShare	**0.1589**	**5.0552**

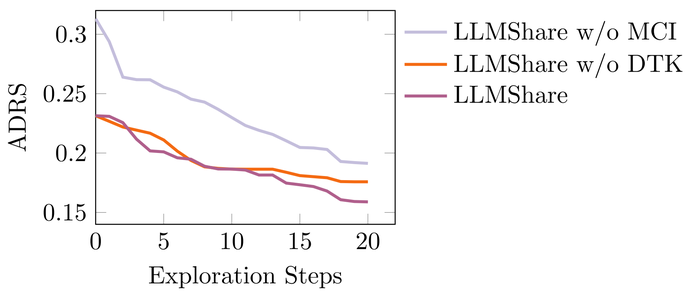


Figure 3. Ablation study on the effectiveness of DTK and MCI.

References

[1] Mariette Awad, Rahul Khanna, Mariette Awad, and Rahul Khanna. Support vector regression. *Efficient learning machines: Theories, concepts, and applications for engineers and system designers*, pages 67–80, 2015.

[2] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong. Boom-explorer: Risc-v boom microarchitecture design space exploration framework. In *ICCAD*, pages 1–9. IEEE, 2021.

[3] Dandan Li, Shuzhen Yao, Yu-Hang Liu, Senzhang Wang, and Xian-He Sun. Efficient design space exploration via statistical sampling and adaboost learning. In *DAC*, pages 1–6, 2016.

[4] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *ISCA*, pages 118–132. IEEE, 2024.

[5] Zhiyao Xie, Guan-Qi Fang, Yu-Hung Huang, Haoxing Ren, Yanqing Zhang, Brucek Khailany, Shao-Yun Fang, Jiang Hu, Yiran Chen, and Erick Carvajal Barboza. Fist: A feature-importance sampling and tree-based method for automatic design flow parameter tuning. In *ASP-DAC*, pages 19–25. IEEE, 2020.

[6] Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. Llmcompass: Enabling efficient hardware design for large language model inference. In *ISCA*, pages 1080–1096. IEEE, 2024.

liuhongduosc@gmail.com