Frontier: Simulating the Next Generation of LLM Inference Systems

Yicheng Feng[1] Xin Tan[1] Kin Hang Sew[1] Yimin Jiang[2] Yibo Zhu[2] Hong Xu[1]

[1]*The Chinese University of Hong Kong* [2]*StepFun*

Abstract

Large Language Model (LLM) inference is growing increasingly complex with the rise of Mixture-of-Experts (MoE) models and disaggregated architectures that decouple components like prefill/decode (PD) or attention/FFN (AF) for heterogeneous scaling. Existing simulators, however, fall short in modeling the system-level complexities of distributed serving, and thus are unable to capture the intricate system dynamics of these emerging paradigms. We present Frontier, a high-fidelity simulator designed from the ground up for this new landscape. Frontier introduces a unified framework to model both co-located and disaggregated systems, providing native support for MoE inference with expert parallelism (EP). It enables the simulation of complex workflows like cross-cluster expert routing and advanced pipelining strategies for latency hiding. To ensure fidelity and usability, Frontier incorporates refined operator models for improved accuracy. Frontier empowers the community to design and optimize the future of LLM inference at scale.

1 Introduction

The demand for scalable and cost-effective Large Language Model (LLM) serving is driving a fundamental shift away from traditional, co-located deployments. The industry is actively exploring next-generation paradigms to improve performance and efficiency [2, 5, 8, 11, 12, 14, 16, 17]. These include *Mixture-of-Experts (MoE) models* [10–13], which employ sparse activation to scale parameter counts sub-linearly with compute costs, and *disaggregated architectures*. Disaggregation splits the inference process into distinct computational stages such as separating compute-intensive prefill from memory-bound decode [2, 16] (PD disaggregation) or decoupling Attention from FFN computations [14, 17] (AF disaggregation)—to optimize resource usage for each. Such approaches offer a promising path to improving the critical trade-off between system throughput and user-perceived interactivity, often referred to as the Pareto frontier [4, 12, 17].

While promising, these advanced architectures introduce unprecedented complexity. Designing a disaggregated system, for example, requires navigating a vast and intricate search space of model partitioning, concurrency control, and dynamic rate matching between specialized hardware pools. Similarly, MoE models introduce systemic challenges of token load imbalance across experts and expensive collective communication for result aggregation [11]. Optimizing these systems through empirical, trial-and-error experimentation on real hardware is prohibitively expensive and time-consuming, given the immense configuration space [4, 7, 12]. For example, identifying the optimal serving configuration for a standard 72B dense model in a 16-GPU co-located cluster setting can consume around 18,000 GPU-hours—amounting to a cost of over $93,000 [4].

High-fidelity simulation is a promising tool for tackling this complexity [4, 7]. However, state-of-the-art simulators like Vidur [4] are built around a replica-centric abstraction, which is fundamentally misaligned with the architecture of fully distributed and disaggregated LLM inference systems. This traditional design views the system as a pool of homogeneous, self-contained replicas, reducing the primary challenge to load-balancing requests among them. This core assumption is broken by disaggregated and MoE architectures, where inference is no longer a monolithic task but a multi-stage workflow orchestrated across specialized, heterogeneous clusters. The replica-centric model lacks the native primitives to represent this workflow, including inter-cluster routing, data transfer (e.g., KV-Cache), and complex synchronization. The critical abstraction has thus shifted from managing a pool of replicas to orchestrating the flow of a request through a distributed system—a concept prior simulators cannot natively represent.

Accurately simulating these new paradigms requires addressing three fundamental challenges.

First, the challenge of intra-node modeling fidelity: pushing the accuracy and completeness of operator-level modeling. The foundation of any simulator is its ability to accurately model the execution within a single computational graph, a domain where existing simulators have primarily focused [4, 7]. However, this foundation is cracking under the pressure of modern workloads and architectures. Two critical gaps have emerged: (1) Inaccurate modeling for modern workloads: The predictive power of existing operator models, particularly for `Attention`, degrades significantly on batches with high variance in sequence lengths. For instance, because Vidur's attention model oversimplifies the operator's runtime characteristics, we found that it can exhibit an error of over 55% (0.151ms vs. 0.340ms) for a single FlashAttention operation on a batch of 72 requests with skewed lengths. Such shortcomings in capturing operator workload dynamics can severely undermine simulation accuracy—particularly in multi-batch scenarios. (2) Incomplete modeling for new paradigms: Crucial computational patterns from emerging architectures, such as the heterogeneous `GroupedGEMM` in MoE models, are simply not accounted for, leaving a significant blind spot in performance prediction.

Second, the challenge of inter-node orchestration: creating a new simulation abstraction for distributed, multi-stage workflows. Even with perfect intra-node models, fully distributed systems fundamentally break the traditional replica-centric abstraction, recasting inference as a distributed workflow across specialized, independent clusters—a "system-of-systems". This demands a new simulation paradigm capable of modeling inter-node coordination. For instance, simulating PD disaggregation requires capturing the producer-consumer dynamics, where the prefill stage's output rate is constrained by the decode stage's memory availability via system-level backpressure [2, 12, 16]. Likewise, simulating AF disaggregation necessitates modeling a tightly-coupled, fine-grained pipeline, where the end-to-end latency is determined by the critical path of an event dependency graph that spans multiple clusters [12, 14, 17]. Existing simulators lack the native primitives to express these stateful, inter-dependent workflows, creating a critical gap in our ability to reason about the performance of disaggregated architectures.

Third, the challenge of system-level practical constraints: modeling the diverse and dynamic policies of real-world inference engines. A physically deployed system's performance is ultimately governed by the software policies of its serving engine. Different engines (e.g., vLLM [9], SGLang [15], TensorRT-LLM [3]) implement a wide array of strategies for dynamic batching, request scheduling, and memory management (e.g., PagedAttention [9]). These policies create complex, dynamic behaviors that significantly impact performance but are often abstracted away in current simulators. A truly practical simulation framework must treat these system-level policies as first-class citizens, allowing researchers to plug in, compose, and evaluate different strategies—from a specific batching algorithm to a novel memory management scheme.

To address these fundamental simulation challenges, we present *Frontier*, **the first simulation framework designed to systematically explore the design space of next-generation LLM inference systems, i.e., systems featuring disaggregated and MoE architectures**. Our primary contribution is a novel stage-centric simulation architecture that fundamentally departs from the traditional replica-based abstraction. This new abstraction provides the native primitives required to model complex, distributed workflows, enabling Frontier to capture critical inter-node dynamics. Frontier is grounded in a high-fidelity execution predictor capable of modeling data-dependent micro-workflows like MoE straggler effects, and is exposed through a modular framework with pluggable modules for exploring diverse, system-level policies from real-world inference engines. We plan to open source Frontier to the community.

Simulator	Disagg.		Parallelism			Sched.
	PD	AF	PP/TP	DP	EP	
LLMServingSim [7]	✗	✗	✓	✗	✗	✗
Vidur [4]	✗	✗	✓	✗	✗	–
Frontier (ours)	✓	✓	✓	✓	✓	✓

Table 1: Comparison of state-of-art inference simulators. PD: Prefill/Decode disaggregation, AF: Attention/FFN disaggregation, EP: Expert Parallelism. Sched: diverse advanced batching/memory scheduling. ✓: full support, ✗: no support. –: partial or conditional support.

2 Background and Motivation

2.1 The Next-Generation Inference Paradigms

To overcome the scaling and efficiency limitations of traditional co-located deployments, the field is converging on two primary architectural paradigms that restructure the inference process.

MoE architecture. MoE models replace dense FFN layers with numerous "expert" FFNs, only a subset of which are activated for each token by a router network [10–13]. This design allows for a massive increase in parameter count with only a sub-linear rise in computational cost.

Inference disaggregation. This paradigm exploits the distinct computational profiles of different inference phases by assigning them to specialized, independent hardware clusters. This is applied at a coarse grain by separating compute-bound prefill from memory-bandwidth-bound decode (PD disaggregation) [2, 16], or at a finer grain by decoupling the attention and FFN computations (AF disaggregation) [14, 17].

2.2 The Simulation Gap of Existing Simulators

Existing LLM inference simulators [4, 7] exhibit a substantial simulation gap when applied to emerging paradigms. This gap aligns with the challenges discussed in §1 and is evident in three key dimensions. Table 1 contrasts Frontier with prior simulators. Several intra-framework simulators [12,16], likely based on simplified roofline models, suffer from low fidelity.

3 Frontier Design

Frontier models the disaggregated and MoE inference based on the key designs and insights of mainstream frameworks (e.g., vLLM [9], SGLang [15], TensorRT-LLM [3]) and works (e.g., DistServe [16], Step-3 [14], MegaScale-Infer [17]). Frontier adheres to the event-driven and modular design principles established by Vidur [4].

3.1 Architecture Overview

The **Frontier Core**, depicted in Figure 1, is a hierarchical system designed to model the complex interactions within and between specialized compute clusters. The design comprises a central orchestration entity, the `GlobalController`, and a collection of modular `ClusterWorkers`, providing a principled framework for simulating the system-of-systems nature of modern LLM serving.

Global Controller. The `GlobalController` is the stateful orchestrator of inter-stage workflows, essential for modeling disaggregated systems. It manages the end-to-end lifecycle of requests by coordinating events between independent `ClusterWorkers`, supported by integrated modules for workload generation and performance data. Its key role is managing complex, state-dependent interactions: in PD disaggregation, it models system-level backpressure by initiating KV-Cache transfers only upon receiving memory availability signals; in AF disaggregation, it orchestrates the event dependency graph for the fine-grained pipeline.

Cluster Worker. A `ClusterWorker` is the fundamental abstraction for a specialized hardware cluster (e.g., a prefill or attention cluster), containing a `ClusterScheduler` and a pool of `ReplicaWorkers`. The `ClusterScheduler` manages local resources and participates in inter-stage coordination, such as signaling memory availability for pull-based transfers in PD disaggregation or managing micro-batch handoffs in the AF pipeline.

Replica Worker. The `ReplicaWorker` simulates a single model instance, with its core logic encapsulated in the `Execution Predictor`. Moving beyond monolithic operators, the predictor's key feature is its ability to decompose a logical layer into a data-dependent micro-workflow of events. This is critical for MoE simulation, where it models the gating decision to generate a token-to-expert assignment map and simulates expert computation as a set of heterogeneous tasks. By taking the maximum of these varied task times, the `ExecutionPredictor` natively captures the performance impact of token load imbalance and the resulting straggler effects.

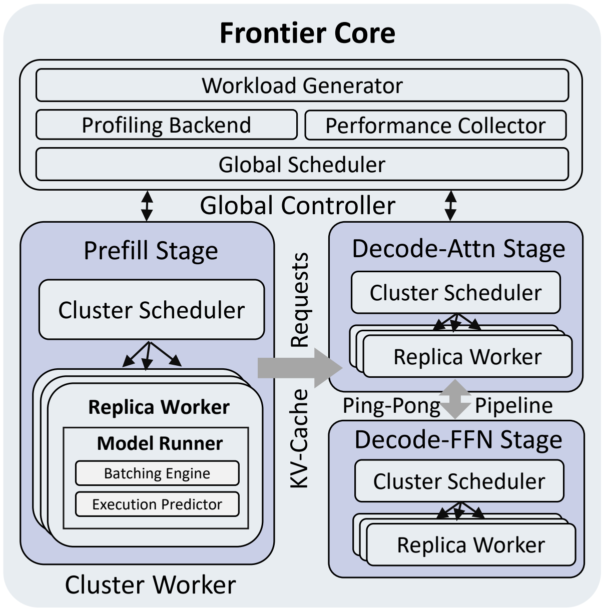

Figure 1: Frontier architecture overview.

3.2 Accurate Operator Runtime Prediction

Challenges. Unlike simple matrix multiplication (e.g., GEMM), where runtime mainly depends on input size, many operators have more complex input characteristics. For example, `Attention` can involve widely varying sequence lengths within a batch, and `GroupedGEMM` often faces imbalanced internal workloads, making runtime prediction significantly more challenging. Existing methods like Vidur simplify estimation by using a single proxy length (typically the square root of batch sequence lengths), but this overlooks important factors affecting actual kernel execution. In practice, kernel execution involves partitioning and tiling computations—processes that become much more complex and less efficient with input heterogeneity, leading to phenomena such as wave quantization.

Finer-grained modeling. To overcome these challenges, we employ fine-grained modeling tailored to the computation patterns and input characteristics of specific operators. For `Attention`, we utilize a rich set of features—including aggregate and distributional statistics of sequence lengths—to train an ML model (e.g. random forest [6]) that more accurately captures workload dynamics, particularly under high input variance. Likewise, for `GroupedGEMM` in MoE, we extract features that reflect both input properties and expert load distribution, such as token counts, expert number, model dimensions, expert selection ratios, and various load balance metrics. Such a comprehensive approach yields robust and precise predictions, even for workloads with highly variable characteristics.

3.3 Modeling Disaggregated and MoE Infer.

PD disaggregation workflow simulation. The fundamental problem of simulating PD disaggregation is to *accurately model the producer-consumer dynamics between two specialized, rate-mismatched subsystems*. The core challenge is capturing the system-level coordination and backpressure required to balance these two stages under dual SLOs.

Frontier is designed to model these intricate system dynamics. For any given PD configuration (i.e., a fixed number of PD instances with specific parallelism), Frontier simulates the end-to-end request lifecycle with high fidelity through a stateful, event-driven workflow: (1). Frontier models the prefill stage as a producer. When requests arrive, the `GlobalController` routes them to the prefill stage. The `ClusterScheduler` and `ReplicaWorker` simulate its queuing and execution. Upon completion, the worker signals to the `Global Controller`, which transitions the request's state to `PREFILL_COMPLETE`. At this point, the generated KV-Cache is conceptually held in the prefill stage's memory buffer. (2). The decode stage is modeled as a consumer with a finite resource: GPU memory for KV-Caches. The `ClusterScheduler` of the decode stage continuously tracks its memory utilization. When a decoding request completes and its KV-Cache is evicted, the scheduler signals its updated memory availability to the `GlobalController`. (3). The `GlobalController` acts as the central coordinator that respects backpressure. It maintains a queue of `PREFILL_COMPLETE` requests. It will initiate a `KV_CACHE_TRANSFER` event for a request.

AF disaggregation workflow simulation. The core simulation challenge, as highlighted by systems like MegaScale-Infer and Step-3, is to *accurately capture the critical path of a multi-stage, micro-batch-driven workflow*, where even small imbalances between stages can create significant performance-degrading pipeline bubbles.

Frontier addresses this by simulating the AF workflow as an event dependency graph. For any given AF configuration, Frontier models the generation of a single token by orchestrating a complex graph of fine-grained events. (1). The simulation begins when the `GlobalController` initiates a decode step. The `ReplicaWorker` in the decode-attn stage first partitions this global batch into a series of m simulated micro-batches. (2). Frontier's `GlobalController` and `ClusterSchedulers` dynamically construct a dependency graph for all operations across L model layers. Frontier's event-driven engine processes this graph by scheduling events as soon as their dependencies are met. This inherently simulates the overlap: while `A_TO_F_TRANSFER(i,k)` is in flight, the simulator can schedule `ATTN_COMPUTE(i+1,k)` on the now-free attention GPU, perfectly capturing the latency-hiding principle of the ping-pong pipeline. (3). The total time to generate one token is determined by the timestamp of the final event in the graph—typically `FFN_COMPUTE(m, L)`—completing.

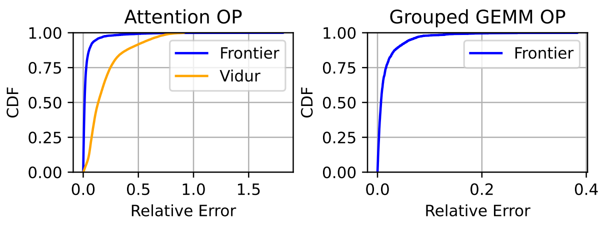

Figure 2: CDF of the relative error in simulated operator runtime under dynamic workloads.

Batch Size	Avg Input	Output	Profiled throughput	Predicted throughput
4	32	1024	111.355	90.498
8	128	256	131.831	109.366
16	256	128	151.425	127.157
32	32	128	313.236	240.743

Table 2: End-to-end performance (throughput in tokens/s/GPU).

MoE inference workflow simulation. Simulating MoE inference introduces a unique challenge: *modeling a dynamic, data-dependent workflow* where performance is dictated not by average-case behavior, but by the worst-case straggler caused by token load imbalance.

Frontier addresses these challenges by decomposing the MoE layer execution into a detailed, multi-step micro-workflow within the `ReplicaWorker`, ensuring that the effects of imbalance are modeled with high fidelity. (1). Frontier first configures the virtual model sharding to satisfy the system's topological constraints (e.g., `attn_dp*attn_tp==moe_tp*moe_ep`). (2). When the `ExecutionPredictor` encounters an MoE layer, it simulates the following sequence of events, explicitly tracking the consequences of token routing. The simulation first models the GEMM for the gating network. Subsequently, a pluggable routing module is invoked. Frontier simulates the routing decision to generate a token-to-expert assignment map for the current batch. With the assignment map, the simulation of expert computation becomes heterogeneous. The `ExecutionPredictor` queries GroupedGEMM performance model with the actual number of tokens assigned to it for each expert i. (3). After that, Frontier simulates synchronization and straggler effects. Frontier models the implicit synchronization barrier by calculating the latency as `max[T_expert1,T_expert2,...,T_expertN]`.

4 Preliminary Evaluation

Setup. Experiments are performed on an 8-GPU node featuring NVIDIA A800-SXM4-80GB GPUs with 400 GB/s NVLink interconnects. The profiling and training environment is configured with PyTorch 2.3, CUDA 12.1, Ray 2.42.1, and FlashInfer 0.1.6. End-to-end evaluation is conducted using vLLM 0.10.1 with the SharedStorageConnector KV interface. We use the Qwen2-7B-Instruct model [1].

Operator accuracy. We evaluate the accuracy of Frontier on two critical operators: `Attention` and `GroupedGEMM`. These operators are highly sensitive to variable inputs and prone to inaccuracies. For the `Attention` operator, as shown in Figure 2, Frontier consistently outperforms Vidur, achieving significantly lower relative errors, with over 94% of cases falling below 10%. For the `GroupedGEMM` operator, which is not supported by Vidur, we report results solely for Frontier. Frontier demonstrates high accuracy, with over 95% of errors remaining below 6%.

End-to-End accuracy. We validate the end-to-end accuracy of Frontier by simulating a PD disaggregated system with a 1:1 ratio of prefill to decode instances. As shown in Table 2, we compare the predicted system output throughput against the profiled performance of a real system across a range of batch sizes and sequence lengths. The results demonstrate that Frontier captures performance trends, with the predicted throughput (in tokens/s/GPU) consistently falling within a 19.0% to 23.2% relative error margin across tested cases.

5 Discussion and Conclusion

We introduced Frontier, a high-fidelity simulator tailored for emerging LLM inference architectures, including disaggregated and MoE systems. Future work will expand on modeling core operators, quantifying simulation fidelity and cost, and demonstrating Frontier's utility through diverse case studies for large-scale system design and optimization.

References

[1] Qwen2 technical report. 2024.

[2] Nvidia dynamo. Website, 2025. https://github.com/ai-dynamo/dynamo.

[3] Nvidia tensorrt-llm. Website, 2025. https://github.com/NVIDIA/TensorRT-LLM.

[4] Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A large-scale simulation framework for llm inference. *Proceedings of Machine Learning and Systems*, 6:351–366, 2024.

[5] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, 2024.

[6] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] Jaehong Cho, Minsu Kim, Hyunmin Choi, Guseul Heo, and Jongse Park. Llmservingsim: A hw/sw co-simulation infrastructure for llm inference serving at scale. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*, pages 15–29. IEEE, 2024.

[8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[9] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[10] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed {MoE} training and inference with lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 945–959, 2023.

[11] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[12] Tiyasa Mitra, Ritika Borkar, Nidhi Bhatia, Ramon Matas, Shivam Raj, Dheevatsa Mudigere, Ritchie Zhao, Maximilian Golub, Arpan Dutta, Sailaja Madduri, et al. Beyond the buzz: A pragmatic take on inference disaggregation. *arXiv preprint arXiv:2506.05508*, 2025.

[13] Siddharth Singh, Olatunji Ruwase, Ammar Ahmad Awan, Samyam Rajbhandari, Yuxiong He, and Abhinav Bhatele. A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training. In *Proceedings of the 37th International Conference on Supercomputing*, pages 203–214, 2023.

[14] StepFun, :, Bin Wang, Bojun Wang, Changyi Wan, Guanzhe Huang, Hanpeng Hu, Haonan Jia, Hao Nie, Mingliang Li, Nuo Chen, Siyu Chen, Song Yuan, Wuxun Xie, Xiaoniu Song, Xing Chen, Xingping Yang, Xuelin Zhang, Yanbo Yu, Yaoyu Wang, Yibo Zhu, Yimin Jiang, Yu Zhou, Yuanwei Lu, Houyi Li, Jingcheng Hu, Ka Man Lo, Ailin Huang, Binxing Jiao, Bo Li, Boyu Chen, Changxin Miao, Chang Lou, Chen Hu, Chen Xu, Chenfeng Yu, Chengyuan Yao, Daokuan Lv, Dapeng Shi, Deshan Sun, Ding Huang, Dingyuan Hu, Dongqing Pang, Enle Liu, Fajie Zhang, Fanqi Wan, Gulin Yan, Han Zhang, Han Zhou, Hanghao Wu, Hangyu Guo, Hanqi Chen, Hanshan Zhang, Hao Wu, Haocheng Zhang, Haolong Yan, Haoran Lv, Haoran Wei, Hebin Zhou, Heng Wang, Heng Wang, Hongxin Li, Hongyu Zhou, Hongyuan Wang, Huiyong Guo, Jia Wang, Jiahao Gong, Jialing Xie, Jian Zhou, Jianjian Sun, Jiaoren Wu, Jiaran Zhang, Jiayu Liu, Jie Cheng, Jie Luo, Jie Yan, Jie Yang, Jieyi Hou, Jinguang Zhang, Jinlan Cao, Jisheng Yin, Junfeng Liu, Junhao Huang, Junzhe Lin, Kaijun Tan, Kaixiang Li, Kang An, Kangheng Lin, Kenkun Liu, Lei Yang, Liang Zhao, Liangyu Chen, Lieyu Shi, Liguo Tan, Lin Lin, Lin Zhang, Lina Chen, Liwen Huang, Liying Shi, Longlong Gu, Mei Chen, Mengqiang Ren, Ming Li, Mingzhe Chen, Na Wang, Nan Wu, Qi Han, Qian Zhao, Qiang Zhang, Qianni Liu, Qiaohui Chen, Qiling Wu, Qinglin He, Qinyuan Tan, Qiufeng Wang, Qiuping Wu, Qiuyan Liang, Quan Sun, Rui Li, Ruihang Miao, Ruosi Wan, Ruyan Guo, Shangwu Zhong, Shaoliang Pang, Shengjie Fan, Shijie Shang, Shilei Jiang, Shiliang Yang, Shiming Hao, Shuli Gao, Siming Huang, Siqi Liu, Tiancheng Cao, Tianhao Cheng, Tianhao Peng, Wang You, Wei Ji, Wen Sun, Wenjin Deng, Wenqing He, Wenzhen Zheng, Xi Chen, Xiangwen Kong, Xianzhen Luo, Xiaobo Yang, Xiaojia Liu, Xiaoxiao Ren, Xin Han, Xin Li, Xin Wu, Xu Zhao, Yanan Wei, Yang Li, Yangguang Li, Yangshijie Xu, Yanming Xu, Yaqiang Shi, Yeqing Shen, Yi Yang, Yifei Yang, Yifeng Gong, Yihan Chen, Yijing Yang, Yinmin Zhang, Yizhuang Zhou, Yuanhao Ding, Yuantao Fan, Yuanzhen Yang, Yuchu Luo, Yue Peng, Yufan Lu, Yuhang Deng, Yuhe Yin, Yujie Liu, Yukun Chen, Yuling Zhao, Yun Mou, Yunlong

Li, Yunzhou Ju, Yusheng Li, Yuxiang Yang, Yuxiang Zhang, Yuyang Chen, Zejia Weng, Zhe Xie, Zheng Ge, Zheng Gong, Zhenyi Lu, Zhewei Huang, Zhichao Chang, Zhiguo Huang, Zhirui Wang, Zidong Yang, Zili Wang, Ziqi Wang, Zixin Zhang, Binxing Jiao, Daxin Jiang, Heung-Yeung Shum, and Xiangyu Zhang. Step-3 is large yet affordable: Model-system co-design for cost-effective decoding, 2025.

[15] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

[16] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, 2024.

[17] Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, et al. Megascale-infer: Serving mixture-of-experts at scale with disaggregated expert parallelism. *arXiv preprint arXiv:2504.02263*, 2025.