LIBRA: EFFECTIVE YET EFFICIENT LOAD BALANCING FOR LARGE-SCALE MOE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Distributed inference of large-scale Mixture-of-Experts (MoE) models faces a critical challenge: expert load imbalance. Numerous system-level approaches have been proposed for load balancing, but they either fail to achieve a satisfactory level of balance or introduce new bottlenecks due to the overhead of the load balancing mechanism itself. To this end, we propose *Libra*, a system that achieves near-optimal load balancing with minimal overhead. *Libra* adopts sophisticated mechanisms that accurately predict future expert activations and, based on these predictions, systematically perform load balancing. At the same time, it effectively hides the associated overhead by reconstructing the execution flow so that these costs are overlapped with MoE computation. Evaluations with two large-scale state-of-the-art MoE models on 8 H200 GPUs demonstrate that *Libra* improves throughput by up to 19.2%.

1 INTRODUCTION

The Mixture-of-Experts (MoE) architecture has become a cornerstone for state-of-the-art Large Language Models (LLMs) such as DeepSeek-V3, Qwen3MoE, and GLM-4.5 (DeepSeek-AI et al., 2025; Yang et al., 2025; GLM-4.5 Team et al., 2025). Through sparse activation, MoE enables models to scale to trillions of parameters while keeping the training and inference computation cost manageable (Du et al., 2022; The Mosaic Research Team, 2024; Jiang et al., 2024; Fedus et al., 2022; Lepikhin et al., 2020; Rajbhandari et al., 2022).

At the same time, the dynamic nature of MoE models introduces a key deployment challenge: *load imbalance*. One common way to scale MoE inference is through Expert Parallelism (EP), in which experts within MoE layers are partitioned across multiple GPUs. Under this setup, load imbalance arises when a disproportionate number of tokens are assigned to a few *hot* experts, causing the GPUs hosting them to become stragglers that determine the end-to-end latency.

Existing system-level load balancing approaches suffer from fundamental limitations, proving to be less effective and/or inefficient (DeepSeek, 2025; Li et al., 2023; Doucet et al., 2025). Some approaches fail to achieve satisfactory balance because they rely on ineffective heuristics, leaving substantial room for improvement (DeepSeek, 2025; Li et al., 2023). Others achieve a considerable level of balance but introduce new bottlenecks due to the additional operations required for load balancing (Doucet et al., 2025).

To address these challenges, we propose *Libra*, a system that achieves near-optimal balance with virtually zero overhead. In other words, it catches two birds at once: effective load balancing and efficient realization of that mechanism. For effectiveness, Libra predicts expert activations for the next layer with high accuracy by leveraging the observation that hidden states in LLMs evolve slowly across consecutive blocks, and based on these predictions, applies a sophisticated algorithm that yields near-optimal balance. For efficiency, Libra reconstructs the inference execution flow so that any overhead incurred by this process is hidden under MoE computations. In evaluations on eight benchmarks using two state-of-the-art MoE models, Qwen3MoE and GLM-4.5, on 8 H200 GPUs, Libra improves throughput by up to 19.2% compared to the state of the art.

2 BACKGROUND AND MOTIVATION

2.1 Expert Load Imbalance in MoE Inference

The Mixture-of-Experts (MoE) (Jacobs et al., 1991; Jordan & Jacobs, 1994; Shazeer et al., 2017) architecture enhances the capacity of Large Language Models (LLMs) by replacing the dense Feed-Forward Network (FFN) layer in a Transformer block with a sparse MoE layer. This layer consists of a large pool of subnetworks (experts) and a gating network that selectively activates a small subset of experts (e.g., top-k) for each input token. This sparse activation allows MoE models to scale to hundreds of billions or even trillions of parameters while keeping the computational cost for inference relatively low (Du et al., 2022; The Mosaic Research Team, 2024; Jiang et al., 2024; Fedus et al., 2022; Rajbhandari et al., 2022; Lepikhin et al., 2020). Consequently, large-scale open-source MoE models have achieved performance comparable to leading proprietary models like GPT-4.1, demonstrating the efficacy of this architecture (Yang et al., 2025; DeepSeek-AI et al., 2025; GLM-4.5 Team et al., 2025; Baidu ERNIE Team, 2025; OpenAI et al., 2025; Kimi Team et al., 2025).

However, the inherent mechanism that grants MoE models their efficiency—independent token assignment—introduces a significant challenge: *expert load imbalance*. Historically, this issue was addressed during the training phase by incorporating an auxiliary load-balancing loss term, which encouraged a more uniform distribution of tokens across all experts (Xue et al., 2024; Muennighoff et al., 2025; Fedus et al., 2022). While effective for balancing, this approach often came at the cost of model performance, as it could hinder the degree of expert specialization (Wang et al., 2024; Guo et al., 2025; Qiu et al., 2025; DeepSeek-AI et al., 2025).

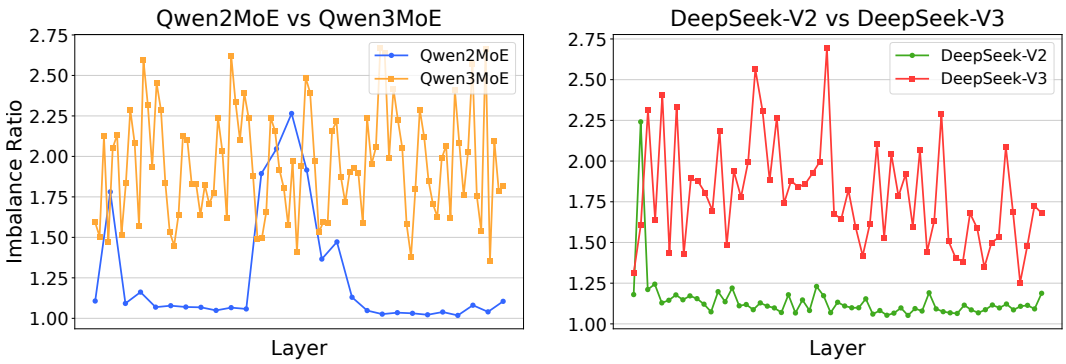

Figure 1: Intensified expert load imbalance in recent MoE models.

Reflecting this trade-off, recent state-of-the-art MoE models have moved away from strict load-balancing loss in favor of techniques that maximize expert specialization (Yang et al., 2025; GLM-4.5 Team et al., 2025; DeepSeek-AI et al., 2025). This aggressive pursuit of specialization has successfully pushed model performance to new heights but has the critical side effect of intensifying the expert load imbalance during inference. We measure this using the imbalance ratio, defined as the maximum load on any single GPU divided by the average load across all GPUs, where a value of 1.0 indicates a perfect balance. This trend is illustrated in Figure 1, showing a stark contrast in the imbalance ratio between newer MoE models and their predecessors (see Appendix A for experimental setup details). This reveals a fundamental trade-off: *achieving state-of-the-art performance in large MoE models exacerbates the expert load imbalance*, a problem projected to become more severe as models advance.

To efficiently serve large-scale MoE models across multiple GPUs, the standard strategy is a hybrid approach that applies Expert Parallelism (EP) to MoE layers and Data Parallelism (DP) to non-MoE layers (e.g., self-attention) (Perplexity AI, 2025; SGLang Team, 2025; Doucet et al., 2025; Li et al., 2025). While this strategy is essential for managing the massive parameter counts of these models, its performance is highly susceptible to expert load imbalance, which consequently becomes a critical performance bottleneck (Doucet et al., 2025). This issue stems from the synchronous execution of the MoE layers, forcing all devices to wait for the most heavily loaded GPU—the one hosting the hot expert(s)—to finish its computation. This phenomenon, known as the *staggler effect*, leads to significant idle time on less-loaded workers, severely degrading end-to-end latency and overall system throughput. Consequently, mitigating this staggler effect has become a central challenge in MoE inference, prompting the exploration of various system-level solutions.

2.2 System-Level Solutions for Load Balancing and Limitations

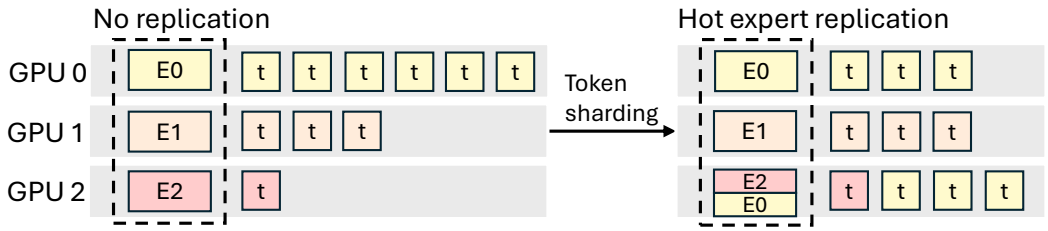

Figure 2: Hot expert replication and token sharding

A number of studies have attempted to mitigate load imbalance in MoE inference through system-level techniques. Most of these approaches employ *hot expert replication*. Figure 2 illustrates this approach: instead of assigning each expert to an unique GPU without redundancy, hot experts (i.e., experts that are likely to receive many tokens) are replicated across multiple GPUs. During MoE execution, tokens routed to these hot experts can then be distributed across replicas on different GPUs—a process we refer to as *token sharding*. This alleviates bottlenecks that would otherwise arise if a single GPU were forced to process a disproportionate number of tokens.

In the following, we discuss three representative works that constitute the most widely used and/or state-of-the-art techniques. They share the above approach but differ in how they perform expert replication and token sharding. While these methods present promising results, they also exhibit notable limitations, as their strategies for replication and sharding achieve only limited effectiveness and/or efficiency.

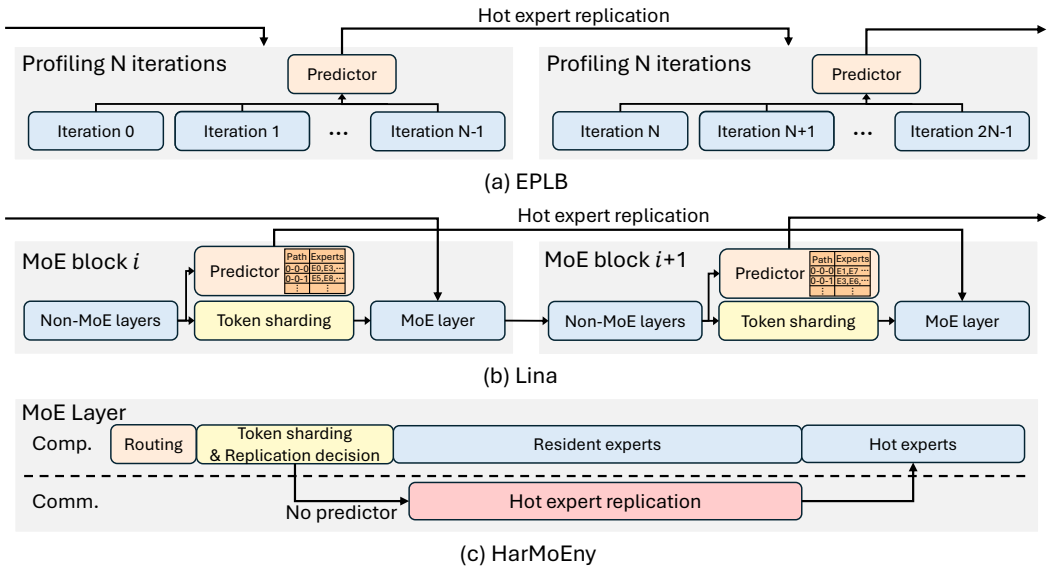

Figure 3: Overview of EPLB, Lina, and HarMoEny

EPLB. Expert Placement Load Balancer (EPLB) (DeepSeek, 2025) periodically performs expert replication based on historical data. Figure 3(a) illustrates this process. It profiles expert popularity over a fixed number of iterations (N in the figure) and uses the aggregated statistics to guide expert replication. However, this strategy is less effective because past popularity does not capture the instantaneous and dynamic variations across requests. EPLB's token sharding is also limited in effectiveness: once experts are replicated, tokens routed to them are randomly distributed across the GPUs holding replicas. In short, while EPLB is relatively efficient, it is not effective in either expert replication or token sharding.

Lina. Figure 3(b) illustrates how Lina (Li et al., 2023) performs expert replication. Specifically, Lina replicates experts for the upcoming block: when executing MoE computation for Block i, it performs expert replication for Block $i+1$, thereby removing replication overhead from the critical path. For this purpose, it relies on a pre-constructed lookup table that tracks each token's expert-selection-path. This path, which is the sequence of experts a token has selected in previous few layers (e.g., from layer i-4 to layer i-1), is used to predict which experts will be popular in the current layer (layer i). Such expert replication, however, is limited in effectiveness. In our evaluations on eight different benchmarks with two models (Qwen3MoE and GLM-4.5), Lina's prediction accuracy falls below 50% (43.7% and 11.8%, respectively). Details of the experimental setup are provided in Appendix A.

For token sharding, Lina simply distributes tokens uniformly across all replicas (e.g., assigning an equal number of tokens to each). Similar to the random token sharding in EPLB, this strategy is largely oblivious to the actual GPU loads, leaving significant room for improvement.

HarMoEny. Figure 3(c) illustrates HarMoEny (Doucet et al., 2025), which makes decisions on expert replication based on exact routing results. In other words, expert replication is performed only after the current input and its routing outcomes become available. This design makes replication highly effective, as it is guided by exact information. To prevent replication overhead from appearing on the critical path, HarMoEny first performs MoE computation for tokens routed to resident experts, while replication of hot experts proceeds in parallel. Once replication completes, the MoE computation for hot experts is executed. Token sharding is also effective, as HarMoEny employs a sophisticated algorithm that computes a near-optimal token assignment.

Despite this effectiveness, HarMoEny suffers from efficiency issues. The overhead required to realize such accurate replication and token sharding is substantial. After routing and before completing MoE computation, HarMoEny must perform decision making through complex algorithms to determine both expert replication and token sharding. Because these algorithms run synchronously on the GPU, they extend the critical path and introduce new bottlenecks.

3 DESIGN

In this section, we propose Libra, a system for MoE inference that achieves near-optimal load balancing with minimal overhead. Unlike prior methods, Libra simultaneously addresses both effectiveness and efficiency in hot expert replication and token sharding, thereby overcoming the key limitations of existing approaches. For expert replication, Libra follows the spirit of Lina by prefetching hot experts for the next layer while processing the current layer, based on prediction. This design avoids the inefficiency observed in HarMoEny, which cannot exploit Grouped-GEMM optimizations. At the same time, Libra employs a more accurate prediction mechanism than Lina, thereby improving effectiveness. For token sharding, Libra adopts the strategy of HarMoEny but effectively hides its cost from the critical path by restructuring the execution flow and leveraging the CPU.

The remainder of this section is organized as follows. Section 3.1 explains Libra 's execution flow. Section 3.2 describes the hot expert replication mechanism, and Section 3.3 details the token sharding mechanism.

3.1 LIBRA EXECUTION FLOW

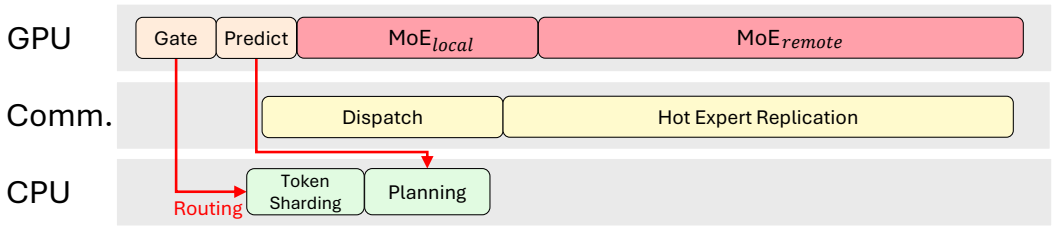

Figure 4: Two-Stage Locality-Aware Execution of Libra

Figure 4 illustrates the execution flow of Libra. The key novelty lies in its *Two-Stage Locality-Aware Execution*, which splits MoE computation into two phases based on token locality: MoE_{local} and MoE_{remote}. The MoE_{local} phase processes tokens routed to experts residing on the same GPU as the tokens themselves, while the MoE_{remote} phase handles tokens that must be dispatched to other GPUs. After decomposing the computation into these two phases, Libra first performs MoE_{local}, followed by MoE_{remote}.

This execution flow creates a time window in which the overhead of sophisticated token sharding mechanism can be hidden. In the conventional execution flow with load balancing, MoE computation begins only after token sharding completes. By contrast, in Libra, the MoE_{local} phase has no dependency on token sharding; it can start immediately after the gating function, with only the MoE_{remote} phase depending on the results of token sharding and Dispatch operation. This enables token sharding mechanism to run in parallel with MoE_{local}.

To further enhance the effectiveness of this parallelism, Libra performs token sharding on the CPU rather than the GPU. In addition, Libra implements dispatch using AllGather (i.e., all tokens are transferred to all GPUs) instead of All2All, where tokens are sent only to their assigned GPU. While this design increases the raw communication volume of dispatch, its latency impact is negligible. More importantly, it improves efficiency by removing dispatch from the critical path: in an All2All-based implementation, dispatch must wait for token sharding to finish, whereas in the AllGather-based implementation, dispatch can also proceed in parallel with token sharding.

3.2 Hot Expert Replication

As mentioned, Libra follows the spirit of Lina for hot expert replication: performing expert replication for the next layer while processing the current layer, based on prediction. However, Libra departs substantially from Lina in both how the prediction is performed and how expert replication planning (i.e., determining which experts to replicate to which GPUs) is carried out.

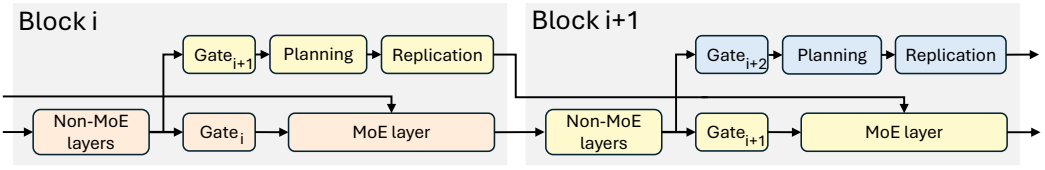

Figure 5: Hot expert replication of Libra with lookahead predictors

Predictor Design. Libra employs a lookahead predictor, leveraging a well-established property of Transformer-based LLMs: hidden states evolve slowly across layers (Liu et al., 2023b;b; Hwang et al., 2024). Figure 5 illustrates its concept. It speculatively executes the gating function of the next layer using the hidden states from the current layer, and then uses the results to determine which experts should be replicated across GPUs. This runtime-based approach achieves substantially higher accuracy than Lina's predictor (e.g. 70-80% vs 20-30%).

Locality-Aware Expert Replication Planning. During expert replication planning, Libra introduces an additional consideration beyond load balancing: locality enhancement. In other words, Libra not only balances load but also seeks to extend the MoE_{local} computation window, thereby providing more opportunity to hide token sharding overhead.

To this end, Libra performs expert replication planning in two phases. In the first phase, each GPU brings in $N \times \alpha$ experts that are most frequently activated by the tokens on that GPU and are not already resident on it, thereby extending the MoE_{local} computation window. Here, N denotes the maximum number of additional experts a GPU may host, determined by its available memory capacity and the allowable time window (a function of MoE computation time and communication bandwidth), while α is a hyperparameter that controls what fraction of N is allocated to the first phase. In the second phase, load balancing is performed iteratively: at each step, the hottest expert from the most heavily loaded GPU is selected for replication and placed on the least-loaded GPU among those that have not yet received N extra experts.

Figure 6 illustrates the expert replication planning process of Libra with an example. First, to enhance locality, Libra addresses the initial placement (left), where a significant portion of tokens on each GPU is routed to remote experts on other devices. Each GPU identifies its most requested remote experts and replicates them locally. For instance, GPU 0 replicates E4 from GPU 2 to serve its local tokens. This process converts remote tokens into local computations, securing the MoE_{local} computation window necessary to hide system overhead. Second, to establish a foundation for load balancing, the algorithm identifies and replicates heavily loaded experts to under-loaded GPUs. In the figure, this is shown by replicating expert E2 to an under-loaded device. This facilitates effective token sharding by allowing the workload from overloaded GPUs to be redistributed, ultimately enabling a near-perfect load balance.

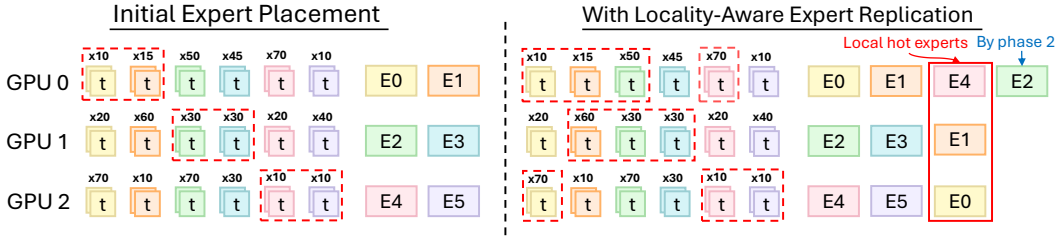

Figure 6: Locality-aware expert replication planning

3.3 Token Sharding

For token sharding, Libra adopts an algorithm similar to that used by HarMoEny, but with two key differences. First, Libra applies token sharding only to remote tokens. Second, this process is offloaded to the CPU.

Figure 7 explains the iterative greedy strategy of token sharding. The algorithm's main loop begins by checking if any GPU's load exceeds the target threshold (❷). If the system is balanced, the process terminates. Otherwise, it selects the most overloaded GPU, g_s, to resolve (❸). To find the most effective transfer, the algorithm enters an inner loop, starting by selecting the hottest remote expert, e, on g_s—the one accounting for the largest number of its remote tokens (❹). It then searches for an optimal destination: the least-loaded GPU (g_d) that hosts a replica of expert e and has enough capacity to accept new tokens (❺). A replica of the expert is necessary on the destination GPU to process the transferred tokens, and these replicas are enabled by hot expert replication. If a suitable destination is found, the algorithm calculates the number of tokens to transfer and updates the loads on both g_s and g_d (❻). Crucially, after each successful transfer, the algorithm returns to the main loop's start (❷) to re-evaluate the entire system's balance, ensuring it always addresses the most critical imbalance first. If no suitable destination is found (❼), it attempts to transfer the tokens for the next hottest expert on g_s (❽) until all options are exhausted, at which point it also returns to re-evaluate the global state. The full algorithm is detailed in Appendix C.

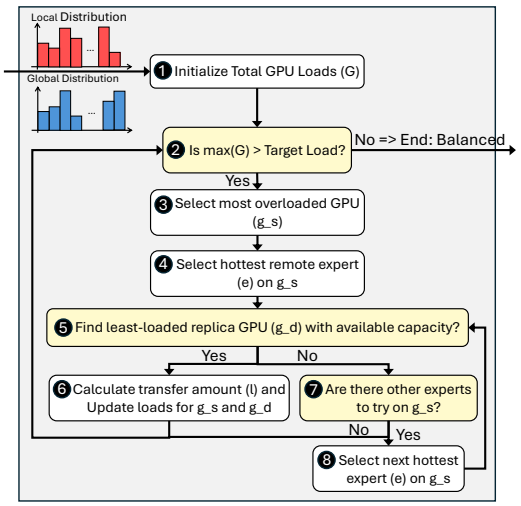

Figure 7: Logic flow of the iterative greedy rebalancing strategy.

4 IMPLEMENTATION DETAILS

We implement Libra atop the SGLang (v0.4.10) LLM serving framework. Our core mechanisms for expert replication planning and token sharding are implemented in Cython to ensure minimal overhead and are integrated as native modules during SGLang's build process.

To efficiently perform hot expert replication, we leverage PyTorch `SymmetricMemory` for copy engine-based P2P transfers. We employ a double-buffering strategy by pre-allocating two large buffers: an even buffer and an odd buffer. During the execution of an even-numbered MoE layer, original and duplicated experts are gathered in the even buffer to be processed via a high-performance Grouped-GEMM kernel. Concurrently, the system loads the necessary experts for the subsequent odd-numbered layer into the odd buffer. When processing an odd-numbered MoE layer, the roles are reversed: computation utilizes the odd buffer while the even buffer is populated for the next layer. This pipelining mechanism effectively hides the expert replication overhead by overlapping the P2P copy operations with the ongoing computation.

5 EVALUATION

We conduct a comprehensive evaluation to demonstrate the effectiveness of Libra. Our experiments are designed to answer three key questions: (1) How does Libra's prefill performance compare against baselines? (2) How does the prediction accuracy of Libra's speculative execution compare against existing methods like Lina? (3) How stable and robust is Libra's performance under workloads with dynamic and shifting token distributions?

5.1 SETUP

Model and Data. We evaluate Libra using two representative state-of-the-art large MoE models: Qwen3MoE (235B) (Yang et al., 2025) and GLM-4.5 (355B) (GLM-4.5 Team et al., 2025). To ensure coverage of a wide range of inputs, we use eight datasets: BookCorpus (Zhu et al., 2015), Codeforces (Penedo et al., 2025), DeepSeek-Prover (Xin et al., 2024), FineWeb (Penedo et al., 2024), GSM8K (Cobbe et al., 2021), HellaSwag(Zellers et al., 2019), HumanEvalPlus (Liu et al., 2023a), and LMSYS-Chat-1M (Zheng et al., 2023). All experiments are run using BF16 precision.

Environments. All experiments are conducted on a single node equipped with 8 NVIDIA H200-SXM5 GPUs, each with 141 GB of HBM3e memory. Intra-node communication leverages NVSwitch with 900 GB/s of P2P bandwidth.

Baselines. We compare Libra against three baselines. The vanilla MoE implementation in SGLang (v0.4.10) serves as our foundational baseline, representing a standard system without advanced load balancing. For the widely adopted proactive expert replication approach, we use EPLB DeepSeek (2025) from its implementation within SGLang. As the strongest baseline, we evaluate against Lina Li et al. (2023). Since no public implementation of Lina is available, we developed an in-house version built on SGLang, faithfully following the description in the original paper.

Metrics. The primary performance metric is prefill throughput, measured in tokens per second. We assume a prefill-decode disaggregated serving system where the prefill and decode phases are separated and handled by different GPUs (Zhong et al., 2024b; Hu et al., 2025; Feng et al., 2025). and therefore target only the prefill phase in our evaluation. We also measure the imbalance ratio (defined as the load of the most burdened GPU divided by the average load across all GPUs) to analyze the effectiveness of load balancing.

5.2 RESULTS

Throughput Results. First, Libra substantially improves the performance of the prefill phase. As shown in Figure 8, Libra achieves the highest throughput across all tested models and datasets. The four datasets for this evaluation were specifically chosen from a total of eight because they exhibited the most severe expert load imbalance, allowing for a clear demonstration of performance differences between the laod balancing systems. Notably, This evaluation was conducted under an experimental setup deliberately designed to be highly advantageous for the baseline systems. For Lina, its expert-selection-path table was constructed using the same dataset as the evaluation and

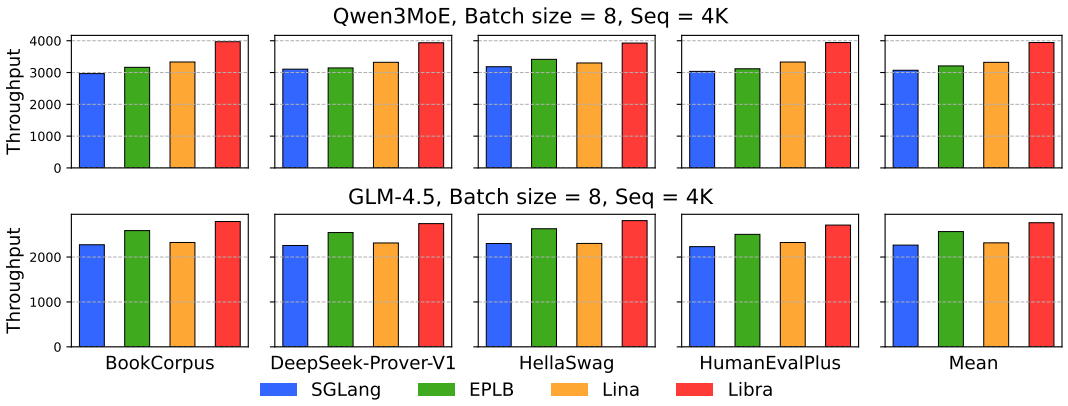

Figure 8: Prefill throughput of Libra and baselines.

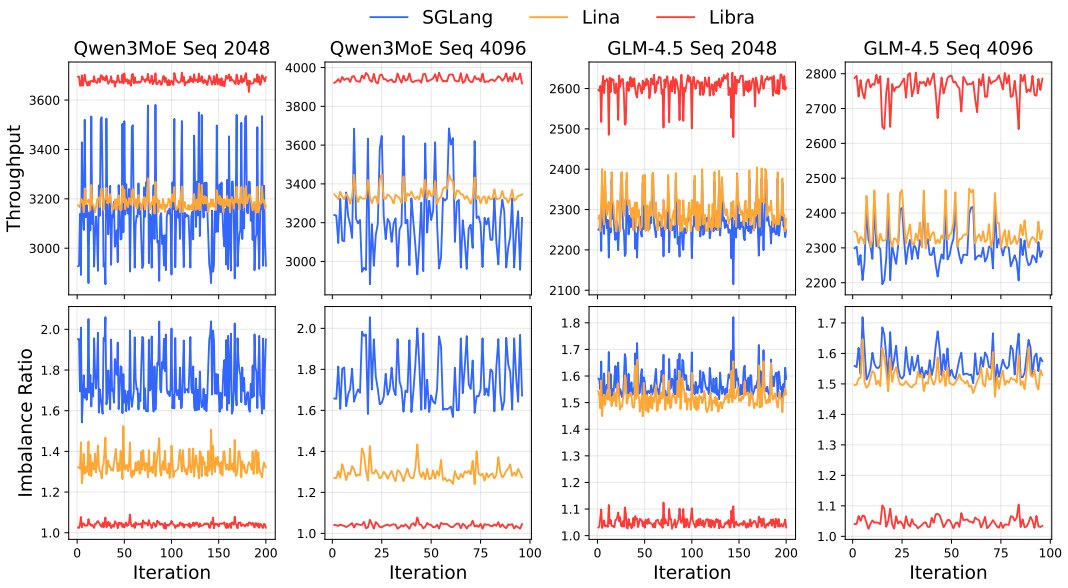

Figure 9: Throughput fluctuation and imbalance ratio under a dynamic workload.

its expert prefetching allowed each GPU to hold 8 additional experts. Similarly, EPLB's expert placement was determined by profiling on the identical dataset. It statically placed 8 identified hot experts—one on each of the 8 GPUs—across every MoE layer. This static replication, however, results in higher memory consumption compared to the dynamic approaches of Lina and Libra because of a large number of layers. To meet the memory budget with Lina, Libra was configured with N set to 8 and α set to 0.5. Despite these favorable, even biased, conditions for the baselines, Libra consistently and significantly outperforms them for both Qwen3MoE and GLM-4.5. These results confirm that Libra's dynamic load balancing effectively resolves the straggler problem, leading to superior computational efficiency and overall system performance.

Per-Iteration Fluctuation. Libra also delivers significantly higher and more stable throughput. Figure 9 illustrates this robustness using a mixed-dataset designed to simulate dynamic shifts in expert load imbalance. For this test, Lina's expert-selection-path table was constructed using a workload created by mixing one-eighth of the build split from each of the eight datasets. This comparison centers on dynamic systems like Lina and Libra, excluding EPLB, as its reliance on periodic profiling and static reconfiguration is ill-suited for workloads where imbalance shifts frequently and intensely. While baseline systems suffer from volatile performance that plummets as the imbalance ratio spikes, Libra effectively decouples its performance from the input distribution. By maintaining

Qwen3MoE		
Dataset	**Lina**	**Libra**
BookCorpus	47.3	91.7
DeepSeek-Prover-V1	45.4	86.5
HellaSwag	37.5	86.6
HumanEvalPlus	44.5	87.0

GLM-4.5		
Dataset	**Lina**	**Libra**
BookCorpus	11.7	79.6
DeepSeek-Prover-V1	12.7	72.9
HellaSwag	11.5	76.6
HumanEvalPlus	11.2	72.7

Table 1: Prediction accuracy.

a near-perfect imbalance ratio close to 1.0, Libra provides consistently high throughput, proving its resilience to the dynamic nature of expert load imbalance.

Prediction Accuracy. We quantitatively evaluate the accuracy of Libra's predictor for hot expert replication compared against Lina. Table 1 presents a direct comparison between the prediction accuracy of Libra's speculative execution-based approach and Lina's offline-constructed lookup table based method on the Qwen3MoE and GLM-4.5 models. Accuracy is defined as the fraction of correctly predicted experts for each token, where the set of actually activated experts serves as the ground truth. We construct Lina's expert-selection-path table on the build split of mixed dataset, then evaluate on the evaluation split of four datasets. Evaluation setup is detailed in Appendix A.

The results reveal a stark contrast between the two methods. Libra's predictor consistently achieves a high and stable accuracy in the 70-90% range across all datasets, demonstrating the effectiveness of its runtime prediction based on current-layer hidden states. In contrast, Lina's lookup-based predictor shows noticeably lower accuracy across datasets, highlights the critical generalization limitations of an offline-built lookup table. This effect is more pronounced on GLM-4.5, where Lina successfully identifies fewer than one correct experts between top-8 experts. These findings confirm that Libra 's dynamic prediction mechanism is significantly more robust and reliable for handling diverse and unpredictable workloads.

6 Conclusion

We introduces *Libra*, a dynamic load balancing system addressing intensified expert load imbalance in modern Mixture-of-Experts (MoE) models. *Libra* proposes *Two-Stage Locality-Aware Execution*, an innovative paradigm hiding dynamic load balancing overhead by overlapping it with ongoing GPU computations. This is enabled by two synergistic core components: *Hierarchical Expert Prefetcher*, using speculative execution for highly accurate (70-80%) expert prediction to strategically prefetch the necessary experts for the next layer, and *Adaptive Token Rebalancer*, computing an optimal assignment schedule by accounting for processed local token load. Implemented on SGLang, *Libra* demonstrates state-of-the-art performance, reducing the prefill throughput by up to 19.2% while maintaining an imbalance ratio of nearly 1.0 under dynamic workloads. *Libra* thus achieves dynamic load balancing with virtually zero-overhead for efficient serving of large-scale MoE models.

REFERENCES

Baidu ERNIE Team. Ernie 4.5 technical report, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek. Expert parallelism load balancer (eplb), 2025. URL https://github.com/deepseek-ai/EPLB.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Zachary Doucet, Rishi Sharma, Martijn de Vos, Rafael Pires, Anne-Marie Kermarrec, and Oana Balmau. Harmoeny: Efficient multi-gpu inference of moe models, 2025. URL https://arxiv.org/abs/2506.12417.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient scaling of language models with mixture-of-experts. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5547–5569. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/du22c.html.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL http://jmlr.org/papers/v23/21-0998.html.

Jingqi Feng, Yukai Huang, Rui Zhang, Sicheng Liang, Ming Yan, and Jie Wu. Windserve: Efficient phase-disaggregated llm serving with stream-based dynamic scheduling. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 1283–1295, 2025.

GLM-4.5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

Hongcan Guo, Haolang Lu, Guoshun Nan, Bolun Chu, Jialin Zhuang, Yuan Yang, Wenhao Che, Sicong Leng, Qimei Cui, and Xudong Jiang. Advancing expert specialization for better moe, 2025. URL https://arxiv.org/abs/2505.22323.

Xiannan Hu, Tianyou Zeng, Xiaoming Yuan, Liwei Song, Guangyuan Zhang, and Bangzheng He. Bestserve: Serving strategies with optimal goodput in collocation and disaggregation architectures. *arXiv preprint arXiv:2506.05871*, 2025.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-gated moe: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 1018–1031. IEEE, 2024.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

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, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6(2):181–214, 1994. doi: 10.1162/neco.1994.6.2.181.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu,

Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020. URL https://arxiv.org/abs/2006.16668.

Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed MoE training and inference with lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pp. 945–959, Boston, MA, July 2023. USENIX Association. ISBN 978-1-939133-35-9. URL https://www.usenix.org/conference/atc23/presentation/li-jiamin.

Yan Li, Pengfei Zheng, Shuang Chen, Zewei Xu, Yuanhao Lai, Yunfei Du, and Zhengang Wang. Speculative moe: Communication efficient parallel moe inference with speculative token and expert pre-scheduling, 2025. URL https://arxiv.org/abs/2503.04398.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572. Curran Associates, Inc., 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023b.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. Olmoe: Open mixture-of-experts language models, 2025. URL https://arxiv.org/abs/2409.02060.

OpenAI, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park

Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

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)*, pp. 118–132, 2024. doi: 10.1109/ISCA59077.2024.00019.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces. https://huggingface.co/datasets/open-r1/codeforces, 2025.

Perplexity AI. Efficient and portable mixture-of-experts communication, 2025. URL https://www.perplexity.ai/hub/blog/efficient-and-portable-mixture-of-experts-communication.

Zihan Qiu, Zeyu Huang, Bo Zheng, Kaiyue Wen, Zekun Wang, Rui Men, Ivan Titov, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Demons in the detail: On implementing load balancing loss for training specialized mixture-of-expert models, 2025. URL https://arxiv.org/abs/2501.11873.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18332–18346. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/rajbhandari22a.html.

SGLang Team. Deploying deepseek with pd disaggregation and large-scale expert parallelism on 96 h100 gpus, 2025. URL https://lmsys.org/blog/2025-05-05-large-scale-ep/.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

The Mosaic Research Team. Introducing dbrx: A new state-of-the-art open llm, March 2024. URL https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm. Databricks Mosaic Research Blog.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts, 2024. URL https://arxiv.org/abs/2408.15664.

Huajian Xin, Daya Guo, Zhihong Shao, Z.Z. Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Advancing theorem proving in LLMs through large-scale synthetic data. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL https://openreview.net/forum?id=TPtXLihkny.

Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. Openmoe: An early effort on open mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2402.01739.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric. P Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. Lmsys-chat-1m: A large-scale real-world llm conversation dataset, 2023.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, Santa Clara, CA, July 2024a. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

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

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

A Experimental Setup Details

Models. For Figure 1 1, we compare the imbalance ratio across recent MoE families by pairing models with different load-balancing strategies during training. Within the Qwen family, we evaluate Qwen2MoE and Qwen3MoE. Qwen2MoE has 57 billion total parameters and 14 billion activated parameters, and it utilizes a micro-batch level auxiliary load-balancing loss during training to ensure load balance. In contrast, Qwen3MoE, with 235 billion total and 22 billion activated parameters, forgoes this term in favor of a global-batch load balancing loss, a strategy that maximizes expert specialization while still addressing balance. The DeepSeek family shows a similar contrast: DeepSeek-V2 (236B total, 21B activated) also employs an auxiliary load-balancing loss, whereas DeepSeek-V3 (671B total, 37B activated) improves training efficiency by adopting the auxiliary-loss-free load balancing technique. For all other experiments, we evaluate Libra and baselines on Qwen3MoE and GLM-4.5, models that are trained without such fine-grained balancing losses. GLM-4.5 has 355 billion total parameters and 32 billion activated parameters.

Datasets. We evaluate *Libra* and baselines on eight datasets: BookCorpus (Zhu et al., 2015), Codeforces (Penedo et al., 2025), DeepSeek-Prover (Xin et al., 2024), FineWeb (Penedo et al., 2024), GSM8K (Cobbe et al., 2021), HellaSwag(Zellers et al., 2019), HumanEvalPlus (Liu et al., 2023a), and LMSYS-Chat-1M (Zheng et al., 2023). Unless noted otherwise, each dataset contributes a total of 2.0M tokens. The first 1.6M tokens form the build split for EPLB offline profiling and for constructing Lina's prediction table. The next 0.4M tokens form the evaluation split used for testing. Figure 1 is the only exception, which uses a 0.07% subset of the BookCorpus dataset. For Table 1, we build Lina's expert-selection-path table is on a mixed workload that uniformly interleaves the build splits of all eight datasets. We then evaluate Lina and Libra separately on each dataset's evaluation split. Figure 8 reports results on BookCorpus, DeepSeek-Prover-V1, HellaSwag, and HumanEvalPlus dataset. Figure 9 uses a shuffuled workload constructed from all eight datasets.

Environments. All expertiments use a single-node system equipped with eight NVIDIA H200-SXM5 GPUs, each with 141GB of HBM3e memory. The server configuration is summarized in Table 2.

Table 2: Server configuration

CPU	2× Intel Xeon Platinum 8580 (128 cores)
GPU	8× NVIDIA H200-SXM5-141GB
System Memory	32× 64 GB DDR5-5600 (total 2,048 GB)
GPU Memory	141GB HBM3e per GPU
GPU Interconnect	Connected with NVSwitch (900GB/s bandwidth)

Metrics. Table 1 reports accuracy, defined as the fraction of tokens whose ground-truth experts appear in the predicted top-k set. Figure 8 and Figure 9 report prefill throughput in tokens per second. Throughout these experiments, we adopt Prefill-Decode disaggregation (Patel et al., 2024; Zhong et al., 2024a) setup, and therefore we evaluate preill only as we target prefill phase. Figure 1 and Figure 9 also report the imbalance ratio, defined as the load of the most heavily utilized GPU divided by the average load across all GPUs.

B Hierarchical Expert Prefetcher

The *Hierarchical Expert Prefetcher* optimally places experts on GPUs for the next MoE layer. This process is crucial for facilitating effective load balancing and extending the MoE_{local} computation window, which in turn hides system overhead. The algorithm operates in two main phases after an initial setup.

First, it duplicates "local hot experts"—those most frequently requested by a GPU's local tokens but residing on other GPUs—onto the source GPU itself. This strategically increases the number of local tokens that can be processed without inter-GPU communication, creating a sufficient time window for the *Adaptive Token Rebalancer* to execute in parallel without affecting the critical path.

Second, the algorithm iteratively balances the remaining load by duplicating global hot experts. It identifies the most overloaded GPU and replicates its hottest expert to the least-loaded GPU that has

available capacity. This ensures that the overall load is distributed as evenly as possible before the next layer's computation begins. The final output is an optimized expert placement map that serves as the foundation for the rebalancing stage.

Algorithm 1 Hierarchical Expert Prefetcher

Inputs: Predicted expert IDs for upcoming tokens next_topk_ids, Total number of experts E, Total number of GPUs G, Max duplicated experts per GPU N, Number of local hot experts to duplicate L.
Outputs: A binary matrix for expert placement on GPUs: $M_{\text{next}} \in \{0,1\}^{E \times G}$.

Calculate ExpertLoad$[g, e]$ (requests for expert e from GPU g) based on next_topk_ids.
Initialize M_{next} by assigning each expert to its home GPU.
for each GPU g_s **do** ▷ Phase 1: Duplicate Local Hot Experts
    Identify top L remote experts most requested by g_s.
    Duplicate these experts to g_s.
end for
Calculate initial GPU loads based on the current mapping in M_{next}.
$B \leftarrow$ Target balanced load per GPU.
for $i = 1$ to $(N - L) \times G$ **do** ▷ Phase 2: Balance Load via Iterative Duplication
    $g_{\text{src}} \leftarrow$ most overloaded GPU where load $> B$.
    if no such GPU exists **then break**
    end if
    $e \leftarrow$ expert contributing most to g_{src}'s remote load.
    $g_{\text{dst}} \leftarrow$ least loaded candidate GPU that can host e (respecting capacity N).
    if g_{dst} is found and duplicating e keeps Load$[g_{\text{dst}}] \leq B$ **then**
        Update M_{next} by duplicating e to g_{dst}.
        Update GPU loads to reflect newly localized computations.
    end if
end for
return M_{next}

C Adaptive Token Rebalancer

The *Adaptive Token Rebalancer* determines an optimal assignment for remote tokens to resolve load imbalance, operating on the expert placement map generated by the *Hierarchical Expert Prefetcher*. Its core strategy is an iterative greedy approach that ensures the final token distribution is as close to perfectly balanced as possible.

The algorithm begins by calculating the total load for each GPU. It then enters a loop that continues as long as any GPU's load exceeds a target threshold. Within the loop, it identifies the most overloaded GPU (g_s) and selects its hottest remote expert (e)—the one responsible for the largest portion of its remote token load. It then finds the least-loaded GPU (g_d) that already hosts a replica of expert e and has sufficient capacity.

A calculated number of tokens for expert e are then transferred from g_s to g_d, and the load states of both GPUs are updated. After each transfer, the algorithm restarts its loop to re-evaluate the global system state, ensuring it always addresses the most critical imbalance first. This process repeats until the loads are balanced or no further beneficial transfers can be made.

Algorithm 2 Adaptive Token Rebalancer

Require: Expert-to-GPU mapping $M \in \{0,1\}^{E\times G}$, Local GPU loads $L \in \mathbb{Z}^G$, Remote expert loads $R \in \mathbb{Z}^{G\times E}$, Average target load $B \in \mathbb{R}$, Imbalance tolerance $\varepsilon \in (0,1)$
Ensure: Rebalanced loads R, G, and maximum load $t_{\max}$

$G[g] \leftarrow L[g] + \sum_e R[g,e]$ for all $g \in G$ ▷ Initialize total GPU loads
while $\max_g G[g] > (1+\varepsilon)B$ **do**
  moved $\leftarrow$ false
  for each $g_s \in \{g \mid G[g] > B\}$ in descending order of $G[g]$ **do**
    for each $e \in \{e \mid R[g_s, e] > 0\}$ in descending order of $R[g_s, e]$ **do**
      $\mathcal{C} \leftarrow \{g \neq g_s \mid M[e,g] = 1\}$ ▷ Find candidate destination GPUs
      if $\mathcal{C}$ is not empty **then**
        $g_d \leftarrow \arg\min_{g\in\mathcal{C}} G[g]$ ▷ Select least-loaded destination
        cap $\leftarrow B - G[g_d]$ ▷ Calculate destination's remaining capacity
        if cap > 0 **then**
          $l \leftarrow \min(R[g_s,e], \text{cap})$ ▷ Determine amount to move
          $R[g_s,e] \leftarrow R[g_s,e] - l$; $R[g_d,e] \leftarrow R[g_d,e] + l$
          $G[g_s] \leftarrow G[g_s] - l$; $G[g_d] \leftarrow G[g_d] + l$ ▷ Perform the token transfer
          moved $\leftarrow$ true
          break ▷ Exit inner loop to re-evaluate the most overloaded GPU
        end if
      end if
    end for
  end for
  if not moved **then**
    break ▷ Converged or stuck, exit outer loop
  end if
end while
$t_{\max} \leftarrow \max_g G[g]$; **return** $R, G, t_{\max}$