LLMServingSim2.0: A Unified Simulator for Heterogeneous Hardware and Serving Techniques in LLM Infrastructure

Jaehong Cho, Hyunmin Choi, *Member, IEEE*, and Jongse Park, *Senior Member, IEEE*

***Abstract*—This paper introduces LLMServingSim2.0, a system simulator designed for exploring heterogeneous hardware in large-scale LLM serving systems. LLMServingSim2.0 addresses two key limitations of its predecessor: (1) integrating hardware models into system-level simulators is non-trivial due to the lack of a clear abstraction, and (2) existing simulators support only a narrow subset of serving techniques, leaving no infrastructure that captures the breadth of approaches in modern LLM serving. To overcome these issues, LLMServingSim2.0 adopts trace-driven performance modeling, accompanied by an operator-level latency profiler, enabling the integration of new accelerators with a single command. It further embeds up-to-date serving techniques while exposing flexible interfaces for request routing, cache management, and scheduling policies. In a TPU case study, our profiler requires 18.5× fewer LoC and outperforms the predecessor's hardware-simulator integration, demonstrating LLMServingSim2.0's low-effort hardware extensibility. Our experiments further show that LLMServingSim2.0 reproduces GPU-based LLM serving with 1.9% error, while maintaining practical simulation time, making it a comprehensive platform for both hardware developers and LLM service providers.**

***Index Terms*—Large language model (LLM), Inference serving, Heterogeneous hardware, Simulation infrastructure**

I. Introduction

WHILE large language model (LLM) serving technologies are advancing rapidly across both academia and industry, current efforts remain separated: (1) the systems community focuses on optimizing software for GPU-based scale-out infrastructures, while (2) the architecture community develops custom hardware accelerators without an easy way to evaluate them under these evolving software environments. This separation leaves hardware developers with few options for validating their accelerators at deployment scale, while also making it difficult for LLM service providers to assess and adopt emerging hardware in their serving systems.

To address these challenges, LLMServingSim [1] developed a system-level simulator for heterogeneous LLM serving, allowing researchers to plug in custom accelerators and explore software techniques. However, there are two key limitations: (1) integrating new hardware models is non-trivial due to the lack of a clear abstraction, and (2) existing simulators [2]–[4] support only a narrow subset of serving techniques, leaving them insufficient to capture modern deployments. Table I summarizes the differences among existing simulators and our work. Prior efforts target isolated aspects of LLM serving, but none provide a unified framework that combines parallelism, caching, offloading, and disaggregation with realistic memory and network modeling. As deployments increasingly employ these techniques together, a framework that captures both operator-level performance and system-level dynamics becomes essential.

TABLE I
COMPARISON OF LLM SERVING SIMULATORS

Simulator	Dissagg.		Parallelism			Memory Model		
	PD	AF	PP/TP	DP	EP	PA	PC	EO
LLMServingSim [1]	×	✓	✓	×	×	✓	×	×
Vidur [2]	×	×	✓	✓	×	△	×	×
APEX [3]	×	×	✓	✓	✓	×	×	×
TokenSim [4]	✓	×	✓	△	×	✓	△	×
Ours	✓	✓	✓	✓	✓	✓	✓	✓

PD: Prefill/Decode Disaggregation, AF: Attention/FFN Disaggregation, PP/TP: Pipeline/Tensor Parallelism, DP: Data Parallelism, EP: Expert Parallelism, PA: PagedAttention, PC: Prefix Caching, EO: Expert Offloading. ✓: fully supported; ×: not supported; △: limited or partial support.

To this end, we present LLMServingSim2.0, a hardware/-software co-simulation infrastructure that streamlines the integration of custom accelerators and incorporates modern serving techniques, allowing systematic evaluation of large-scale LLM serving. Compared to its predecessor, LLMServingSim2.0 introduces four key advancements.

- **Trace-driven performance modeling.** It supports trace-driven performance models, enabling researchers to easily perform system-level design explorations with diverse hardware and model configurations.
- **Multi-instance and disaggregated serving.** It enables multi-instance simulation and P/D disaggregation, capturing complex interactions between heterogeneous instances.
- **MoE support with expert parallelism and offloading.** It provides MoE model support with expert parallelism and offloading, faithfully modeling network congestion, memory bandwidth contention, and expert routing dynamics.
- **Memory-aware serving with prefix caching.** It introduces the first network and memory-aware simulation of prefix caching, including a customizable management policy.

Unlike prior simulators, LLMServingSim2.0 brings these four capabilities together in a single flexible framework, enabling researchers to explore heterogeneous hardware, diverse serving topologies, and customizable policies. This unification bridges the gap between accuracy, scalability, and usability, establishing LLMServingSim2.0 as a comprehensive platform for large-scale LLM serving. LLMServingSim2.0 is available at https://github.com/casys-kaist/llmservingsim.

II. LLMServingSim2.0

We design LLMServingSim2.0, a next-generation system-level simulator for LLM inference that models a large-scale distributed LLM serving. Fig. 1(a) illustrates a simple example system supported by our simulator. LLMServingSim2.0 models heterogeneous instances along two dimensions: (1) model type (dense and MoE) and (2) serving structure (non-disaggregated and P/D disaggregated). Each instance can be

Manuscript received October 5, 2025.

Jaehong Cho, Hyunmin Choi, and Jongse Park are with the School of Computing, Korea Advanced Institute of Science and Technology, Daejeon, 34141, South Korea. (e-mail: jhcho@casys.kaist.ac.kr, hmchoi@casys.kaist.ac.kr, jspark@casys.kaist.ac.kr).

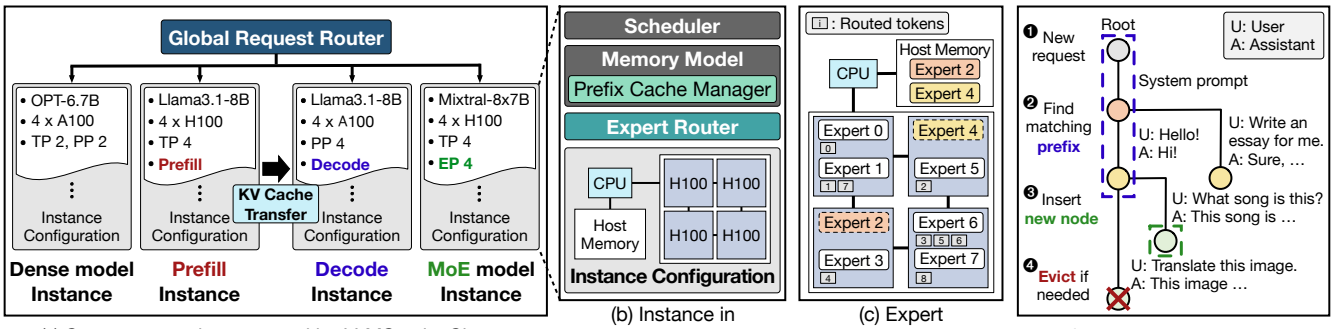

Fig. 1. Overview of LLMServingSim2.0 simulator.

configured with distinct compute/memory resources, parallelism schemes, and network topologies, demonstrating the flexibility of our simulator. Building on this flexibility, the LLMServingSim2.0 further supports advanced deployment scenarios, including multi-instance serving, P/D disaggregation, MoE expert parallelism/offloading (Fig. 1(c)), and prefix caching (Fig. 1(d)) within a unified simulation environment.

A. Trace-driven Flexible Performance Modeling

The key advantage of the original LLMServingSim [1] was its ability to compose large-scale heterogeneous serving systems by integrating different hardware simulators, such as NPU and PIM. However, porting a new hardware simulator to the platform can be a time-consuming and challenging task. Moreover, due to the massive amount of repetitive computation and the autoregressive nature of LLMs, cycle-accurate hardware simulation remained slow even with computation reuse.

Operator-level profiler. To address these challenges, LLMServingSim2.0 introduces trace-driven performance modeling, enabling users to build performance studies directly from traces without extensive hardware simulator integration. We developed *operator-level profiler*, a PyTorch-based profiling tool that inserts hooks between LLM layers to measure layer-wise latency. This profiler collects all the information required by LLMServingSim2.0 and, through validation against real execution, helps maintain high simulation accuracy. With the profiler, users can analyze any model on their own hardware with a single-line command. Compared to repeatedly simulating hardware in cycle-accurate mode, this approach is far more convenient and achieves 232× faster execution on average, while enabling flexible integration of diverse hardware platforms and models.

B. Multi-instance and Disaggregation for Realistic Serving

In large-scale LLM serving, hundreds of instances are deployed across rack-scale clusters or even entire data centers. Such multi-instance execution is indispensable for emerging LLM-based applications and it introduces critical system-level effects, such as resource sharing, network contention, and global routing dynamics. However, the original LLMServingSim only supported single-instance simulation, limiting its ability to capture the realities of modern large-scale serving.

Global request router. To tackle this challenge, LLMServingSim2.0 introduces a *global request router* for multi-instance serving. As illustrated in Fig. 1(a), the global request router resides outside individual instances and manages all instances, dynamically dispatching requests according to the current resource status. Such routing can adapt to diverse factors, including load balancing, workload characteristics, and the state of prefix caches. Because routing policies critically affect overall performance, LLMServingSim2.0 provides customizable routing interfaces, enabling researchers to easily implement and explore new policies for large-scale LLM serving.

Heterogeneous multi-instance. As shown in Fig. 1(b), LLMServingSim2.0 realizes multi-instance support by replicating components that previously existed as global entities, thereby giving each instance its own scheduler and memory model. In real LLM services, multiple models often coexist, each with different resource demands, while server hardware can also vary. To capture this heterogeneity, LLMServingSim2.0 allows users to construct heterogeneous multi-instance systems. As depicted in Fig. 1(a), each instance can be configured with different models, hardware types and counts, memory sizes and bandwidths, parallelism schemes, as well as distinct network topology. This flexibility enables LLMServingSim2.0 to model realistic serving infrastructures and allows researchers to explore a wider range of system designs.

P/D disaggregation. Beyond hosting multiple heterogeneous instances, real-world deployments increasingly adopt P/D disaggregation [5], [6], where the compute-intensive prefill stage and the memory-intensive decode stage are placed on separate clusters. To support P/D disaggregation, LLMServingSim2.0 introduces dedicated prefill and decode instance types as shown in Fig. 1(a). Once the prefill stage completes in a prefill instance, the request and its KV cache are forwarded to a decode instance for continuation. Moreover, LLMServingSim2.0 exposes this KV cache transfer policy as a configurable option, allowing researchers to experiment with alternative strategies. Building on phase-aware optimizations in various P/D-disaggregated serving systems [5], [6], LLMServingSim 2.0 supports systematic exploration of these and broader design choices across diverse system configurations.

C. MoE Support: Expert Parallelism and Offloading

LLM advances increasingly rely on compute and memory efficient designs to improve performance. Among various approaches, MoE architecture has gained significant traction as a high-performing, promising approach. However, the original LLMServingSim only supports dense models and thus can not capture the emerging trend of MoE-based LLM serving.

Expert router. To address this limitation, LLMServingSim2.0

TABLE II
SERVING CONFIGURATIONS USED FOR EVALUATION.

Config	Description	Instance / GPU per Inst.
S(D/M)	Single-instance Dense/MoE	1 inst., 1 × RTX3090
M(D/M)	Multi-instance Dense/MoE	2 inst., 1 × RTX3090
PD(D/M)	P/D-disaggregated Dense/MoE	2 inst., 1 × RTX3090
* + PC	* + Prefix Cache	–

introduces support for MoE models through expert routing and parallelism. In MoE, a gate function routes tokens to the most suitable experts, thereby scaling up model capacity for higher accuracy while activating only a subset of experts per token to keep the computational cost nearly constant. We design an *expert router* (Fig. 1(b)) that mimics the behavior of real gate functions. This router can be flexibly customized, enabling researchers to experiment with alternative routing strategies.

Expert parallelism. Once tokens are routed to their designated experts, the next step is to distribute the experts across compute units, referred to as expert parallelism. As illustrated in Fig. 1(c), it partitions experts across compute units and routes tokens to their assigned experts. In LLMServingSim2.0, MoE instances are designed such that non-expert layers operate under conventional tensor or pipeline parallelism, while expert layers employ the expert router to dispatch tokens according to the assigned experts. Between these two types of layers, LLMServingSim2.0 models an all-to-all communication operation in the network to synchronize token dispatches across compute units. This design faithfully models the interplay between expert routing and parallel execution in MoE serving.

Expert offloading. While expert parallelism improves scalability, placing all experts directly on compute units is often infeasible when memory is limited, motivating a line of research on expert offloading [7], [8]. Some approaches prefetch experts to overlap computation with data transfer [7], while others offload experts to memory-intensive compute units (e.g., PIM) to improve performance [8]. To enable such studies, as shown in Fig. 1(c), LLMServingSim2.0 implements expert offloading schemes and, to the best of our knowledge, is the first system-level simulator to provide such capability. Researchers can flexibly configure where and how experts are offloaded, enabling exploration of diverse offloading strategies.

D. Memory-aware Serving with Prefix Caching

Prefix caching exploits the high reuse of input sequences in LLM inference by reusing cached prefixes, thereby significantly reducing Time to First Token (TTFT). Recent LLM serving frameworks [9], [10] already support prefix caching, and active research continues to explore new caching mechanisms and policies [11]. Despite its importance, none of the prior works models it systemically, leaving the effects of prefix caching unexamined in simulation. This gap motivates a simulator that can capture prefix caching and its system-level implications, enabling systematic evaluation of cache policies.

Prefix cache manager. To address this gap, LLMServingSim2.0 implements prefix caching and, to the best of our knowledge, is the first system-level simulator to support this feature. We build this feature on RadixAttention [10] and extend the memory model with a radix tree–based prefix cache controlled by a *prefix cache manager* (Fig. 1(d)). Each request triggers a longest-prefix match from radix tree, and on a prefix hit, corresponding memory-transfer events are inserted into execution trace to model loading of the blocks. After prefill phase, new prefixes are inserted into radix tree, and capacity pressure triggers eviction. In the current implementation, the radix tree uses the compute unit's local memory (e.g., GPU or NPU memory) as the first-tier cache, with evicted blocks spilling into host CPU memory. The hierarchy is easily reconfigured (e.g., to include SSD tiers), and LLMServingSim2.0 also supports both per-instance and global shared caches to study different caching scopes. This flexible modeling of prefix caching allows researchers to explore diverse cache management strategies and understand their impact.

TABLE III
HARDWARE INTEGRATION PERFORMANCE OF LLMSERVINGSIM2.0

Simulator	LoC	Prof. Time	Sim. Time	Error
LLMServingSim [1]	4764	–	1524.7 min	14.7%†
LLMServingSim2.0	**258**	21 hr	**3.0 min**	**2.25%**

†Error rate of LLMServingSim is taken from the original paper [1].

III. EVALUATION

A. Methodology

System baseline. We use two baseline systems for evaluating LLMServingSim2.0: (1) a real GPU system equipped with four NVIDIA RTX 3090 GPUs and an Intel Xeon Gold 6326 CPU, and (2) a Google Colab system with a TPU-v6e-1 instance. As the serving software, we employ vLLM [9], a state-of-the-art framework for LLM inference. In our evaluation, we use two representative LLMs: Llama3.1-8B as a dense model and Phi-mini-MoE as a MoE model. For the workload, we sample 100 requests from ShareGPT [12] and synthesize arrival patterns using a Poisson process with a rate of 10 requests per second.

LLMServingSim2.0 configuration. To evaluate LLMServingSim2.0, we integrate GPU and TPU backends by extracting performance models through *operator-level profiler*. On the GPU side, we profile both LLM models on the GPU system mentioned above, configuring the simulator with matching device specifications of 24GB memory capacity, 936GB/s memory bandwidth, and interconnect modeled as PCIe 4.0×16. To further demonstrate the ease of hardware integration, we modify the profiler to run on a TPU-v6e-1 instance in Colab, configuring the simulator with an equivalent setup of 32GB of memory capacity, 1.6TB/s memory bandwidth, and an interconnect bandwidth of 800GB/s. Table II summarizes the serving configurations used to evaluate LLMServingSim2.0.

Simulator baseline. We also compare the hardware integration complexity and simulation time of LLMServingSim2.0 with previous LLMServingSim [1]. Specifically, we use two variants: (1) LLMServingSim with the hardware simulator enabled, and (2) LLMServingSim+, which omits hardware simulation by replaying pre-simulated results.

B. Hardware Integration

Table III compares the hardware integration cost of LLMServingSim2.0 against the previous LLMServingSim [1] when integrating a TPU backend, measured in terms of Lines of Code (LoC) excluding comment and blank lines, offline profiling time, online simulation time, and error rate compared

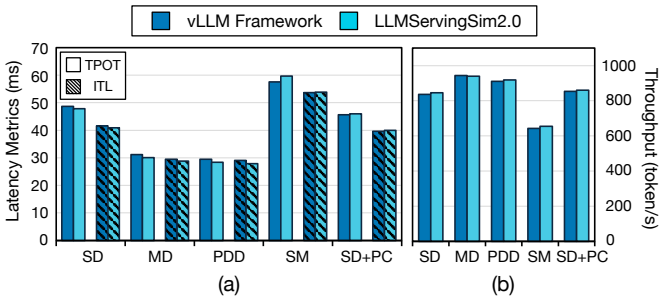

Fig. 2. Latency and throughput comparison of vLLM and LLMServingSim2.0, across five system configurations.

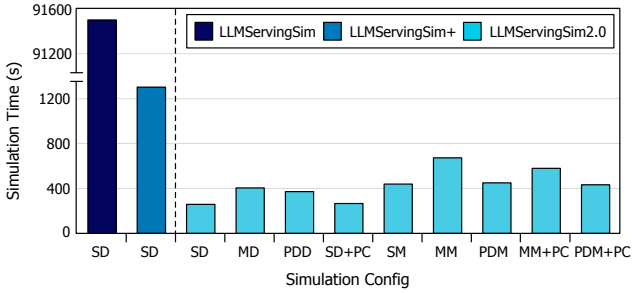

Fig. 3. Simulation time comparison of LLMServingSim2.0 against LLMServingSim and LLMServingSim+ across multiple system configurations.

to the real hardware. In the case of LLMServingSim, enabling the hardware simulator requires substantial code modifications, including rewriting the compilation and simulation flow to match its requirements and even creating new APIs for usability. In contrast, LLMServingSim2.0 reduces the implementation effort to only 258 lines of code, mainly by just replacing CUDA APIs with XLA APIs for TPU compatibility. Compared to the 4.8K lines required by LLMServingSim, this dramatically lowers the engineering burden while offline profiling takes only 21 hours using our *operator-level profiler*. Moreover, as LLMServingSim2.0 is trace-driven, simulation time is reduced by 509×, and since the profiler itself performs validation, the error rate decreases from 14.7% to 2.25%. This lightweight integration flow allows researchers to seamlessly integrate novel hardware and explore a wide range of systems.

C. Simulator Validation

Fig. 2 compares LLMServingSim2.0 with vLLM across five configurations. Fig. 2(a) shows average Time per Output Token (TPOT) and Inter-Token Latency (ITL), while Fig. 2(b) shows average token generation throughput. Across all configurations, the simulated latency and throughput closely follow the trends observed in real GPU-based LLM serving, while the error rate remains within 5%. Single-instance settings show lower error rates than multi-instance settings because multi-instance deployments introduce additional variability, including request routing and network contention, which are harder to capture in a simulator. Within the multi-instance case, PDD incurs higher error than MD because the two instances must coordinate and exchange intermediate results. For MoE, the gating function routes tokens differently across layers and batches, increasing variance and making accurate reproduction more difficult. Despite these challenges, LLMServingSim2.0 captures key sources of variability and closely reproduces real GPU-based serving behavior, indicating that it provides a reliable abstraction of modern LLM serving systems.

D. Simulation Time

Fig. 3 reports the wall-clock time to simulate 100 ShareGPT requests. We run LLMServingSim2.0 across nine configurations and compare against two baselines: LLMServingSim [1] and LLMServingSim+. LLMServingSim incurs the longest runtime due to its hardware simulation, while LLMServingSim+ finishes much faster by removing the hardware simulation. For LLMServingSim2.0, even in the slowest case (MM), LLMServingSim2.0 finishes 1.94× faster than LLMServingSim+, due to a streamlined simulation workflow that minimizes unnecessary computation. Among the configurations, single-instance setups are fastest, followed by P/D disaggregation, and then multi-instance, with runtime increasing as system complexity grows. MoE is slower than the dense model because each layer must perform expert routing, which introduces additional overhead. Prefix caching, on the other hand, can either reduce or increase simulation time, since it accelerates request serving and decreases simulation iterations, but also incurs extra overhead from cache management. Overall, simulating 100 requests in under 12 minutes shows that LLMServingSim2.0 achieves practical simulation speed, even with the additional functionality integrated into the framework.

IV. Conclusion

This paper introduces LLMServingSim2.0, which brings modern LLM serving into a unified, accurate, and configurable simulator so that systems can be evaluated as they would run on scale-out deployments. By enabling controlled experiments on diverse hardware, disaggregation, MoE, parallelism, and caching within one framework, LLMServingSim2.0 lowers the barrier to rigorous methodology and accelerates progress in this field. Looking ahead, we will extend support for emerging devices such as Compute Express Link (CXL) to broaden the range of research questions it can address.

References

[1] J. Cho *et al.*, "LLMServingSim: A HW/SW Co-Simulation Infrastructure for LLM Inference Serving at Scale ," in *IISWC*, 2024, pp. 15–29.
[2] A. Agrawal *et al.*, "Vidur: A Large-Scale Simulation Framework For LLM Inference," 2024. [Online]. Available: https://arxiv.org/abs/2405.05465
[3] Y.-C. Lin *et al.*, "APEX: An Extensible and Dynamism-Aware Simulator for Automated Parallel Execution in LLM Serving," 2025. [Online]. Available: https://arxiv.org/abs/2411.17651
[4] F. Wu *et al.*, "TokenSim: Enabling Hardware and Software Exploration for Large Language Model Inference Systems," 2025. [Online]. Available: https://arxiv.org/abs/2503.08415
[5] P. Patel *et al.*, "Splitwise: Efficient Generative LLM Inference Using Phase Splitting," in *ISCA*, 2024, pp. 118–132.
[6] Y. Zhong *et al.*, "DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving," in *OSDI*, 2024, pp. 193–210.
[7] R. Hwang *et al.*, " Pre-gated MoE: An Algorithm-System Co-Design for Fast and Scalable Mixture-of-Expert Inference ," in *ISCA*, 2024, pp. 1018–1031.
[8] S. Yun *et al.*, " Duplex: A Device for Large Language Models with Mixture of Experts, Grouped Query Attention, and Continuous Batching ," in *MICRO*, 2024, pp. 1429–1443.
[9] W. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *SOSP*, 2023, p. 611–626.
[10] L. Zheng *et al.*, "SGLang: Efficient Execution of Structured Language Model Programs," 2024. [Online]. Available: https://arxiv.org/abs/2312.07104
[11] J. Yao *et al.*, "CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion," in *EuroSys*, 2025, p. 94–109.
[12] ShareGPT Team, "ShareGPT," https://sharegpt.com, 2023.