

Rethinking LLM Deployment for Intent-Based Serving

Dimitrios Liakopoulos
The University of Texas at Austin

Prasoon Sinha
The University of Texas at Austin

Tianrui Hu
The University of Texas at Austin

Myungjin Lee
Cisco Systems

Neeraja J. Yadwadkar
The University of Texas at Austin

Abstract

Large Language Models (LLMs) expose a large deployment configuration space spanning parallelism and compression techniques, with each configuration introducing different tradeoffs in latency, memory consumption, cost, and output quality. Existing systems either rely on expensive profiling across deployment configurations or inefficiently utilize fragmented GPU resources in multi-tenant clusters. We present *MaverIQ*, an intent-based LLM inference serving system that automatically maps user intents to deployment configurations while minimizing operational cost for the provider. To reduce profiling overheads, *MaverIQ* introduces lightweight LLM *fingerprints* and analytical models that extrapolate latency and memory footprint from only a few observations. To efficiently utilize fragmented GPU resources, *MaverIQ* leverages our observation that, unlike training, unevenly distributing LLM layers across GPUs has little impact on inference latency. Our evaluation shows that *MaverIQ* reduces profiling cost by 7-15 $\times$ compared to state-of-the-art baselines and reduces operational cost by 3.8-8.3 $\times$ across diverse LLMs, traces, and loads while effectively meeting user intents. Our code is available at <https://github.com/UT-SysML/MaverIQ>.¹

1 Introduction

Large Language Models (LLMs) are increasingly deployed across applications with diverse serving objectives. Real-time assistants prioritize low latency, while educational platforms and automated FAQs emphasize cost efficiency. To satisfy these diverse intents, practitioners rely on deployment configurations that combine compression techniques (e.g., quantization [6, 7, 10, 22, 45, 46], pruning [9, 25, 42]) and parallelism techniques (e.g., data [39], pipeline [13], tensor parallelism [44]). These configurations vary significantly in latency, memory consumption, GPU usage, and serving cost. For example, Llama-2-70B can be deployed in 180 ways on

an 8-GPU cluster, resulting in up to 8 $\times$ variation in inference latency and 25.4 $\times$ variation in user cost. Thus, *the choice of deployment configuration is critical for meeting diverse user intents*. As AI-native systems increasingly serve diverse and dynamic workloads, deployment decisions become central to efficiently utilizing shared GPU infrastructure.

Existing systems struggle to navigate this large configuration space efficiently. Some systems fix deployment configurations across models [8, 32, 48], while others require users to manually specify deployment parameters [4, 5, 18, 23, 29, 33]. More recent approaches use profiling to map user intents to deployment configurations [1, 19, 20, 26, 32, 37, 41], but incur substantial overheads by profiling full LLMs across many configurations [1, 20]. For example, profiling a subset of Llama2-70B’s configurations took us ~ 20 GPU hours. At the same time, previous systems inefficiently utilize fragmented GPU resources because they evenly distribute LLMs across GPUs [2, 4, 12, 18, 20, 29]. Thus, we need new techniques that *efficiently estimate deployment tradeoffs while utilizing fragmented GPU resources*.

To this end, we build *MaverIQ*, an automated intent-based LLM inference serving system that translates user-expressed intents into deployment configurations while reducing operational cost for the provider. To reduce profiling overheads, we make a key observation that commonly used LLMs contain many repeated layers and only a few unique layers. This repeated structure enables us to replace the full LLM with a compact proxy, which we introduce as an LLM *fingerprint*, that retains the unique architectural components of the model while containing only a few representative hidden layers. *MaverIQ* further combines these lightweight fingerprints with analytical models to accurately estimate latency and memory footprint across deployment configurations from only a few observations. As a result, *observing lightweight fingerprints under a small number of configurations is sufficient to efficiently estimate the implications of deployment configurations for full LLMs*.

To efficiently utilize fragmented GPU resources, we make another key observation: unlike training, LLM inference does

¹This article summarizes the key ideas and findings from our full SC’25 paper on *MaverIQ* [21].

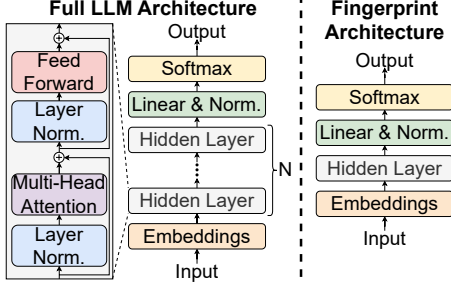


Figure 1: Architecture of a decoder-only LLM (left) and its fingerprint (right). The fingerprint has the same components as the full LLM, but only 1-2 hidden layers. See § 2.

not require balanced layer placement across GPUs. During training, evenly distributing layers across GPUs helps reduce synchronization overheads introduced by forward and backward passes. Inference, however, only performs a forward pass. As a result, *unevenly distributing layers across GPUs introduces little synchronization overhead while enabling more efficient utilization of fragmented GPU resources in multi-tenant clusters*. *MaverIQ* exploits this insight to improve cluster utilization and reduce operational cost while preserving inference latency.

We summarize our contributions as follows:

- We introduce LLM fingerprints, compact proxies that capture the essential structure of an LLM and enable lightweight profiling of deployment configurations.
- We develop analytical models that accurately estimate the latency and memory footprint of LLM deployment configurations from only a few fingerprint observations, reducing profiling cost by 7-15 $\times$ while reducing estimation error by 1.3-1.7 $\times$.
- We show that we can unevenly distribute LLM layers across GPUs to utilize fragmented resources without harming inference latency. This technique reduces operational cost by up to 2 $\times$.
- We build *MaverIQ*, an intent-based LLM inference serving system that leverages fingerprint-guided extrapolation and fragmentation-aware deployment to efficiently utilize GPU resources while meeting diverse user intents. Under strict accuracy requirements, *MaverIQ* reduces latency by 28-45% for users while reducing operational cost by 3.8-8.3 $\times$ for providers.

2 Background

Model architecture of LLMs. Modern LLMs primarily follow decoder-only transformer architectures [38], consisting of hundreds of repeated layers that process previously generated tokens to produce outputs of arbitrary length. This repeated structure motivates our use of lightweight LLM *fingerprints*, compact proxies that retain the key architectural components of the full model while containing only a few hidden layers (Figure 1).

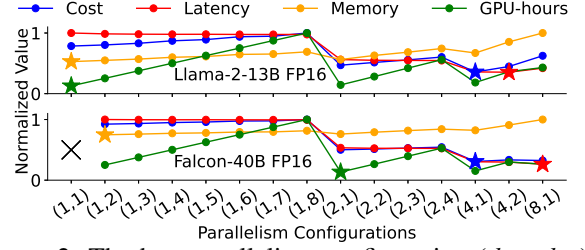


Figure 2: The best parallelism configuration (d_{TP} , d_{PP}) is user-intent- and LLM-specific. d_{TP} , d_{PP} : degrees of tensor and pipeline parallelism. See § 3.

Deployment configurations for LLMs. LLM deployment configurations combine parallelism and compression techniques. Parallelism techniques distribute LLMs across GPUs using data parallelism (DP) [39], tensor parallelism (TP) [44], and pipeline parallelism (PP) [13]. Compression techniques reduce memory footprint through quantization [6, 7, 10, 22, 45, 46] and pruning [9, 25, 36, 42]. Throughout this paper, we refer to an LLM deployment configuration as a selection of TP and PP degrees together with the applied compression techniques.

3 Motivation & Characterization

Deployment configurations affect latency, memory consumption, GPU usage, and operational cost differently across LLMs. Using an 8-GPU cluster with NVIDIA RTX A6000 GPUs and six state-of-the-art LLMs, we characterize the effects of parallelism and compression techniques on these deployment objectives.

1. Effect of Parallelism. Figure 2 shows the impact of tensor parallelism (TP) and pipeline parallelism (PP) on latency, memory consumption, cost, and GPU-hours. Increasing PP degree has minimal impact on inference latency, while reducing per-GPU memory footprint. In contrast, increasing TP degree reduces latency but increases memory consumption and GPU-hours due to synchronization overheads [40]. Moreover, the optimal combination of TP and PP varies across LLMs and user intents. For example, configurations that minimize latency do not necessarily minimize memory consumption or cost.

Takeaway #1. Increasing PP inflates total memory footprint but lowers per-GPU memory usage, while increasing TP reduces inference latency at the cost of higher synchronization overheads. *The best TP/PP configuration is LLM- and intent-specific.*

2. Effect of Unbalanced Pipeline Parallelism. Pipeline parallelism is known to introduce stalls during training because GPUs must synchronize forward and backward passes [13, 27]. As a result, training systems evenly distribute layers across GPUs [24]. Inference, however, only requires a forward pass [11]. We observe that pipeline stalls during inference are significantly reduced compared to training, and unevenly

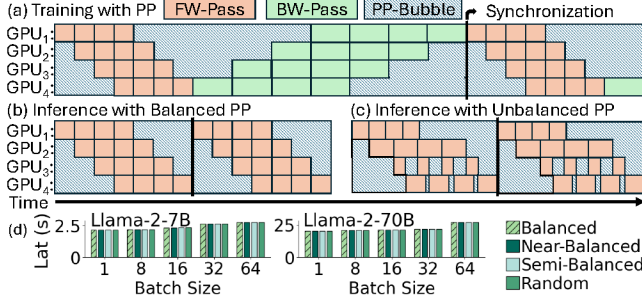


Figure 3: Unlike LLM training, the impact of unbalanced PP on latency is negligible in LLM inference. See § 3.

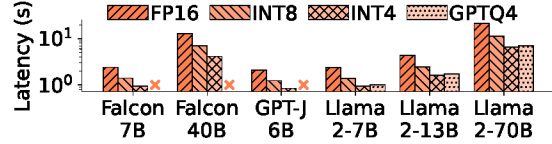


Figure 4: The impact of quantization on latency is LLM-specific. See § 3.

distributing layers across GPUs has negligible impact on inference latency (Figure 3). Instead, unbalanced PP changes the memory footprint across PP partitions, enabling more efficient utilization of fragmented GPU memory.

Takeaway #2. We can leverage unbalanced PP during inference to utilize fragmented GPU memory and improve cluster throughput without degrading inference latency.

3. Effect of Quantization. Quantization predictably reduces memory consumption across LLMs. For example, INT8 reduces memory consumption by 50%, while INT4/GPTQ4 reduces it by 75% compared to FP16. Quantization also reduces inference latency, although its impact varies across LLMs (Figure 4). However, these reductions in latency and memory footprint come at the expense of output quality [10, 17].

Takeaway #3. Quantization presents a tradeoff between memory consumption, latency, and output quality, and its impact is both LLM- and intent-specific.

4. Effect of Pruning. Pruning techniques reduce memory footprint by removing or sparsifying model weights [9, 25, 42]. However, pruning can also increase inference latency due to hardware inefficiencies and matrix conversion overheads, with effects varying across LLMs. Moreover, pruning degrades output quality [25, 42].

Takeaway #4. Pruning introduces tradeoffs between accuracy, latency, and memory consumption, and its effects vary across LLMs.

4 MaverIQ Design

Given a user intent, MaverIQ identifies a deployment configuration for an LLM without expensive profiling and deploys it in a cost-efficient manner by utilizing fragmented GPU

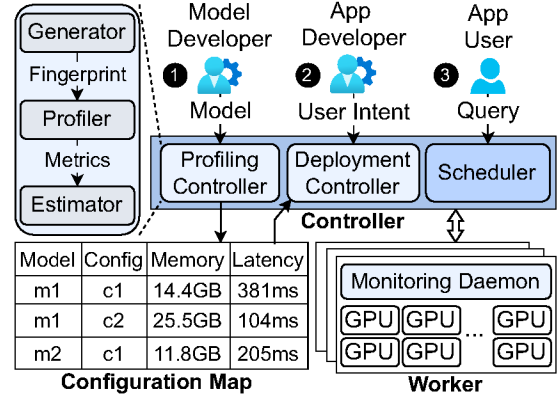


Figure 5: Architecture of MaverIQ. See § 4.

Table 1: MaverIQ’s APIs. See § 4.

API	Parameters
register_model	model, [compressionTechnique: accuracy]
deploy_model	model, maxOutputLength, intent <minCost, minLat, latSLO, costSLO, accSLO>
inference_model	model, inputText

resources. Figure 5 presents the architecture of MaverIQ.

Intent-based interface. MaverIQ exposes an intent-based interface that allows users to deploy LLMs without specifying deployment configurations or GPU allocations. Users register models together with supported compression techniques and accuracy scores, and specify intents such as minimizing latency or cost, or meeting latency, cost, and accuracy SLOs (Table 1).

Profiling Controller. The Profiling Controller translates user intents into deployment configurations using lightweight LLM fingerprints and analytical models that estimate latency and memory footprint across deployment configurations.

Deployment Controller. The Deployment Controller deploys selected configurations while utilizing fragmented GPU resources. It leverages unbalanced LLM partitioning together with load-aware and fragmentation-aware placement policies to reduce operational cost while meeting user intents.

Scheduler. The Scheduler routes inference requests to workers hosting deployed LLMs. MaverIQ uses a simple FCFS scheduler and is designed modularly to support future scheduling policies.

4.1 Key Workflows.

We now describe MaverIQ’s three key workflows.

1 Profiling workflow. When an LLM is registered, MaverIQ generates a lightweight fingerprint of the model, profiles it under a small number of deployment configurations, and uses the resulting observations together with analytical models to estimate latency and memory footprint for the full configuration space. Section 5 details MaverIQ’s profiling that is inexpensive yet accurate.

Config	Per GPU Balanced Footprint	Total Memory
No Parallel	W A	$\text{mem}_{\text{base}} = (W+A)$
TP	W/d_{TP} A $\times d_{TP}$	$\text{mem}_{\text{base}} + (d_{TP}-1) \times A$
PP	W/d_{pp} A/d_{pp} A_p $\times d_{pp}$	$\text{mem}_{\text{base}} + (d_{pp}-1) \times A_p$
TP + PP	$W/d_{TP \cdot d_{pp}}$ A/d_{pp} A_p $\times d_{TP \cdot d_{pp}}$	$\text{mem}_{\text{base}} + (d_{TP}-1) \times A + d_{TP} \times (d_{pp}-1) \times A_p$

Figure 6: LLM parallelism memory overheads. Weights (W), activations (A), and pipeline activations (A_p) are replicated on d_{TP} and/or d_{pp} GPUs. See § 5.2.

② **Model-deployment workflow.** Given a user intent, the Deployment Controller selects a feasible deployment configuration and maps the LLM across GPUs using load-aware and fragmentation-aware placement policies. Section 6 describes *MaverIQ*’s cost-efficient deployment policies.

③ **Inference-serving workflow.** When users submit inference requests, the Scheduler dispatches them to worker nodes hosting the deployed LLMs.

5 Inexpensive yet Accurate Profiling

MaverIQ’s Profiling Controller estimates an LLM’s latency and memory footprint across deployment configurations without exhaustively profiling the full model. To do so, *MaverIQ* combines lightweight LLM *fingerprints* with analytical models that extrapolate deployment characteristics from a small number of observations.

5.1 LLM Fingerprints

Profiling full LLMs across deployment configurations is prohibitively expensive, requiring substantial GPU memory and profiling time. We observe that modern LLMs contain many repeated hidden layers. Thus, instead of profiling the full LLM, *MaverIQ* profiles a compact proxy of the model, which we refer to as an LLM *fingerprint*.

Fingerprints preserve the key architectural components of the original model while containing only a few hidden layers (Figure 1). As a result, they substantially reduce profiling overheads: Llama-2-70B’s fingerprint is $41\times$ smaller and $63\times$ faster to profile than the full model. We further observe a linear relationship between the number of hidden layers and the LLM’s memory footprint, enabling extrapolation from fingerprints to full LLMs.

5.2 Estimating Memory

Parallelism techniques increase an LLM’s memory footprint by replicating weights, activations, and pipeline activations across GPUs (Figure 6). Without parallelism, the memory footprint consists of weights and activations. TP increases memory footprint through replicated activations, while PP introduces additional pipeline activation overheads. Thus, estimating the weights (W), activations (A), and pipeline activations (A_p) is sufficient to estimate the memory footprint

across deployment configurations.

MaverIQ explores two memory profiling strategies.

M1. LLM under selected configurations. M1 profiles the full LLM under three reference configurations and estimates W , A , and A_p by solving a system of equations derived from the parallelism overheads.

M2. Fingerprint under selected configurations. M2 instead profiles lightweight fingerprints under the same reference configurations and extrapolates the resulting measurements to the full LLM. Compared to profiling full LLMs, M2 substantially reduces the resources consumed during profiling.

5.3 Estimating LLM Inference Latency

LLM inference latency depends on the output length and can be decomposed into the time to first token (TTFT) and time per output token (TPOT):

$$L = TTFT + \text{output_length} \times TPOT \quad (1)$$

Since TTFT and TPOT vary across deployment configurations, naively estimating latency would require profiling all configurations. Instead, *MaverIQ* develops analytical models that capture the effects of TP and PP on inference latency.

Increasing TP distributed the work across multiple GPUs, thus reducing the overall computation latency. However, TP introduces a communication overhead due to the all-reduce operations that is non-linear with respect to d_{TP} . Increasing PP partitions layers across GPUs, thus slightly reducing the overall computation latency. Similar to TP, PP introduces a communication overhead due to the send-recv operations, which is non-linear with respect to d_{pp} . In addition, PP reduces the amount of data that traverses the GPU-interconnects, thus reducing the TP overhead non-linearly with respect to d_{pp} . *MaverIQ* models these effects as:

$$L(d_{TP}, d_{pp}) = \frac{L(1,1)}{d_{TP} \times d_{pp}^\alpha} + \frac{(d_{TP}-1)^\beta}{d_{pp}^\gamma} \times O_{TP} + (d_{pp}-1)^\delta \times O_{PP} \quad (2)$$

where α , β , γ , and δ capture the scaling behavior of TP and PP, while O_{TP} and O_{PP} model synchronization and pipeline overheads. *MaverIQ* determines these constants using a one-time fitting process across diverse LLMs and configurations.

MaverIQ explores two latency profiling strategies.

L1. LLM under selected configurations. L1 profiles the full LLM under three reference configurations to estimate TTFT and TPOT.

L2. Fingerprint under selected configurations. L2 instead profiles fingerprints and extrapolates TTFT and TPOT to the full LLM using layer-wise decomposition:

$$TTFT = \text{num_layers} \times TTFT_{\text{layer}} + TTFT_{\text{other}} \quad (3)$$

$$TPOT = \text{num_layers} \times TPOT_{\text{layer}} + TPOT_{\text{other}} \quad (4)$$

Using these formulations, *MaverIQ* extrapolates latency from fingerprints to full LLMs across deployment configurations.

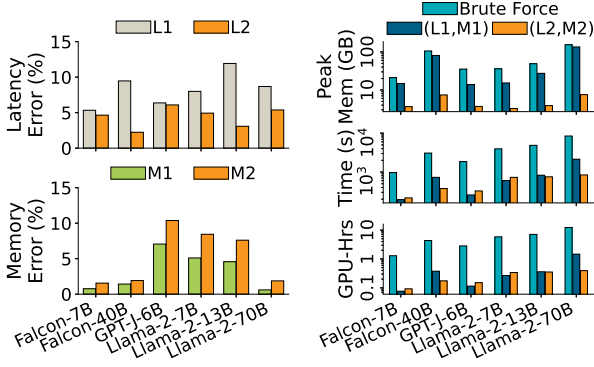


Figure 7: Design exploration of profiling methods. MaverIQ uses fingerprint-based profiling (L2,M2) to reduce profiling cost while maintaining similar estimation accuracy. See § 5.4.

5.4 Design Exploration of Profiling Techniques

We compare profiling methods that use full LLMs with methods that use fingerprints. Fingerprint-based profiling substantially reduces profiling memory consumption, time, and GPU-hours while maintaining similar estimation accuracy (Figure 7). For larger LLMs, fingerprint-based profiling reduces profiling time and GPU-hours by up to $2.5\times$ and $3.8\times$, respectively. Thus, MaverIQ uses fingerprint-based profiling due to its substantially lower profiling cost.

5.5 Support for Various Batch Sizes

MaverIQ also supports profiling larger batch sizes. We observe that latency and memory overheads scale linearly with batch size:

$$L(BS) = L(1) + (BS - 1) \times O_{BS}^L \quad \& \quad M(BS) = M(1) + (BS - 1) \times O_{BS}^M \quad (5)$$

where $L(BS)$ and $M(BS)$ denote latency and memory footprint for batch size BS . MaverIQ estimates these overheads using the same profiling methodology developed for batch size one.

6 Cost-Efficient Deployment

MaverIQ’s Deployment Controller deploys selected configurations while minimizing operational cost in multi-tenant GPU clusters with fragmented resource availability. Unlike training, inference only requires a forward pass, and unevenly distributing LLM layers across GPUs has little impact on inference latency. MaverIQ leverages this insight to efficiently utilize fragmented GPU resources while meeting user intents.

Overview. Using the profiling-based estimations (Section 5), MaverIQ ranks deployment configurations according to the specified user intent. It then selects the first feasible configuration using two strategies: load-aware GPU selection and fragmentation-aware layer mapping. We describe them below.

Load-aware GPU selection. For a deployment configuration requiring $d_{TP} \times d_{PP}$ GPUs, MaverIQ identifies GPU sets whose free memory satisfies the deployment requirements.

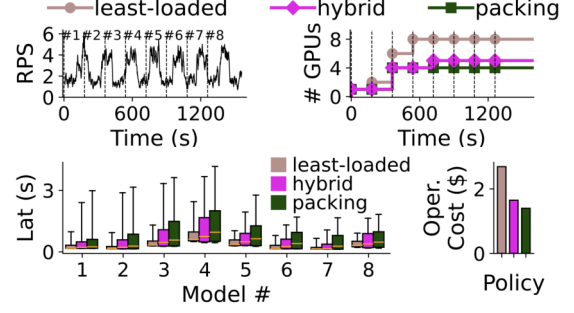


Figure 8: Design exploration of placement policies. The hybrid policy balances GPU utilization and inference latency. See § 6.

It then selects GPUs using one of three placement policies: least-loaded, packing, or hybrid.

Least-loaded disperses LLMs across GPUs to reduce contention and latency. Packing colocates LLMs on the same GPUs to minimize the number of GPUs consumed. Hybrid balances these objectives by packing LLMs when GPU load is low and dispersing them when load increases.

Figure 8 shows that least-loaded reduces latency but increases operational cost, while packing minimizes GPU usage at the expense of higher contention and latency. Hybrid balances these tradeoffs and achieves the best overall balance between latency and operational cost. Thus, MaverIQ uses the hybrid policy by default.

Fragmentation-aware layer mapping. With the selected configuration and GPUs, MaverIQ maps LLM layers across GPUs using fragmentation-aware layer placement. The minimum deployment unit is a memory block corresponding to a single layer:

$$mem_{block} = \frac{mem_{total}}{d_{TP} \times num_{layers}}$$

where mem_{total} is the estimated memory footprint and num_{layers} is the number of hidden layers in the LLM.

MaverIQ attempts to keep adjacent layers close and distribute layers evenly across GPUs. However, when GPU memory is fragmented, MaverIQ unevenly distributes layers across GPUs to utilize fragmented resources. Because unbalanced partitioning has little impact on inference latency during inference, this approach enables MaverIQ to deploy more LLMs on fewer GPUs, improving cluster throughput while reducing operational cost.

7 Implementation

We implement MaverIQ in Python atop TensorRT-LLM [29], a widely used LLM inference serving framework [28, 30, 31]. MaverIQ’s Controller is a persistent process that manages profiling, deployment, and request routing.

Profiling Controller. The Profiling Controller generates lightweight fingerprints from LLM architecture and weight files using PyTorch APIs [14]. It profiles fingerprints under

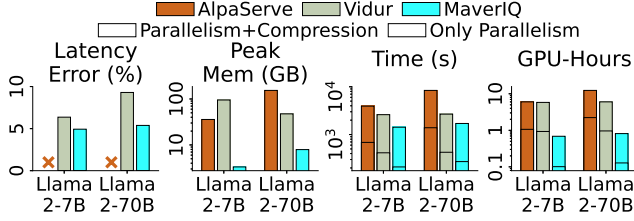


Figure 9: MaverIQ reduces profiling error and profiling cost compared to Vidur and AlphaServe. See § 8.1.

selected deployment configurations, collects latency and memory metadata, and extrapolates deployment characteristics for the full LLM, as described in Section 5.

Deployment Controller. The Deployment Controller, described in Section 6, selects deployment configurations, GPUs, and layer-to-GPU mappings. We extend TensorRT-LLM to support manual layer placement across GPUs, enabling MaverIQ to unevenly distribute layers and utilize fragmented GPU resources.

Monitoring Daemon. MaverIQ launches a lightweight monitoring daemon per worker that tracks GPU memory and load via the PyTorch CUDA API. The daemon periodically sends updates to the Controller through Linux TCP sockets [15], enabling load-aware deployment decisions.

8 Evaluation

Experimental setup. We evaluate MaverIQ using two Azure LLM inference traces [34]: a bursty *code* trace and a steady high-load *conversation* trace. We use six state-of-the-art LLMs (Llama-2-7B, 13B, 70B, GPT-J-6B, and Falcon-7B, 40B) on an 8-GPU cluster with NVIDIA RTX A6000 GPUs. We compare against three state-of-the-art baselines: (1) Accelerate [12], an LLM serving framework that automates LLM deployment by leveraging FSDP [47], (2) AlphaServe [20], an LLM serving system that automatically selects parallelism configurations to meet SLOs by leveraging brute-force profiling (we also test AlphaServe*, a variant not allowed to drop LLMs), and (3) Vidur [1] a simulation tool that predicts the most cost-efficient parallelism configuration using random forest regressors. We report both user-facing metrics (latency, cost, memory footprint, GPU-hours, SLO attainment) and provider metrics (cluster throughput and operational cost).

8.1 Profiling Accuracy and Cost

We first evaluate MaverIQ’s profiling methodology against Vidur and AlphaServe. Figure 9 shows that MaverIQ reduces latency estimation error by 1.3-1.7 $\times$ compared to Vidur. Vidur trains a regressor per LLM to predict latency, however, MaverIQ’s analytical modeling across LLMs is more accurate. At the same time, fingerprint-based profiling substantially reduces profiling overheads, decreasing profiling

memory consumption, time, and GPU-hours by 6-28 $\times$, 1.6-5 $\times$, and 7-15 $\times$, respectively, compared to Vidur, which trains regressors to predict LLM profiles, and AlphaServe, which brute-force profiles an LLM across all deployment configurations.

MaverIQ accurately estimates deployment characteristics while substantially reducing profiling memory consumption, time, and GPU-hours compared to state-of-the-art baselines.

8.2 Intent-Based Deployment

We evaluate MaverIQ’s ability to meet user intents across varying query arrival rates, traces, model sets, and accuracy requirements.

Figure 10 shows that MaverIQ consistently reduces inference latency while maintaining high cluster throughput across both Azure traces. Compared to Accelerate and AlphaServe, MaverIQ reduces latency by up to > 90% under high load and continues to outperform baselines as arrival rates vary. These gains arise from MaverIQ’s ability to tailor deployment configurations to individual LLMs and utilize load-aware placement policies that reduce GPU contention.

We further evaluate MaverIQ under strict accuracy SLOs, where deployment configurations without compression must be selected. Figure 11 shows that MaverIQ continues to outperform the baselines, reducing latency by 28-45% while increasing cluster throughput and reducing operational cost by 3.8-8.3 $\times$.

MaverIQ robustly meets user intents across varying workloads, traces, and accuracy requirements while improving cluster throughput and reducing operational cost.

8.3 Fragmentation-Aware Deployment

We evaluate MaverIQ’s fragmentation-aware deployment under increasing deployment pressure. Figure 12 shows that MaverIQ deploys substantially more LLMs than the baselines while improving cluster throughput. By leveraging quantization, adaptive parallelism, and uneven layer placement, MaverIQ efficiently utilizes fragmented GPU memory and scales across diverse deployment objectives.

9 Discussion

Extensibility to new models. MaverIQ’s fingerprint-based profiling extends naturally to model architectures with repetitive structures beyond decoder-only transformers [43]. Examples include MoE-based models such as Mixtral-8-7B [16], recurrent architectures such as RWKV [35], and multimodal models such as Flamingo [3]. For example, Mixtral-8-7B contains repeated experts within each MoE layer, suggesting that lightweight fingerprints may substantially reduce profiling overheads for MoE-based deployment as well. Extending

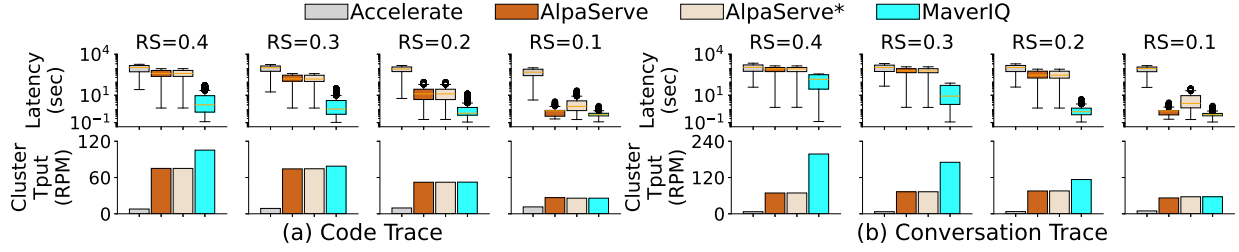


Figure 10: MaverIQ minimizes latency and improves cluster throughput across varying query arrival rates (RS) and traces. See § 8.2.

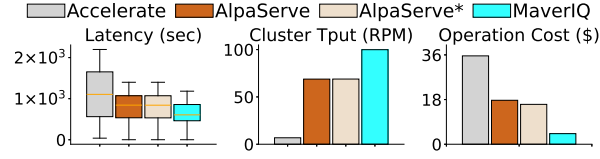


Figure 11: MaverIQ performs best under strict accuracy SLOs. See § 8.2.

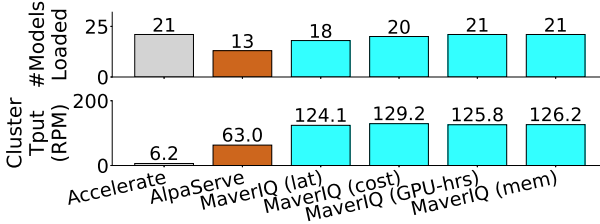


Figure 12: MaverIQ increases the number of deployable models and improves cluster throughput across user intents. See § 8.3.

the analytical formulations to such architectures remains an important direction for future work.

Extensibility to new intents. MaverIQ currently supports intents derived from latency and memory characteristics, including cost, GPU-hours, and throughput. The same profiling framework can be extended to additional deployment objectives such as energy or carbon efficiency by profiling and extrapolating additional metrics.

Extensibility to heterogeneous hardware. MaverIQ currently targets clusters with homogeneous GPU types. In homogeneous GPU clusters, MaverIQ’s profiling cost remains constant as cluster size increases because deployment characteristics can be extrapolated without additional profiling. Extending fingerprint-based profiling and deployment to heterogeneous clusters introduces additional challenges due to latency variation across devices and deployment stages. Nevertheless, the lightweight profiling methodology developed by MaverIQ provides a foundation for future AI-native infrastructure systems that dynamically adapt deployment decisions across heterogeneous resources.

10 Conclusion

We presented MaverIQ, an intent-based LLM inference serving system that combines lightweight fingerprint-based pro-

filings with fragmentation-aware deployment. MaverIQ uses fingerprints and analytical models to efficiently estimate deployment tradeoffs across large configuration spaces, while leveraging uneven layer placement to utilize fragmented GPU resources without degrading inference latency. Across diverse LLMs, traces, and deployment objectives, MaverIQ reduces profiling overheads and operational cost while effectively meeting user intents compared to state-of-the-art baselines. More broadly, MaverIQ highlights the importance of lightweight profiling and resource-aware deployment for emerging AI-native infrastructure.

References

- [1] Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A Large-Scale Simulation Framework For LLM Inference. *Proceedings of Machine Learning and Systems*, 6:351–366, May 2024.
- [2] 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, Santa Clara, CA, July 2024. USENIX Association.
- [3] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a Visual Language Model for Few-Shot Learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

- [4] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. *arXiv preprint arXiv: 2207.00032*, 2022.
- [5] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 472–487, New York, NY, USA, 2022. Association for Computing Machinery.
- [6] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.
- [7] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, 2023.
- [8] Jiaang Duan, Shiyu Qian, Dingyu Yang, Hanwen Hu, Jian Cao, and Guangtao Xue. MOPAR: A Model Partitioning Framework for Deep Learning Inference Services on Serverless Platforms. *arXiv preprint arXiv:2404.02445*, 2024.
- [9] Elias Frantar and Dan Alistarh. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.
- [10] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. In *International Conference on Learning Representations*, 2023.
- [11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [12] Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. Accelerate: Training and inference at scale made simple, efficient and adaptable. <https://github.com/huggingface/accelerate>, 2022. Accessed: 2024-08-17.
- [13] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, Hyoungho Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.
- [14] HuggingFace. Safetensors. <https://github.com/huggingface/safetensors>, 2022. Accessed: 2024-10-08.
- [15] IBM Developer. Socket programming in C, 2020. Accessed: 2025-04-13.
- [16] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, L  lio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Th  ophile Gervet, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. Mixtral of Experts. *arXiv preprint arXiv:2401.04088*, 2024.
- [17] Renren Jin, Jiangcun Du, Wuwei Huang, Wei Liu, Jian Luan, Bin Wang, and Deyi Xiong. A Comprehensive Evaluation of Quantization Strategies for Large Language Models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 12186–12215, Bangkok, Thailand, August 2024. Association for Computational Linguistics.
- [18] 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 PageAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery.
- [19] Malgorzata Lazuka, Andreea Anghel, and Thomas Parnell. LLM-Pilot: Characterize and Optimize Performance of your LLM Inference Services. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '24. IEEE Press, 2024.
- [20] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. In *17th USENIX*

Symposium on Operating Systems Design and Implementation (OSDI 23), pages 663–679, Boston, MA, July 2023. USENIX Association.

- [21] Dimitrios Liakopoulos, Prasoon Sinha, Tianrui Hu, Myungjin Lee, and Neeraja J. Yadwadkar. MaverIQ: Fingerprint-Guided Extrapolation and Fragmentation-Aware Layering for Intent-Based LLM Serving. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '25*, page 1676–1696, New York, NY, USA, 2025. Association for Computing Machinery.
- [22] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Weiming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of Machine Learning and Systems*, volume 6, pages 87–100, 2024.
- [23] Zhiqi Lin, Youshan Miao, Guodong Liu, Xiaoxiang Shi, Quanlu Zhang, Fan Yang, Saeed Maleki, Yi Zhu, Xu Cao, Cheng Li, Mao Yang, Lintao Zhang, and Lidong Zhou. SuperScaler: Supporting Flexible DNN Parallelization via a Unified Abstraction. *arXiv preprint arXiv:2301.08984*, 2023.
- [24] Weijie Liu, Zhiqian Lai, Shengwei Li, Yabo Duan, Keshi Ge, and Dongsheng Li. AutoPipe: A fast pipeline parallelism approach with balanced partitioning and micro-batch slicing. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 301–312. IEEE, 2022.
- [25] Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the Structural Pruning of Large Language Models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*, 2023.
- [26] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. SpotServe: Serving Generative Large Language Models on Preemptible Instances. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS '24*, page 1112–1127, New York, NY, USA, 2024. Association for Computing Machinery.
- [27] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21*, New York, NY, USA, 2021. Association for Computing Machinery.
- [28] NVIDIA. Optimizing NVIDIA TensorRT Conversion for Real-time Inference on Autonomous Vehicles. <https://developer.nvidia.com/blog/optimizing-nvidia-tensorrt-conversion-for-real-time-inference-on-autonomous-vehicles/>, 2020. Accessed: 2024-10-15.
- [29] NVIDIA. TensorRT-LLM. <https://github.com/NVIDIA/TensorRT-LLM>, 2023. Accessed: 2024-10-20.
- [30] NVIDIA. Amazon Accelerates Customer Satisfaction With NVIDIA Triton Inference Server and NVIDIA TensorRT. <https://resources.nvidia.com/en-us-inference-customer-story/nvidia-amazon-custom>, 2024. Accessed: 2024-10-17.
- [31] NVIDIA. American Express Prevents Fraud and Foils Cybercrime With NVIDIA AI Solutions. <https://resources.nvidia.com/en-us-inference-customer-story/american-express-prevents-fraud>, 2024. Accessed: 2024-10-17.
- [32] Hyungjun Oh, Kihong Kim, Jaemin Kim, Sungkyun Kim, Junyeol Lee, Du seong Chang, and Jiwon Seo. ExeGPT: Constraint-Aware Resource Scheduling for LLM Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS '24*, page 369–384, 2024.
- [33] OpenNMT. CTranslate2. <https://opennmt.net/CTranslate2/>, 2019. Accessed: 2024-10-06.
- [34] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132, 2024.
- [35] Bo Peng, Eric Alcaide, Quentin Gregory Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Nguyen Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan S. Wind, Stanisław Woźniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

- [36] Jeff Pool, Abhishek Sawarkar, and Jay Rodge. Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT. <https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/>, 2021. Accessed: 2024-10-20.
- [37] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Power-aware Deep Learning Model Serving with μ -Serve. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 75–93, Santa Clara, CA, July 2024. USENIX Association.
- [38] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf, 2018.
- [39] Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the Effects of Data Parallelism on Neural Network Training. *Journal of Machine Learning Research*, 20:1–49, 2019.
- [40] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053*, 2020.
- [41] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. *arXiv preprint arXiv:2408.00741*, 2024.
- [42] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A Simple and Effective Pruning Approach for Large Language Models. In *The Twelfth International Conference on Learning Representations*, 2024.
- [43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [44] Boxiang Wang, Qifan Xu, Zhengda Bian, and Yang You. Tesseract: Parallelize the Tensor Parallelism Efficiently. In *Proceedings of the 51st International Conference on Parallel Processing, ICPP '22*. ACM, August 2022.
- [45] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.
- [46] Jeffrey Yu, Kartik Prabhu, Yonatan Uрман, Robert M. Radway, Eric Han, and Priyanka Raina. 8-bit Transformer Inference and Fine-tuning for Edge Accelerators. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS '24*, page 5–21, New York, NY, USA, 2024. Association for Computing Machinery.
- [47] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *arXiv preprint arXiv:2304.11277*, 2023.
- [48] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578, Carlsbad, CA, July 2022. USENIX Association.