CRAFT: Fine-Grained Cost-Aware Expert Replication For Efficient Mixture-of-Experts Serving

Adrian Zhao [*1,2] **Zhenkun Cai** [1] **Zhenyu Song** [1] **Lingfan Yu** [1] **Haozheng Fan** [1] **Jun Wu** [1] **Yida Wang** [1] **Nandita Vijaykumar** [1,2]

Abstract

Mixture-of-Experts (MoE) has recently emerged as the mainstream architecture for efficiently scaling large language models while maintaining near-constant computational cost. Expert parallelism distributes parameters by partitioning experts across devices, but this introduces token-level load imbalance during inference. Expert replication is a widely adopted load-balancing technique in serving frameworks that alleviates load imbalance in large-scale deployments by replicating experts with high loads. In this work, we demonstrate that existing replication schemes often *over-replicate*, with many replicas providing marginal improvement. Replicas consume substantial GPU memory, which may lead to resource contention and throughput degradation. We present CRAFT, an efficient expert replication framework that maximizes load balance under a given memory budget by performing fine-grained, per-layer replication based on the estimated replication benefit. CRAFT can be seamlessly integrated into existing serving frameworks without any additional training or model changes. Our evaluation shows that CRAFT increases end-to-end serving throughput by $1.14\times$ on average (up to $1.2\times$) over existing replication techniques in large-scale deployments with models ranging from hundreds of billions to a trillion parameters.

1 Introduction

Mixture-of-Experts (MoE) alleviates the steep compute and memory scaling of dense transformers by replacing the dense feed-forward blocks with a router and a pool of experts (Kaplan et al., 2020; Hoffmann et al., 2022; Shazeer et al., 2017). By activating only a subset of experts per token, MoE models decouple total parameters from per-token FLOPs and achieve near-constant inference cost as capacity scales (Du et al., 2022). It has become the foundation for recent large models across various domains, such as LLMs (Guo et al., 2025; Yang et al., 2025; Meta AI, 2025; Kimi Team et al., 2025), image and video diffusion models (Wan et al., 2025; Fang et al., 2024), and vision transformers (Chen et al., 2023), delivering competitive performance at substantially lower compute than dense counterparts. However, deploying MoE models incurs substantial operational costs, primarily due to the massive GPU memory footprint required to host all expert parameters (Rajbhandari et al., 2022; Fedus et al., 2022). Therefore, system-level optimizations that improve inference throughput yield significant cost-performance benefits by maximizing resource utilization and reducing GPU memory pressure (Eassa et al., 2024; Srivastava, 2025; Li et al., 2025).

To effectively distribute large expert parameters across devices, Expert Parallelism (EP) partitions experts across devices, with each device holding weights of a subset of experts. EP uses two all-to-all collectives to dispatch tokens to the devices that host the assigned experts and then recombines them (Fedus et al., 2022; Lepikhin et al., 2020). Despite its promising scalability, EP introduces a new system bottleneck during deployment — *expert load imbalance*, where *expert load* is the number of tokens routed to an expert. The MoE router routes tokens to experts based on input distributions and often sends a disproportionate number of tokens to a few *hot* experts while other experts remain *cold* (Huang et al., 2024; Jin et al., 2025; Du et al., 2024). Thus, the cumulative token loads across GPUs become imbalanced, leading to: (i) idleness on GPUs with low load and (ii) network congestion during dispatch and combine due to imbalanced traffic (Li et al., 2023; Liu et al., 2025; He et al., 2022; Luo et al., 2025). This degrades communication and computation efficiency during MoE inference, necessitating load-balancing strategies for EP.

Expert placement and *expert replication* are two load-balancing techniques applicable to any MoE model and are widely adopted in modern LLM serving frameworks (Zheng et al., 2024; Kwon et al., 2023; NVIDIA Corporation, 2023; Rajbhandari et al., 2022). Expert placement leverages expert load distribution to assign experts to de-

[1]Amazon, Santa Clara, California, USA [2]Department of Computer Science, University of Toronto, Toronto, Ontario, Canada. Correspondence to: Adrian Zhao <adrianz@cs.toronto.edu>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

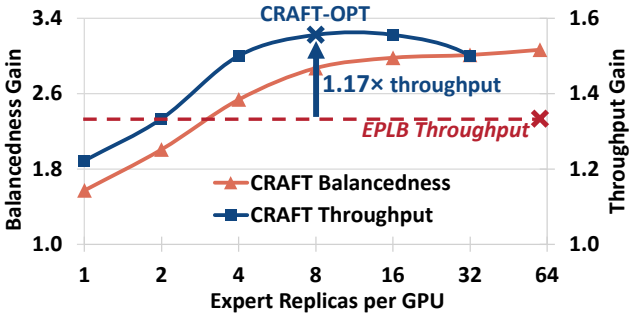


Figure 1. Balancedness gain (orange) and throughput gain (blue) of CRAFT on Kimi-K2-1000B deployed on 64 GPUs, normalized to the baseline with only expert placement (refer to `KE8` configuration in Section 3.1). EPLB is configured with 60 replicas per GPU on Kimi-K2 with 60 MoE layers (at minimum one replica per layer per GPU). The X-axis represents the number of replicas on a base-2 logarithmic scale, where more replicas imply higher memory consumption (to the right).

vices such that the expected tokens per device and traffic patterns are balanced (Yao et al., 2024; Go & Mahajan, 2025; Li et al., 2023). However, expert placement fails in layers where a few experts dominate most of the token load, making a balanced packing impossible. Expert replication addresses this by replicating hot experts (Liu et al., 2024). This mitigates extreme load imbalance by distributing token loads from a single hot expert to multiple devices. We refer to the number of additional experts allocated during replication as *replicas*.

Although expert replication is an effective load-balancing strategy that mitigates extreme expert load imbalance, it requires substantial GPU memory to store the replica weights. The most widely adopted replication technique EPLB (Liu et al., 2024) allocates one replica per layer per GPU, incurring significant GPU memory overhead on large MoE models with many MoE layers. Large-scale LLM deployments are already memory-intensive due to high parameter counts and the large KV cache necessary to efficiently process long-context requests (Kwon et al., 2023; Dao, 2023). Therefore, the memory overhead limits the practicality of replication for large-scale MoE deployments. Expert placement does not incur additional memory overhead, but it cannot handle extreme load skew during inference. Thus, efficiently achieving balanced expert load in large-scale deployment remains an important unsolved problem for large MoE models.

In this work, we perform a detailed characterization of the benefits and overheads of replication as a means to achieve load balance. We make two key observations: (1) As the number of replicas increases, balancedness gains diminish rapidly. (2) MoE layers benefit differently from replication and can be roughly categorized into two types: layers with few experts dominating token load (high-skew) benefit substantially from replication, while layers with balanced token load (low-skew) benefit little. Leveraging these observations, we introduce CRAFT, an end-to-end cost-aware replica allocation framework for efficient expert replication. The key idea of CRAFT is to selectively allocate replicas at fine, per-layer granularity to layers that benefit more from replication. We develop a cost model that estimates the per-layer balancedness gain with replication (i.e., *replication benefit*). Based on this cost model, CRAFT computes a per-layer replication plan with optimal balancedness by assigning more replicas to layers with higher estimated gains.

Figure 1 demonstrates the effectiveness of CRAFT: (1) CRAFT allocates replicas for each layer based on estimated performance benefits. This enables replication under flexible memory budgets that are much lower than EPLB (depicted by the blue curve), reducing memory usage while preserving balancedness. (2) CRAFT with the optimal number of replicas (CRAFT-OPT) achieves $1.17\times$ higher throughput compared to EPLB. The number of replicas can either be manually set by the user to meet specific memory constraints, or CRAFT can automatically set a value that optimizes replication efficiency. We implement CRAFT in three steps: First, we analyze expert load distributions and estimate per-layer benefits across different replica counts. Then, we use dynamic programming to select the optimal number of replicas per layer under the memory budget. Finally, we reserve memory and assign experts evenly across devices to minimize capacity imbalance, producing a complete replication plan for deployment.

Our contributions are summarized as follows:

- To our knowledge, this is the first detailed characterization of how throughput and load balancedness scale with replication. We demonstrate that replication quickly yields diminishing returns due to the trade-off between memory overhead and balanced load, and identify layer-specific behavior as critical to achieving efficiency with replication.

- We introduce CRAFT, an end-to-end expert replication framework designed for efficient expert replication in large-scale MoE deployments. CRAFT integrates directly into existing LLM serving frameworks (Liu et al., 2024; NVIDIA Corporation, 2023; Zheng et al., 2024; Kwon et al., 2023), offering significant throughput gains and reduced operational costs (Nguyen & Perez, 2025). CRAFT applies to any MoE architecture and requires no additional training or model changes.

- We evaluate CRAFT on a state-of-the-art LLM serving framework SGLang (Zheng et al., 2024) for a wide range of datasets and MoE models. We demonstrate

that CRAFT achieves consistent throughput gains, outperforming EPLB by $1.14\times$ on average (up to $1.2\times$) on an 8-node cluster.

2 BACKGROUND

2.1 Mixture of Experts

Mixture-of-Experts (MoE) replaces a dense feed-forward block with a fixed number of *experts* and a learned *router* that assigns each input token to a sparse subset of experts. Each *expert* is an independently parameterized feed-forward block consisting of an up-projection linear layer, a non-linear activation, and a down-projection linear layer. The *router* typically implements a weighted top-K routing mechanism with a lightweight gating network (Shazeer et al., 2017). For each token, the gating network computes a score for each expert and selects the experts with the top-K scores for activation. In practice, K is often set to 8 in large-scale MoE models (Yang et al., 2025; Guo et al., 2025; Kimi Team et al., 2025). After passing through the feed-forward blocks of the activated experts, their outputs are gathered and aggregated into a weighted sum. Because each token only activates a few experts, MoE enables the scaling of model parameters by adding more experts without substantially increasing computation costs (Du et al., 2022).

2.2 Expert Load Imbalance

Unlike traditional dense models, where the compute load across devices is uniform, MoE with EP introduces device-level token load variance due to its dynamic routing. During MoE inference, the router activates experts based on the input tokens (Shazeer et al., 2017). Real-world natural language tokens follow a Zipfian distribution, in which a few tokens are extremely frequent while most others are rare (Roller et al., 2021; Piantadosi, 2014). This leads to expert load imbalance, producing high-load *hot* experts and low-load *cold* experts at each MoE layer (He et al., 2022; Zhou et al., 2022; Huang et al., 2024; Nie et al., 2023).

When deploying MoE models with EP, the expert load imbalance becomes device-level load imbalance: GPUs that host hot experts become performance bottlenecks, causing stalls on GPUs with cold experts and network contention during the all-to-all. Figure 2 illustrates an example of the expert load distribution of an MoE layer with 8 experts. The baseline placement strategy simply assigns experts to devices based on their ID. This leads to a significant load skew, where GPU 1 carries a much higher load than other GPUs, resulting in suboptimal performance.

2.3 Inference Load-Balancing Techniques

Despite the dynamic nature of MoE routing, the expert load at each layer typically follows a distinct distribution.

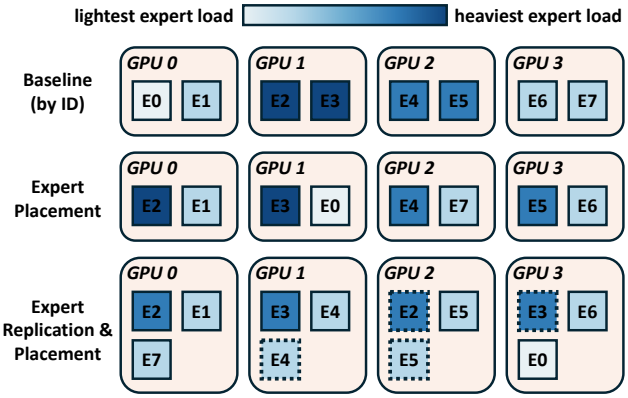
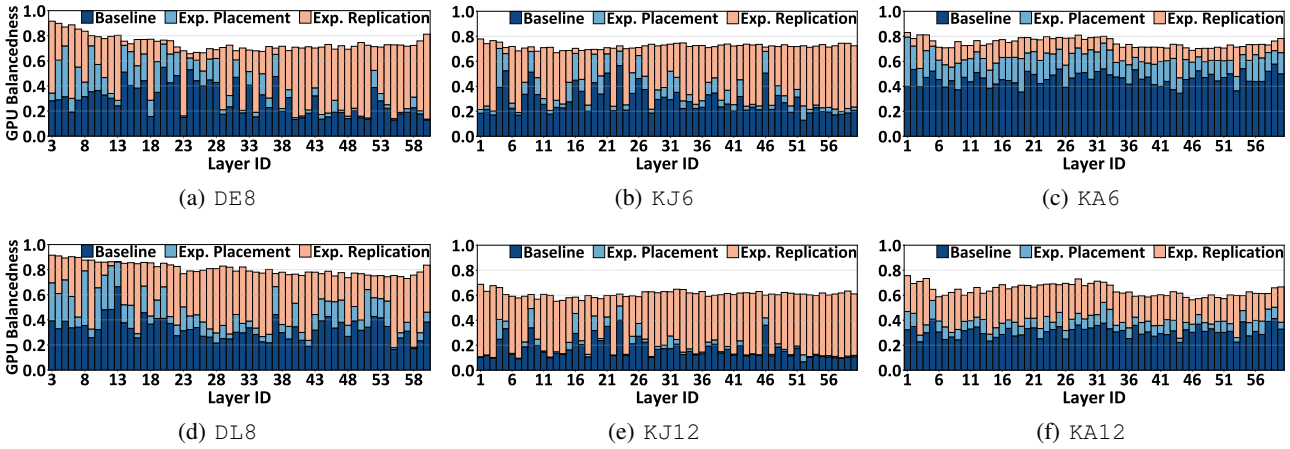


Figure 2. Expert load distribution under different EP optimizations on an MoE layer with 8 experts. Color density represents the number of tokens assigned to the expert (expert load) during inference, and dotted experts represent replicas allocated during replication.

Existing load-balancing techniques profile expert load distributions by recording token loads across three dimensions — batch, layer, and experts. Analysis of the load distribution identifies hot and cold experts in each layer, providing opportunities for load-balancing optimizations.

Expert placement is a widely adopted EP load-balancing technique that alleviates load imbalance by co-locating hot experts with cold experts, thereby minimizing load variance across devices (Liu et al., 2024; Go & Mahajan, 2025). Figure 2 demonstrates the effectiveness of expert placement, as the GPU load is more balanced compared to the baseline placement. However, expert placement falls short when the load distribution is significantly skewed, in which a few experts dominate the token load. For example, in Figure 2, if experts 2 and 3 constitute the majority of the total token load, the GPU load post-placement would remain imbalanced.

Expert replication further improves load balance by replicating hot experts and spreading their load across replicas. Expert Parallelism Load Balancer (EPLB) (Liu et al., 2024) is a widely adopted load balancer in modern LLM serving frameworks (Zheng et al., 2024; Kwon et al., 2023; NVIDIA Corporation, 2023; Rajbhandari et al., 2022) that supports both expert placement and expert replication. It employs *uniform replication*, allocating the same number of replicas per layer to each GPU. The minimum number of replicas per layer with *uniform replication* is equal to the number of GPUs (1 extra replica per GPU). In Figure 2, four replicas are created for experts 2, 3, 4, and 5, and each GPU reserves memory for one additional expert slot to host these replicas. Then, expert placement maps the experts (including replicas) across GPUs. As shown in the figure, expert replication can achieve near-perfect GPU load balance.



(a) DE8 (b) KJ6 (c) KA6

(d) DL8 (e) KJ12 (f) KA12

Figure 3. Breakdown of the final post-replication balancedness by each technique's contribution across various configurations. The total height of a bar represents the GPU balancedness in an MoE layer with expert replication. A technique constituting a higher portion of the bar implies higher contribution to the overall balancedness. Layers excluded from the figures are non-MoE dense layers.

3 MOTIVATION

Although expert replication with placement mitigates most expert-level load imbalance, it introduces an important trade-off between GPU memory usage and load balance. In this section, we present a detailed analysis of the effectiveness of expert replication.

3.1 Experimental Setup

System. Our experiments are conducted on a cluster of AWS EC2 p4de.24xlarge instances, each equipped with 8 NVIDIA A100 GPUs (80 GB each) interconnected via NVLink. Inter-node communication uses the Elastic Fabric Adapter (EFA) (AWS, 2025) with direct peer-to-peer (P2P) GPU communication support. All evaluations use CUDA 12.8 and NCCL 2.26.2 (NVIDIA, 2025).

Metrics. *Expert load* is measured as the total number of tokens dispatched to an expert after routing. We evaluate load balance using *balancedness*, computed as the average load divided by the max load (higher means more balanced).

Models and Workloads. We use two large-scale MoE models in bfloat16 format: `D` — DeepSeek-R1-671B with 58 MoE layers (layers 3 - 60) and 256 experts (Guo et al., 2025), and `K` — Kimi-K2-1000B with 60 MoE layers (layers 1 - 60) and 384 experts (Kimi Team et al., 2025). Both models employ top-8 expert routing. For evaluation, we use diverse long-sequence workloads from large-scale datasets ranging from hundreds of millions to trillions of tokens. The workloads consist of diverse text spanning a broad range of domains, including law, science, entertainment, literature, news articles and technical documents in various formats such as PDFs, passages, and research papers: (1) two splits from the FinePDFs dataset (Kydlíček et al., 2025): `E` — deu_Latn split with German PDF files, and `J` — jpn_Jpan split with Japanese and Chinese PDF files, (2) `L` — Lambada dataset (Paperno et al., 2016) with long narrative passages, (3) `A` — RedPajama-Data-1T dataset (Weber et al., 2024) arXiv split with academic research papers. All inputs are chunked to 4096 tokens with a fixed-size output length of 256 tokens. For each workload, we randomly sample a pool of sequences and run 3000 inference batches to collect the expert load distribution (Section 2.3).

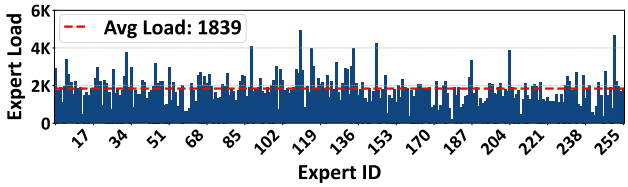
Setup Configuration Syntax. We use a two-letter combination to represent different model-workload setups, followed by a number indicating the cluster size. For example, `DE8` represents the DeepSeek-R1 model and the FinePDF deu_Latn workload on a cluster of 8 nodes.

3.2 Replication Effectiveness on Load-Balancing

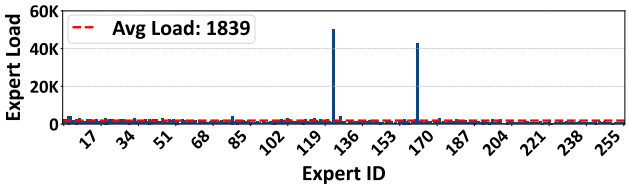
We analyze the effectiveness of EPLB placement and replication by measuring the achieved GPU load balancedness for each layer. Figure 3 illustrates the breakdown of the balancedness gains when applying expert replication. We make two observations.

Observation 1: The effectiveness of replication varies across layers. Although the effectiveness of replication matches or exceeds expert placement in many layers, some layers benefit little. To characterize this variation, we examine two representative layers in Figure 3(a) — layer 20 (least effective) and layer 51 (most effective). Figure 4 illustrates their average expert load distributions. Layer 20 exhibits a balanced, dispersed distribution. This explains its high baseline balancedness with placement, as the peak-to-mean load ratio is only approximately $2.5\times$, allowing for effective hot–cold expert co-location. In contrast, layer 51 exhibits strong skew: the hottest expert receives $> 27\times$ the

average load, and the top two experts account for $\approx 20\%$ of the total load. Placement alone cannot achieve load balance because GPUs hosting these hot experts are inevitably overloaded. Replication mitigates this skew by distributing hot-expert traffic across replicas, substantially reducing the peak load and improving balancedness. We can thus roughly categorize all layers into these two types: high-skew layers (e.g., layer 51) — replication-effective layers, where the maximum expert load significantly exceeds the average expert load ($> 10\times$); and low-skew layers (e.g., layer 20) — replication-ineffective layers, where the maximum load is similar to the average load.



(a) Layer 20 - low-skew layer (expert load range 0 - 6K)



(b) Layer 51 - high-skew layer (expert load range 0 - 60K)

Figure 4. Average expert load distribution of `DE8`. Total load is identical on both layers; the red line marks the average load.

Observation 2: As cluster size increases, overall load balancedness decreases and replication becomes more effective. In Figures 3(b), 3(c), 3(e), and 3(f), increasing cluster size from 6 to 12 nodes decreases baseline balancedness and placement effectiveness across layers. This is because increasing the number of GPUs reduces the number of experts per GPU. This reduces opportunities for hot-cold expert colocation and weakens the averaging effect on each device, making the system more susceptible to skewed load distributions. Replication remains effective because the overall load distribution remains unchanged, and replicating hot experts still reduces peak load and improves balancedness. Moreover, Figures 3(c), 3(f) show that setup `KA` is relatively balanced with strong baseline balancedness and placement effectiveness at 6 nodes. However, as cluster size increases to 12, baseline balancedness substantially degrades, where replication becomes more effective than placement. This underscores the importance of expert replication at scale.

3.3 Replication Scaling

To mitigate performance degradation from replication memory overhead and identify the optimal balancedness-memory trade-off, we analyze how balancedness scales with replica count at fine-grained per-layer granularity. We study both aggregated balancedness across layers and per-layer balancedness scaling, and we make two observations.

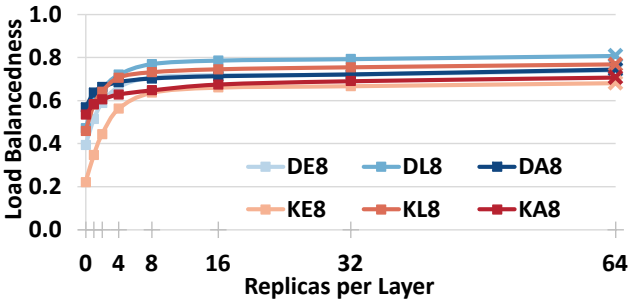


Figure 5. Load balancedness under varying per-layer replica counts across configurations. Balancedness is aggregated across all MoE layers. Each layer is allocated the same number of replicas; zero denotes the placement-only baseline. $\times$ indicates the minimum uniform replication (one replica per layer per GPU, Section 2.3).

Observation 3: Load balancedness scales sublinearly with the number of replicas. Figure 5 depicts the aggregated GPU balancedness. Across configurations, initial replication reduces peak expert load and improves load balance. After replicating hot experts sufficiently, placement alone balances the remaining load, and additional replicas yield little or no benefit. Doubling replicas beyond 16 offers negligible balancedness gains but doubles memory use. Existing *uniform replication* limits the minimum number of replicas (Section 2.3), where it allocates at minimum 63 additional replicas per layer in a 64-GPU system (one GPU holds the original expert copy). This indicates significant memory inefficiency, where 63 replicas are excessive as most replicas do not contribute to load balance.

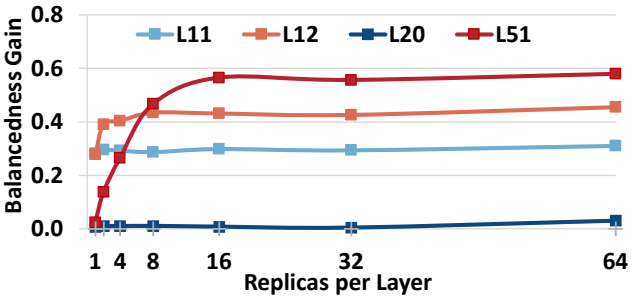


Figure 6. Per-layer load balancedness gain in 4 sample layers under varying replica counts with configuration `DE8`.

Observation 4: The effective replica count varies across layers. Figure 6 depicts balancedness gains at different replica counts for four layers: aforementioned balanced layer 20 and skewed layer 51 (Figure 4), and two layers 11 and 12 with moderate skewness. We observe that (1) the replica count at which gains plateau varies by layer, and (2)

benefits diminish beyond 16 replicas across layers, confirming Observation 3. Layer 20 gains little from replication, consistent with its already-balanced distribution (Observation 1). For layers 11, 12, and 51, the replica counts at which balancedness gains diminish are 1, 8, and 16, respectively. Hence, replication should be allocated at fine, per-layer granularity to match layer-specific needs; a fixed replica count over-allocates balanced layers and under-allocates skewed layers, leading to wasted memory.

4 DESIGN

We introduce CRAFT, an end-to-end replica allocation framework that enables replication under low memory budgets while preserving near-optimal load balance. We define the balancedness gain from replication as *replication benefit*, and the number of experts stored on each GPU as *expert capacity*. The core idea of CRAFT is to independently allocate replicas at the granularity of a layer, in proportion to the layer-specific replication benefit. This benefit-driven policy maximizes per-replica effectiveness and sustains high balancedness gains.

4.1 Design Challenges of CRAFT

Implementing per-layer fine-grained replica allocation requires addressing the following challenges:

Challenge 1: How do we determine the optimal number of replicas to allocate to each layer to maximize the overall balancedness gain? As discussed in Section 3.3, balancedness scales differently with replica count across layers. Selecting each layer's locally optimal replica count is insufficient, because layers benefit from replication differently, and the total number of replicas is constrained by the replication memory budget.

Challenge 2: With varying replica counts per layer, how do we avoid memory imbalance when assigning replicas to devices? The uniform replication baseline allocates one replica per layer per device, so assignment is simple and memory usage is trivially balanced across devices. Under benefit-driven replication, layers have fewer replicas than devices, which challenges replica-to-device assignment for two reasons: (1) The expert capacity across GPUs for each layer may differ. This complicates expert placement within each layer and introduces additional imbalance, because each GPU may hold a different number of experts. (2) Replica memory allocation must remain consistent across devices to maintain a uniform KV cache size. Inconsistent KV cache sizes lead to memory fragmentation, where the maximum concurrency is determined by the device with the lowest KV cache size within a data-parallel (DP) rank.

4.2 CRAFT Workflow

Figure 7 illustrates the high-level CRAFT workflow on an example with four MoE layers ($L = 4$) and four GPUs ($D = 4$). The replication factor R denotes the number of replicas per GPU, quantifying the replication memory usage. The total number of replicas r is calculated as $r = R \times D$. CRAFT allocates replicas in three steps. First, we collect the balancedness scaling and estimate per-layer replication benefit by analyzing the expert load distribution ❶. Second, the value of R is determined either manually by the user or automatically according to the balancedness scaling that optimizes for high replication efficiency ❷. The example in Figure 7 uses $R = 2$ ($r = 8$). Lastly, the replica allocation algorithm computes an allocation plan of r replicas that maximizes balancedness ❸. As replica counts vary across layers, replica-to-device mapping becomes challenging (Challenge 2). CRAFT employs an interleaved expert assignment to minimize the intra-layer expert capacity imbalance across devices ❹. Then, a capacity-aware greedy placement algorithm assigns experts to devices ❺. Compared to uniform replication, which allocates $r = L \times D = 16$ replicas uniformly across layers and devices, CRAFT achieves load balance with half the memory budget. We present the design details of CRAFT in the following sections.

4.2.1 Benefit-Driven Replica Allocation

In this section, we formulate an optimization problem for benefit-driven replica allocation to address Challenge 1 (Section 4.1). The goal is to maximize the balancedness gain (i.e., replication benefit) under the r-replica budget. The allocation has three steps:

Step 1: Estimate per-layer replication benefit ❶. For each layer, we evaluate the replication benefit of $K = \log_2 D + 1$ different replica counts chosen from a base-2 geometric progression over $[1, D]$. We replay each inference batch from the expert load distribution and measure the balancedness gain at each chosen replica count (details in Appendix A.1). This replay-based analysis yields an $L \times K$ balancedness gain matrix T that quantifies per-layer replication benefits across replica counts.

Step 2: Determine replication factor R ❷. CRAFT allows the user to manually set R that meets specific memory constraints. Alternatively, CRAFT supports the automatic selection of an R that optimizes for replication memory efficiency by analyzing the balancedness gains obtained in Step 1, as illustrated in Figure 5. Specifically, CRAFT selects the R that yields the highest per-replica balancedness gain, which avoids allocating replicas with diminishing benefits (Section 3.3). We demonstrate the effectiveness of this method by showing a strong correlation between diminished balancedness gains and end-to-end throughput degradation in Section 5.3.

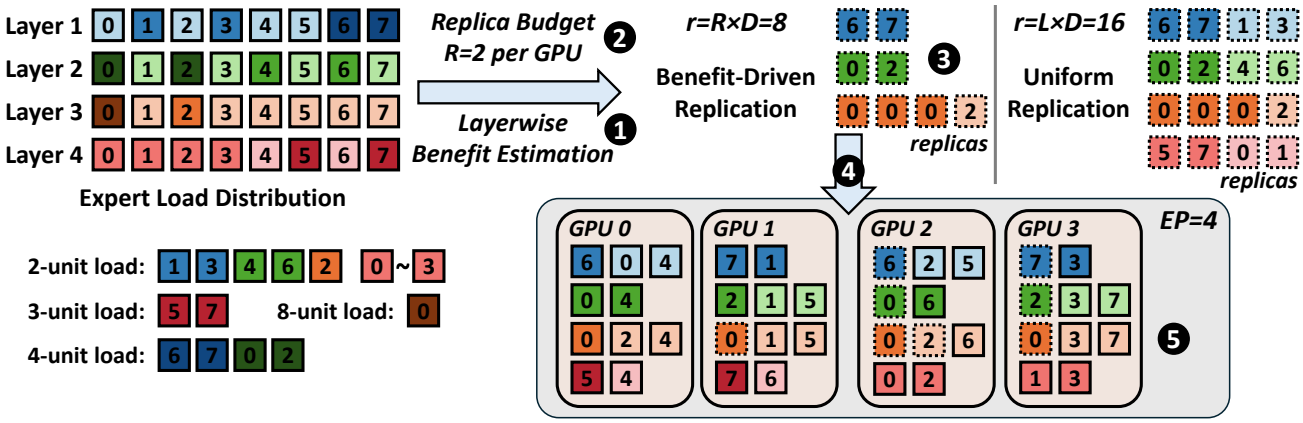


Figure 7. CRAFT workflow. We assume a system with 4 devices, 4 layers, and 8 experts per layer ($L = D = 4$). Each layer has total load of 16 units. Darker color represents heavier load, and experts without load indication have 1 unit load. Dotted boxes present replicas. CRAFT achieves perfect load balance while using only half the number of replicas compared to uniform replication (top-right).

Step 3: Compute an r-replica allocation plan with maximum benefit ❸. We transform the allocation problem into a Multiple-Choice Knapsack Problem (MCKP) — selecting one replica count per layer under the r-replica constraint that maximizes total replication benefit in T. Although MCKP is NP-hard, the parameters D, K, and L remain small even at large-scale deployment, enabling an efficient dynamic programming solution in pseudo-polynomial time (Martello & Toth, 1990). We provide a detailed implementation in Appendix A.2.

Figure 7 shows an example optimal replication plan from benefit-driven replica allocation. Layer 3 is the most skewed and receives 4 replicas (highest benefit). Layers 1 and 2 are moderately skewed and receive 2 replicas each (moderate benefit). Layer 4 is already balanced under expert placement and receives no replicas (zero benefit). This allocation achieves optimal balancedness using far fewer replicas than uniform replication.

4.2.2 *Capacity-Aware Expert Assignment and Placement*

CRAFT addresses Challenge 2 with capacity-aware expert assignment, which uses a greedy algorithm that iteratively assigns experts for each layer under two objectives:

Primary Objective: Each additional expert (replica) is always assigned to one of the GPUs with the fewest experts. Since r is a multiple of D ($r = R \times D$, R is the number of replicas per GPU, and D is the number of GPUs), replication memory usage remains consistent across devices. This constraint does not cause excessive replication, because D and L are of the same order in practice; the minimum setting $R = 1$, $r = D$ incurs marginal memory overhead under benefit-driven replication (Section 4.2.1). This constraint also bounds per-layer, device-level capacity differences to at most one, reducing expert capacity skew.

Secondary Objective: Within each layer, additional experts are assigned in an interleaved manner across devices and nodes. While satisfying the primary objective, many GPUs may tie for the fewest experts. As a tie-breaking measure, we evenly distribute experts among nodes. For example, in Figure 7, the two additional experts in layer 1 are assigned to GPU 0 (node 0) and GPU 2 (node 1); those in layer 2 are assigned to GPU 1 (node 0) and GPU 3 (node 1). This interleaved assignment balances per-layer node-level expert capacity and further improves overall load balance.

We provide the detailed implementation of the greedy assignment algorithm in Appendix A.3. With these two objectives, capacity-aware expert assignment yields an optimized GPU memory allocation plan that minimizes expert capacity imbalance ❹. We then apply a greedy placement algorithm that iteratively assigns the most-loaded expert to the least-loaded device, following the standard strategy (Liu et al., 2024) while respecting the varying expert capacity across devices ❺. After placement, CRAFT produces a complete end-to-end replication plan.

4.3 Adapting to Shifting Workload Distributions

Changes in the workload distribution during runtime may necessitate online adaptation in expert replication plans. Thus, mainstream LLM serving frameworks (Zheng et al., 2024; Kwon et al., 2023; NVIDIA Corporation, 2023) and large-scale MoE deployment practices (Alvarez et al., 2025; vLLM Team, 2025; The SGLang Team, 2025; Moreh, Inc., 2025) adopt *periodic expert rebalancing*, initially proposed by EPLB, to adapt to shifting workloads. Current online periodic rebalancing mechanisms comprise three steps: (1) the framework samples and records the expert load distribution post-routing; (2) at each rebalancing window (configurable, with a 10-minute default in EPLB), the framework triggers

rebalancing and asynchronously sends the recorded distribution to EPLB to compute a new expert-device assignment map; (3) the framework relocates expert weights across devices based on the generated assignment. These steps can be performed asynchronously by pinning expert weights in CPU memory and streaming the weight updates to GPU across multiple iterations, overlapped with ongoing inference batches. Thus, such weight updates can be performed efficiently with negligible impact on user experience and service-level objectives (NVIDIA TensorRT LLM Team, 2025a;b; The SGLang Team, 2025).

CRAFT can serve as a drop-in replacement for EPLB in existing serving frameworks that support online periodic expert rebalancing, substituting EPLB in step 2. When workload shifts occur, CRAFT can adopt periodic rebalancing to improve load balancedness over time. In Section 5.6, we analyze various workload shift and load distribution patterns, identifying scenarios where rebalancing is beneficial and where it is unnecessary.

5 EVALUATION

In this section, we address the following questions:

Question ❶: How does the balancedness-memory tradeoff from replication affect end-to-end performance?

Question ❷: What end-to-end performance improvement does CRAFT deliver compared to existing approaches?

Question ❸: How does inference performance scale with the cluster size under CRAFT?

Question ❹: Does CRAFT introduce any additional overhead in the serving framework?

5.1 Experimental Baseline

System, Models and Workloads. Refer to Section 3.1.

Baseline. We use SGLang v0.4.8 (Zheng et al., 2024) with Expert Parallelism Load Balancer (EPLB) (Liu et al., 2024) as the baseline. We adopt the standard large-scale deployment parallelism configuration: data parallelism (DP), head-parallel tensor-parallel (TP) attention, and EP. We implement dynamic all-to-all EP dispatch and combine with PyTorch and NCCL P2P collectives with EFA support. We enable mixed chunked prefill with a maximum prefill chunk size of 4096. The batch sizes are dynamic depending on request arrivals and KV cache availability at runtime. EPLB is the most widely adopted EP load-balancing algorithm supporting expert placement and replication in mainstream LLM serving frameworks (Zheng et al., 2024; Kwon et al., 2023; Rajbhandari et al., 2022; NVIDIA Corporation, 2023). We implement and integrate CRAFT into SGLang as a direct replacement for EPLB and evaluate its performance. Profiling sequences used to obtain the expert load distributions for EPLB and CRAFT are randomly sampled from the diverse datasets (Section 3.1). The samples used for profiling are excluded from the evaluation inputs. We evaluate three setups — `BASE`: EPLB placement with no replication, `EPLB`: minimum EPLB replication with L replicas per GPU (L is the number of MoE layers), and `CRA`: our approach, suffixed with the replication factor R. In our deployment setup, $R = 8$ is generally the best-performing setting selected by CRAFT across configurations (Section 4.2.1), and we use it for the subsequent end-to-end evaluations. We include an evaluation of the effect of R in Appendix B.

Metrics. We report *goodput* as the primary metric in our experiments, defined as the maximum sustained throughput the system delivers before requests incur prolonged queuing delays. For prefill and decode performance, we report *time-to-first-token* (TTFT) and *inter-token-latency* (ITL).

5.2 End-to-end Performance

We address Question ❷ by measuring end-to-end serving throughput across configurations. Figure 8 plots request rate (throughput) versus time-to-first-token (TTFT) latency for each configuration. The request arrival times are modeled as a Poisson process. The knee point marks saturation, where further increasing request rates overloads the system and TTFT becomes dominated by queuing time rather than prefill time. This knee point defines the system goodput. A higher goodput directly reduces operational costs and is crucial in real deployments, as the number of GPUs required to meet a target peak request rate under a fixed latency budget scales inversely with the per-instance goodput, dominating the total cost of ownership (Nguyen & Perez, 2025). We make three observations:

CRAFT consistently achieves higher goodput than EPLB. Figure 8 demonstrates that compared to `EPLB`, `CRA8` achieves on average $1.15\times$ higher goodput (up to $1.2\times$) on DeepSeek-R1 and $1.12\times$ (up to $1.17\times$) on Kimi-K2 in an 8-node cluster (setups suffixed with 8). This is because `CRA8` allocates $7.25\times$ and $7.5\times$ fewer replicas than `EPLB` while achieving most of the balancedness gains on DeepSeek-R1 and Kimi-K2, respectively. The saved memory enables a larger KV cache and larger sequence batches without significant slowdown from load imbalance.

CRAFT improves TTFT. Expert load imbalance is more significant during prefill due to high token counts from long-sequence requests (Section 3.1), which leads to long TTFTs (Ma et al., 2025; Zhu et al., 2025). At a fixed request rate below saturation, both `EPLB` and `CRA8` achieve lower TTFT than `BASE`. Figure 8 illustrates that `CRA8` reduces TTFT by 29% on average (up to 58%) compared to `BASE`, closely matching `EPLB` (30% on average, up to 59%). This demonstrates that CRAFT's benefit-driven replication reduces load

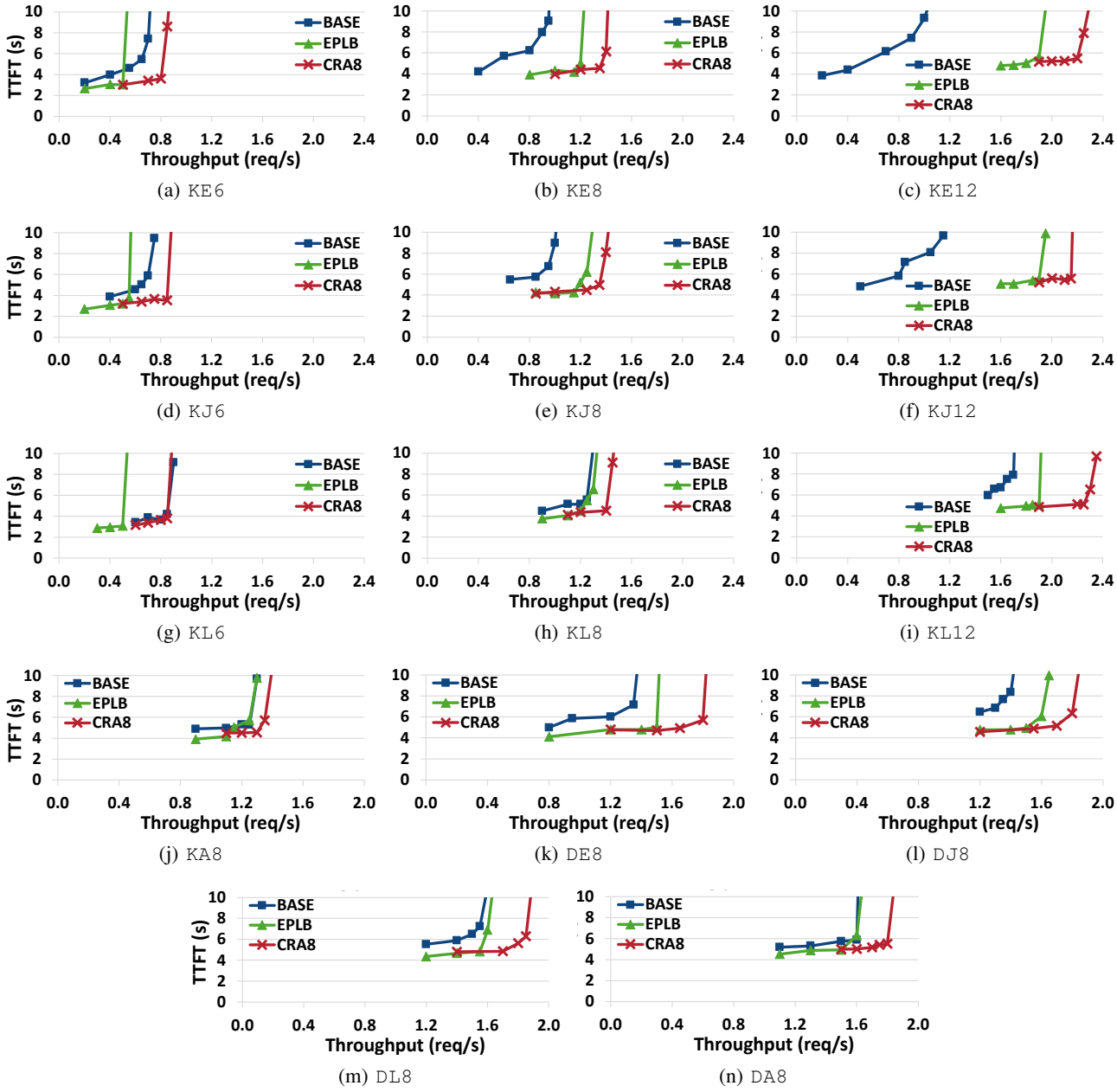


Figure 8. End-to-end throughput to TTFT curves across different system setups for BASE, EPLB and CRA8.

imbalance and improves computational efficiency to a similar extent as EPLB during prefill.

CRAFT is robust across datasets with varying load imbalance. We analyze dataset load distribution skewness as in Figure 3. We observe that datasets E and J are highly skewed (low BASE balancedness), whereas datasets L and A are less skewed (high BASE balancedness). Compared to BASE, CRA8 improves goodput by 1.42× on highly skewed datasets and by 1.14× on less skewed datasets on average, demonstrating broad applicability. In comparison, EPLB yields a 1.24× goodput gain on highly skewed datasets but only 1.02× on less skewed datasets on average. This is because less skewed datasets benefit less from replication, where excessive replication from EPLB yields little benefit.

5.3 CRAFT Balancedness-Memory Trade-Off

Expert replication mitigates load imbalance but consumes additional GPU memory. In this section, we address Question ❶ by empirically quantifying this trade-off in CRAFT and its impact on performance. Figure 9 depicts balanced-

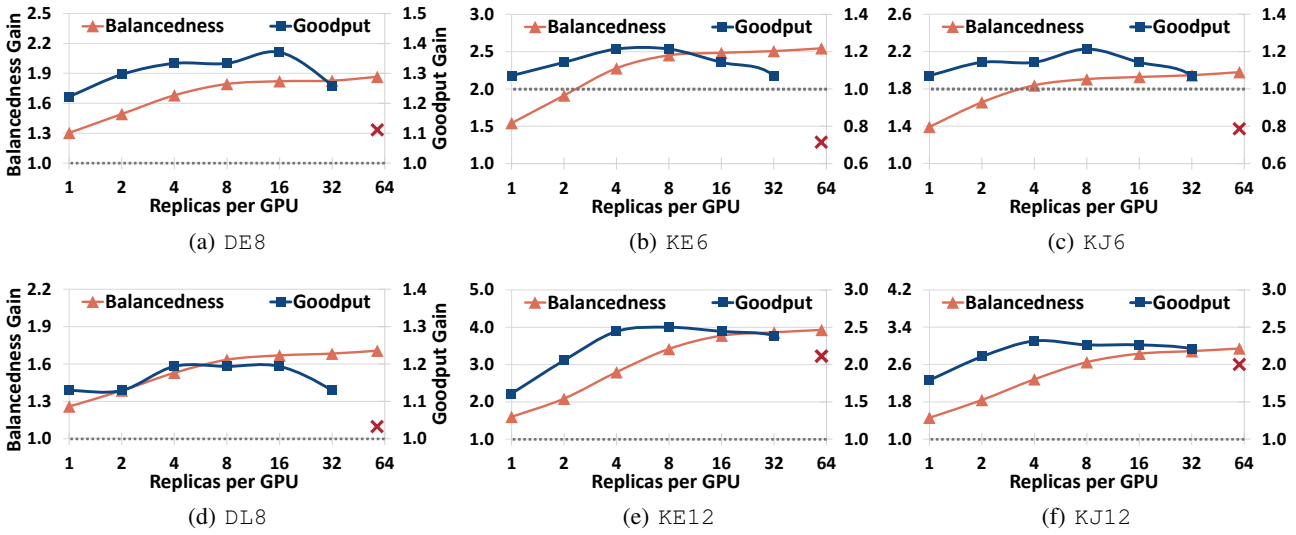


Figure 9. CRA end-to-end goodput gain (right, blue/red) and balancedness gain (left, orange) with different numbers of replicas allocated per GPU (replication factor R), normalized to BASE. Red $\times$ represents EPLB goodput (one replica per MoE layer per GPU). Goodput gains below the dotted line represent slowdowns compared to BASE. Number of replicas per GPU is presented on a log scale.

ness and end-to-end goodput gain of CRA at different values of the replication factor R, normalized to BASE. EPLB goodput is depicted by the rightmost red $\times$ in each figure. We make the following observations.

Excess replication leads to suboptimal goodput. The balancedness gain curves illustrate that CRA achieves most of the balancedness gains with far fewer replicas compared to EPLB. Allocating replicas excessively yields diminished benefits and impedes goodput because it reduces KV cache size. On D⋆8, K⋆6, and K⋆12 ($\star$ denotes any workload), EPLB reduces KV cache size by 19%, 75%, and 24% respectively as memory usage depends on model and cluster size. Goodput drops when the concurrency loss from a smaller KV cache outweighs the benefit of higher balancedness. This explains why EPLB achieves lower throughput than BASE in Figures 9(b) and 9(c) with K⋆6 — the 75% KV cache reduction outweighs the balancedness gains.

5.4 Scalability with Cluster Size

To answer Question ❸, we study how goodput scales with different cluster sizes and setups. Figures 8(a)-8(i) depict the throughput–TTFT curves for cluster sizes of 6, 8, and 12 in KE, KJ and KL. Since the model size is fixed, smaller clusters imply more experts and less KV cache per device. Kimi-K2 (K) has 384 experts without replication, so each GPU holds 8, 6, and 4 experts for cluster sizes of 6, 8, and 12, respectively. We also scale the sequence batch size by setting each node as a DP rank. We make two observations:

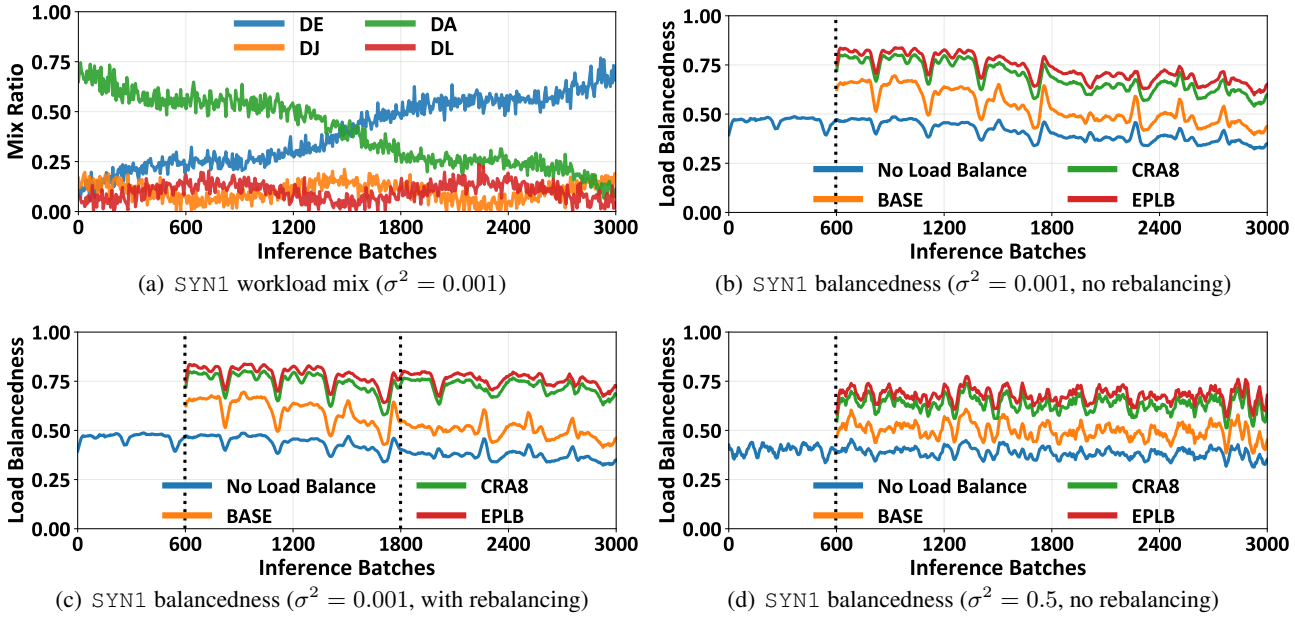
CRAFT scales goodput efficiently with large cluster sizes and outperforms EPLB. The goodput scaling of BASE is poor, with unstable and elevated TTFT. This confirms Observation 2 in Section 3.2: placement-only baseline suffers from high load imbalance at larger clusters. CRA8 scales by 1.65× and 1.6× on average when increasing the cluster size from 6 to 8 and from 8 to 12, respectively, outperforming EPLB and demonstrating good scalability.

The memory overhead of EPLB replication impedes goodput more significantly at smaller clusters. In Figures 8(a), 8(d), and 8(g), EPLB goodput is on average 46% lower compared to BASE. The KV cache in a 6-node cluster is very limited; EPLB further shrinks it by 75% through excessive replication, significantly reducing goodput (Section 5.3). In comparison, CRA8 allocates far fewer replicas, reducing KV cache size by only 6%. Thus, CRA8 still achieves 1.14× higher goodput than BASE on average.

5.5 Overhead Analysis

We answer Question ❹ and analyze both online and offline overhead of CRAFT. At inference time, CRAFT incurs no additional overhead. We present an inter-token-latency (ITL) evaluation in Appendix C. During initialization, the overhead of CRAFT is dominated by replication benefit estimation, as we sweep replica counts and replay the load distribution across all layers (Section 4.2.1). In our experiments, this process takes approximately 10 seconds, which is negligible since framework initialization typically takes minutes. With online periodic rebalancing, the estimation overhead can be overlapped with running batches on CPU.

5.6 Impact of Workload Shifts



(a) SYN1 workload mix ($\sigma^2 = 0.001$)

(b) SYN1 balancedness ($\sigma^2 = 0.001$, no rebalancing)

(c) SYN1 balancedness ($\sigma^2 = 0.001$, with rebalancing)

(d) SYN1 balancedness ($\sigma^2 = 0.5$, no rebalancing)

Figure 10. SYN1 shifting workload mix ratios and achieved load balancedness over time on DeepSeek-R1-671B (D) across 8 nodes. Figure 10(a) depicts the ratio of each dataset over time with burstiness variance $\sigma^2 = 0.001$. Figures 10(b) - 10(d) depict the achieved load balancedness under various configurations. The dotted lines represent triggered online periodic rebalancing, and data lines are smoothed for clearer presentation. Load-balancing techniques (BASE, EPLB, CRA8) are activated after the initial warmup of 600 iterations. Workloads with higher σ^2 only exhibit higher degree of burstiness; the shift and diurnal patterns remain identical.

In this section, we perform an ablation of the impact of shifts in workload distributions on distribution-based load-balancing techniques, CRAFT and EPLB. In online deployments, workloads may change over time, leading to drift in load distributions. For example, prior datacenter studies (Weng et al., 2022; Xiang et al., 2025; Wang et al., 2012; Pitchumani et al., 2015) identify two common characteristics of datacenter workloads—long-term diurnal shifts and short-term stochastic bursty arrivals. LLM-serving requests specifically may exhibit diurnal fluctuations and notably bursty request arrivals (Xiang et al., 2025). At the same time, consecutive requests are typically highly correlated due to shared context, common vocabulary, and request continuity (Tairin et al., 2025). To analyze these scenarios where (1) workload shifts exhibit stochastic burstiness, and (2) diurnal workload shifts that occur gradually over time, we mix the four evaluation datasets and create four synthetic shifting workloads: SYN1, SYN2, SYN3 and SYN4, each defined by the start and end ratios of datasets E, J, A, and L. We synthesize the workloads in a four-step process: (1) the dataset ratios are initially modeled as linear trends from their start ratios to their end ratios. (2) we assign sinusoidal waves with distinct phase shifts to model workload-specific diurnal patterns. (3) we inject zero-mean Gaussian noise $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ into the mix ratio of each dataset to model short-term bursts, where σ^2 is the variance (degree of burstiness) and I is the identity matrix ensuring independent noise across datasets. (4) we normalize the ratios of all datasets within each inference batch to sum to 1.0. For each synthetic workload, we evaluate two different burstiness levels: $\sigma^2 = 0.001$ and 0.5. Figure 10 depicts the workload mix ratios and the achieved load balancedness on SYN1. We use a rebalancing window of 1200 iterations, which is approximately 20 minutes on our setup — 2× longer than EPLB's 10-minute default (Section 4.3). Despite this less frequent rebalancing, we find that periodic rebalancing at this interval is sufficient to sustain high overall balancedness. The results of SYN2, SYN3 and SYN4 are included in Appendix D. We make three observations. First, for shifting workloads that have lower burstiness and more gradual distribution changes ($\sigma^2 = 0.001$), the balancedness degrades over time for both EPLB and CRAFT as expected (Figure 10(b)). With periodic online rebalancing, the balancedness improves to reflect the change in distribution (Figure 10(c)). Second, for shifting workloads that have higher burstiness ($\sigma^2 = 0.5$), the balancedness exhibits higher fluctuations where the short-term bursts become more significant, leading to drastic changes in the load distributions (Figure 10(d)). Thus, the contribution of each dataset to the load distribution becomes more even, and the initial plan learned by EPLB and CRAFT remains effective despite the shifting workload. Third, CRAFT achieves comparable load balancedness to EPLB, and better than the baseline with no load-balancing, across all evaluated configurations. CRA8 achieves this

while consuming 7.25× less memory than EPLB, enabling the benefit of a larger KV cache and improved throughput, as demonstrated in Sections 5.2 and 5.4.

5.7 CRAFT Speedup Breakdown

In this ablation, we break down the end-to-end speedup and analyze how different layer components benefit from CRAFT. Figure 11 depicts the speedups of total layer time and main MoE components with CRA8, normalized to BASE. The speedups are collected with NVIDIA Nsight Systems (NVIDIA Corporation, 2025) and aggregated over 10 iterations, all layers, and all GPUs. We make three observations: (1) MoE runtime only improves marginally because the token count is constant, and runtime is aggregated across GPUs and is approximately proportional to token count and compute. (2) All-to-All Dispatch exhibits moderate speedup, because replication eliminates network link congestion by evenly distributing load. (3) All-to-All Combine speeds up significantly, but its contribution to end-to-end speedup (Section 5.2) is limited as it constitutes only a fraction of total layer runtime in the baseline. Under high load imbalance, cold devices stall on All-to-All Combine, waiting for hot devices. When load is evenly distributed, MoE runtime aligns across devices, eliminating stalls.

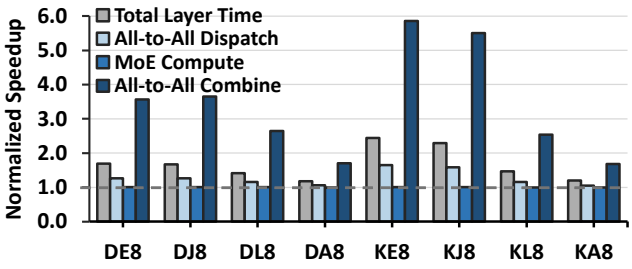


Figure 11. CRA8 speedup of total layer time (including non-MoE blocks) and time of each MoE block step, normalized to BASE.

6 Related Work

Expert Placement. Many existing works aim to achieve optimal expert placement during inference to reduce all-to-all traffic and mitigate load imbalance. A recent line of work reduces all-to-all traffic via topology-aware placement that co-locates experts with high activation affinity (Zhai et al., 2023; Nie et al., 2023; Hwang et al., 2023; Han et al., 2025). ExFlow (Yao et al., 2024) and MoETuner (Go & Mahajan, 2025) formulate placement as integer programs with topology and load constraints. Occult (Luo et al., 2025) constructs an expert-collaboration graph and derives placements by clustering. As discussed in Section 2.3, placement-only strategies cannot address load imbalance in extremely skewed workloads. However, we note that these approaches are orthogonal and can be integrated into CRAFT as a replacement for the current greedy placement (Section 4.2.2), further improving communication efficiency.

Expert Replication. Replication is widely used to mitigate load imbalance during distributed MoE training. FasterMoE (He et al., 2022) introduces shadow experts, selectively replicating hot experts to improve throughput. SwiftMoE (Skiadopoulos et al., 2025) adopts adaptive, non-uniform replication during training that tracks and replicates hot experts on the fly. However, efficient replication during training relies on overlapping replica weight transfers with gradient synchronization, and applying it during inference introduces an additional communication phase with prohibitive latency overheads. Concurrent work GraceMoE (Han et al., 2025) groups experts to reduce communication overhead. It allocates replicas dynamically for each layer, but restricts replication within a single group, since excessive replication impairs the effectiveness of grouping. CRAFT's fine-grained replication is orthogonal to the grouping strategy and can potentially improve load balancedness by replicating across multiple groups.

Routing Prediction and Alteration. Many works use load distributions and routing patterns during inference to predict future expert activations (Kong et al., 2024; Cao et al., 2025; Fang et al., 2025; Chen et al., 2025a; Song et al., 2024; Zhang et al., 2025; Yu et al., 2025; He et al., 2024). They typically combine host offloading with prefetching or caching. However, mispredictions can introduce high latency, and many target resource-constrained environments. Other works modify the routing mechanism to aid prediction and load balance (Hwang et al., 2024; Zhong et al., 2024; Zhou et al., 2022). These post-training techniques add deployment overhead and may compromise model accuracy.

Expert Sharding. Expert sharding is a variant of tensor parallelism that distributes expert weights across devices. Early MoE systems combine sharding with EP to scale the expert parameters (Lepikhin et al., 2020; Rajbhandari et al., 2022). Recent studies show that expert sharding can achieve near-perfect load balance during inference (Balmau et al., 2025). However, cross-node global token reduction incurs high inter-node communication overhead, significantly limiting its scalability (Singh et al., 2023; Jin et al., 2025).

7 Conclusion

In this work, we demonstrate the importance of fine-grained replication for mitigating expert load imbalance at low memory costs. We propose CRAFT, an intuitive and practical framework that allocates replicas based on estimated benefits to improve inference throughput. CRAFT can be flexibly integrated into mainstream serving frameworks, delivering significantly higher throughput and efficiency with no runtime overhead.