MoE-GPS: Guidlines for Prediction Strategy for Dynamic Expert Duplication in MoE Load Balancing

Haiyue Ma
Princeton University
hm1@princeton.edu

Zhixu Du
Duke University
zhixu.du@duke.edu

Yiran Chen
Duke University
yiran.chen@duke.edu

Abstract

In multi-GPU Mixture-of-Experts (MoE) networks, distributing experts across GPUs leads to load imbalance as token assignments vary. Recent methods address this by duplicating popular experts on additional GPUs, requiring accurate prediction of token distributions before routing. This paper examines the tradeoffs between prediction strategy, accuracy, overhead, and system performance. We introduce **MoE-GPS**, a framework that quantifies these impacts and identifies optimal predictor designs for various system settings. Our results highlight **Distribution-Only Prediction**, which predicts coarse token distribution with much lower overhead than Token-to-Expert Prediction, achieving 23% faster inference on the Mixtral 8×7B MMLU dataset.

1 Introduction

Mixture-of-Experts (MoE) [10, 12, 4, 21] models reduce the computation of Large Language Models (LLMs) by activating only a subset of experts for each token, typically using Expert Parallelism (EP) [14] to distribute Feed Forward Network layers across GPUs. However, skewed token-to-expert mappings often cause significant load imbalance [20, 1], especially as modern LLMs incorporate increasing numbers of experts [15]. While training-time techniques such as auxiliary losses [24] can proactively balance loads, inference has fixed expert-to-token mappings and imbalance and underutilization are unavoidable. Expert duplication [1, 25, 6, 16] — a common solution to balance loads — requires accurate, timely predictions of token-to-expert distributions, where increasing predictor complexity improves balance at the expense of overhead. The optimal prediction strategy thus depends on factors like workload patterns, hardware topology, and communication cost.

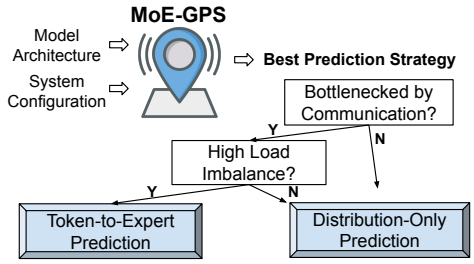


Figure 1: MoE-GPS guidelines for selecting optimal expert prediction strategies that minimizes end-to-end inference latency based on model and hardware characteristics.

Despite its importance, there is currently no systematic method to model the runtime implications of MoE load imbalance, and to help choose the best predictor for different workload and system setups. We present **MoE-GPS**, a framework that simulates end-to-end MoE inference performance with imbalance, and guides the selection of expert prediction strategies that yield the shortest runtime. Built on top of an architectural simulator, LLMCompass [27], MoE-GPS models the runtime tradeoffs among prediction strategies, accuracy, and overhead. Given an arbitrary model architecture and hardware setup, MoE-GPS identifies the strategy that delivers the best system performance.

ML For Systems workshop at Neural Information Processing Systems (NeurIPS 2025).

With insights from MoE-GPS, we advocate for **Distribution-Only Prediction** strategy, which only predicts the coarse-grained token distribution across experts instead of exact token-to-expert mappings (Token-to-Expert Prediction). This lightweight approach is particularly effective when communication is not a bottleneck, because it reduces prediction complexity and still improves compute load balancing. Exact token-level prediction optimizes both computation and communication at the cost of higher overhead, which becomes more favorable when communication cost dominates. In addition, we observe that Distribution-Only Prediction performs better with more balanced workloads. Our contribution can be summarized as follows:

- We propose **MoE-GPS**, a system performance simulation framework that selects the optimal expert prediction strategy to minimize inference latency.
- We identify and validate the effectiveness of **Distribution-Only Prediction** as a lightweight alternative to Token-to-Expert Prediction, offering better scalability and efficiency under varying system bottlenecks.

2 Background

Load Imbalance in Inference Prefill Is Critical. In MoE networks, load imbalance occurs in both training and inference, but while training can proactively distribute tokens to less-popular experts, inference uses a fixed token-to-expert mapping. This work focuses on the inference prefill stage without changing the routing. As decode involves much fewer tokens and is latency-critical, compute imbalance is less of an issue here.

Expert Parallelism Leads to Imbalance. Our setup uses Tensor Parallelism for Attention layers and Expert Parallelism (EP) for FFN layers. TP avoids duplicating KV cache, and Ring All Reduce [19] is used for communication efficiency. EP for FFN provides lower communication latency and preserves expert matrix structure but causes load imbalance since experts are tied to GPUs. Although hybrid (TP+EP) schemes exist [17], for simplicity, this work assumes TP-only for Attention and EP-only for FFN, and the insights are generalizable.

Quantifying Imbalance. To measure GPU load imbalance, we define *skewness* as $\frac{\#\text{ of tokens in the most popular expert}}{\#\text{ of average tokens per experts}}$. Skewness directly scales both FFN compute and communication bottlenecks since the busiest GPU determines overall layer latency.

Performance Impacts of Load Imbalance. For perfect balance (skewness = 1), each GPU transfers $(N-1)/N^2$ of its tokens in a fully connected multi-GPU setup (N is the number of GPUs), but skewed workloads cause longer runtimes for communication and compute on the busiest GPU. The communication time for routing and shuffles after FFN scales with skewness as $(N-1)\cdot\text{skewness}/N^2$.

Current Solutions. To address runtime imbalance, industry approaches often duplicate popular experts across multiple GPUs, guided by predicted token-to-expert distributions. Recent methods include MoE-Prediction [1], Prophet [25], FlexMoE [16], SE-MoE [22], and FasterMoE [6, 5], which propose dynamic duplication and placement strategies. Other approaches mitigate imbalance during training through auxiliary load balancing [24], expert biasing [15], or by adjusting expert allocation for memory-limited scenarios [2]. Several alternative strategies, such as using expert buffers [9] or activation correlation models [26], are also proposed, but are outside the main focus of this paper.

3 Methodology

This section discusses our approach to modeling and analyzing expert prediction strategies in MoE inference. We describe the two different prediction strategies (Distribution-Only and Token-to-Expert). We also show normalized system performance obtained from a performance simulator, LLMCompass [27], to illustrate high-level trends of the runtime implications of expert duplication.

Approach to Expert Duplication. We integrate dynamic expert duplication into the MoE model by inserting a pre-trained predictor before Attention in each layer. For a chosen frequency, the predictor does inference with current batched inputs to predict distribution and guide the expert placement. Since dynamic duplication incurs extra inter-GPU communication for moving experts, existing works propose different intervals for prediction and moving, from every single batch [5, 25] to every 10 minutes [15], to balance overhead and effectiveness.

In this study, we assume single-batch prediction and placement frequency. Our simulator can be configured to model different frequencies of prediction and placement, by averaging out the overhead to multiple batches. In our example, Expert 1 has the most tokens, so it is replicated across multiple GPUs to evenly distribute the load. In general, given arbitrary token-to-expert mappings, experts can be duplicated to achieve per-GPU balance by iteratively shifting experts from overloaded to underloaded devices: keep duplicating the experts on GPUs with $> 1/N$ tokens to GPUs with $< 1/N$ tokens until all GPUs process the same amount of tokens.

Prediction Strategies and Tradeoffs. We explore two prediction strategies with distinct tradeoffs: *Distribution-Only Prediction*, which estimates static, aggregate expert usage; and *Token-to-Expert Prediction*, which targets exact token-level routing. The former has lower overhead and complexity and targets at compute imbalance, while the latter can reduce both compute and communication costs at the cost of high overhead.

Distribution-Only Prediction. Distribution-only prediction estimates per-expert token proportions (e.g., Expert 1: 75%) without specifying which tokens. This balances compute across GPUs but does not reduce communication, since tokens remain randomly scattered after ring all-reduce. We model each layer's expert-activation distribution as multinomial and estimate parameters via Maximum Likelihood Estimation (MLE), which models counts over discrete classes (expert selections) and chooses parameters maximizing the likelihood of the observed data. Formally, let p_i^l denote the probability of selecting expert i in layer l. Assuming i.i.d. token selections from this multinomial, the MLE is $\hat{p}_i^l = \frac{n_i^l}{N}$, where N is the total number of tokens and n_i^l counts activations of expert i.

We evaluate Distribution-Only Prediction on MMLU [7], Alpaca Eval [3], and SST2 [23] using Mixtral 8×7B [11]. For sequence length 512, average batchwise skewness is 1.388, 1.402, and 1.990, respectively. We report the layer-averaged *error rate* between train-set estimates and test-set empirical probabilities, defined as $\frac{|\hat{p}-p|}{1/\#\text{ of experts}}$ where higher values indicate less accurate estimation. For datasets without a dedicated test split, we randomly partition the train set 80/20.

We also simulate normalized end-to-end system performance for this setting (Mixtral 8×7B; batch size = 1; sequence length = 512; four A100 GPUs with NVLINK) using an augmented LLMCompass [27].

Token-to-Expert Prediction. Token-to-Expert Prediction exactly routes each token to its expert's GPU. With a predicted mapping, tokens go directly to the GPU hosting their experts, skipping the post–ring all-reduce scatter and saving both FFN compute and communication time . We cast expert selection as classification: predict the activated expert for each token in the batch processed by the MoE model. We study three model families: a simple probability model, a conditional probability model, and neural predictors.

Probability Model. Assign the expert with the highest global frequency in the training data, treating all tokens identically regardless of identity or position.

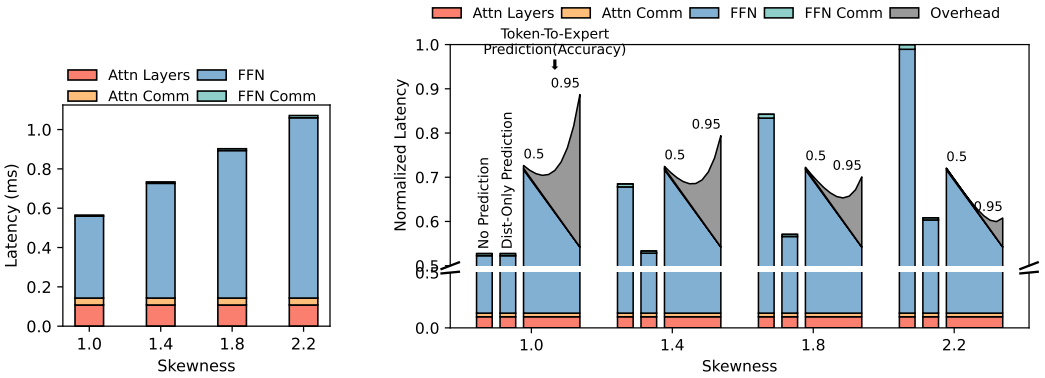
Conditional Probability Model. Condition on token index or position; for each token (or position), select the most frequent expert for that specific index in the training data.

Neural Networks. Train on pairs of token embeddings and expert activations using cross-entropy and Adam [13], to convergence. All samples are padded to sequence length 512. We evaluate simple FFNs and LSTMs [8].

4 Results

In this section, we determine the best predictor for overall system performance under different systems, workload sizes and token skewnesses. Our goal is to identify when to use **Distribution-Only Prediction**, which reduces FFN computation load imbalance without communication savings, and when to use **Token-to-Expert Prediction**, which additionally reduces communication latency at the cost of higher predictor complexity. For Token-to-Expert Prediction, we further seek the optimal prediction accuracy to minimize runtime. We use an extended version of LLMCompass [27], a block-level simulator for large language model inference, validated with silicon measurements.

Figure 2 prefill latency for a single layer Mixtral 8×7B, with four A100s on NVLink 3.0 [18]. Batch size is 1, sequence length is 512, matching the experimental setup in Section 3. Figures 2a shows baseline latencies with no prediction. Figure 2b shows different prediction strategies at varying skewness and accuracies.Token-to-Expert Prediction uses a fitted curve to model overhead by accuracy, with interpolation for unmeasured skewness.



(a) Baseline latency with no prediction (interconnect = NVLink).

(b) Latency of different prediction strategies and accuracies (interconnect = NVLink).

Figure 2: Simulated prefill latency for a single layer of Mixtral 8×7B under different prediction strategies and interconnect types. Latency is broken down by component (attention, FFN, communication, overhead) and evaluated across skewness levels on 4 A100 GPUs using NVLink (top) and PCIe (bottom). (a, c) show baseline latencies without prediction; (b, d) show improvements from prediction strategies at varying accuracies. Distribution-Only Prediction reduces FFN compute without overhead, while Token-to-Expert Prediction introduces overhead that trades off with improved load balancing. For each skewness, the best preditor has the minimum total latency across strategies and accuracies.

For each skewness, the first bar is no prediction, the second is *Distribution-Only Prediction* - reducing only FFN compute time with no overhead since distribution is estimated offline. As skewness increases, distribution estimation becomes less accurate.Remaining curves are *Token-to-Expert Prediction* at several accuracies, showing a U-shape: higher accuracy improves load balancing but adds overhead. The best configuration is the one with the lowest latency, usually at moderate accuracy.

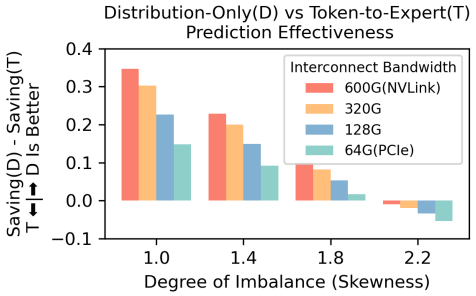


Figure 3: Simulated effectiveness of two strategies' best savings for Mixtral 8X7B on 4 fully-connected A100 with different system interconnect settings.

Generally, Distribution-Only Prediction outperforms Token-to-Expert. For skewness = 1.4 (MMLU-like), it achieves 23% speedup than Token-to-Expert's optimal setting. Token-to-Expert incurs more overhead, especially with low skewnesss where it requires more complex models. When skew is high or communication cost is a bottleneck, Token-to-Expert saves more.

FFN latency reductions from prediction are largely skewness-independent, since mispredictions are measured relative to a perfectly balanced reference scenario. Only overhead varies with skew as higher skewness makes prediction easier.

Figure 3 visualizes savings over baseline (no prediction) for each strategy and system. Difference in savings for the two strategies is calculated by $Distribution\text{-}Only\ Prediction\ saving - Token\text{-}to\text{-}Expert\ Prediction\ saving$. Bars above zero mark Distribution-Only as better; below zero marks Token-to-Expert as better. We also show impacts of different interconnects.

Key takeaways: *Distribution-Only Prediction* excels at low skewness or when communication isn't a bottleneck with its low complexity and zero overhead. *Token-to-Expert Prediction* is only preferable under big skewness and low-bandwidth interconnects where communication savings dominate.

Acknowledgements

This work was partially supported by NSF 2112562 and ARO W911NF-23-2-0224.

References

[1] Peizhuang Cong, Aomufei Yuan, Shimao Chen, Yuxuan Tian, Bowen Ye, and Tong Yang. Prediction is all moe needs: Expert load distribution goes from fluctuating to stabilizing. *arXiv preprint arXiv:2404.16914*, 2024.

[2] Zhixu Du, Shiyu Li, Yuhao Wu, Xiangyu Jiang, Jingwei Sun, Qilin Zheng, Yongkai Wu, Ang Li, Hai Li, and Yiran Chen. Sida: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models. *Proceedings of Machine Learning and Systems*, 6:224–238, 2024.

[3] Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

[4] 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.

[5] Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fastmoe: A fast mixture-of-expert training system. *arXiv preprint arXiv:2103.13262*, 2021.

[6] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 120–134, 2022.

[7] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Haiyang Huang, Newsha Ardalani, Anna Sun, Liu Ke, Hsien-Hsin S Lee, Anjali Sridhar, Shruti Bhosale, Carole-Jean Wu, and Benjamin Lee. Towards moe deployment: Mitigating inefficiencies in mixture-of-expert (moe) inference. *arXiv preprint arXiv:2303.06182*, 2023.

[10] 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.

[11] 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, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[12] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[13] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] 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. *arXiv preprint arXiv:2006.16668*, 2020.

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

[16] Xiaonan Nie, Xupeng Miao, Zilong Wang, Zichao Yang, Jilong Xue, Lingxiao Ma, Gang Cao, and Bin Cui. Flexmoe: Scaling large-scale sparse pre-trained model training via dynamic device placement. *Proceedings of the ACM on Management of Data*, 1(1):1–19, 2023.

[17] NVIDIA. Mixture of experts package. `https://docs.nvidia.com/megatron-core/developer-guide/latest/api-guide/moe.html`, 2025.

[18] NVIDIA. Nvlink and nvlink switch. https://www.nvidia.com/en-us/data-center/nvlink/, 2025.

[19] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

[20] 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 *International conference on machine learning*, pages 18332–18346. PMLR, 2022.

[21] 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. In *International Conference on Learning Representations*, 2017.

[22] Liang Shen, Zhihua Wu, WeiBao Gong, Hongxiang Hao, Yangfan Bai, HuaChao Wu, Xinxuan Wu, Jiang Bian, Haoyi Xiong, Dianhai Yu, et al. Se-moe: A scalable and efficient mixture-of-experts distributed training and inference system. *arXiv preprint arXiv:2205.10034*, 2022.

[23] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[24] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.

[25] Wei Wang, Zhiquan Lai, Shengwei Li, Weijie Liu, Keshi Ge, Yujie Liu, Ao Shen, and Dongsheng Li. Prophet: Fine-grained load balancing for parallel training of large-scale moe models. In *2023 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 82–94. IEEE, 2023.

[26] Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. Edgemoe: Fast on-device inference of moe-based large language models. *arXiv preprint arXiv:2308.14352*, 2023.

[27] Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. Llmcompass: Enabling efficient hardware design for large language model inference. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1080–1096. IEEE, 2024.

[28] Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. Llama-moe: Building mixture-of-experts from llama with continual pre-training. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15913–15923, 2024.

A MLE for Multinomial Distribution

We seek to model the expert activation pattern in a Mixture-of-Experts (MoE) model using a probabilistic approach. Specifically, we are interested in estimating the distribution over expert selections for each MoE layer, based on observed activation frequencies in the training data. To this end, we adopt a multinomial modeling framework and employ Maximum Likelihood Estimation (MLE) to infer the activation probabilities.

Assumptions. We assume that each token independently selects an expert from a fixed pool of K experts in a given layer. Let $E = \{e_1, \ldots, e_K\}$ denote the set of experts in a particular MoE layer. For each token routed through the layer, the expert selection is modeled as an i.i.d. draw from a multinomial distribution with parameters $\mathbf{p} = (p_1, \ldots, p_K)$, where p_i is the probability that expert e_i is selected. Naturally, $\sum_{i=1}^{K} p_i = 1$ and $p_i \geq 0$ for all i.

Maximum Likelihood Estimation. Given N tokens routed through the layer, let n_i denote the number of tokens that selected expert e_i, so that $\sum_{i=1}^{K} n_i = N$. The likelihood of the observed expert assignments under the multinomial distribution is:

$$\mathcal{L}(\mathbf{p}) = \Pr(n_1, \ldots, n_K \mid \mathbf{p}) = \frac{N!}{n_1! \cdots n_K!} \prod_{i=1}^{K} p_i^{n_i}. \tag{1}$$

To estimate $\mathbf{p}$ via MLE, we maximize the log-likelihood:

$$\log \mathcal{L}(\mathbf{p}) = \log \left(\frac{N!}{n_1! \cdots n_K!} \right) + \sum_{i=1}^{K} n_i \log p_i. \tag{2}$$

Ignoring the constant term that does not depend on $\mathbf{p}$, the optimization problem reduces to:

$$\max_{\mathbf{p}} \quad \sum_{i=1}^{K} n_i \log p_i \tag{3}$$

$$\text{s.t.} \quad \sum_{i=1}^{K} p_i = 1, \quad p_i \geq 0. \tag{4}$$

This is a standard constrained optimization problem, and the solution is obtained via the method of Lagrange multipliers. The resulting MLE estimator for each expert's activation probability is:

$$\hat{p}_i = \frac{n_i}{N}, \quad \forall i \in \{1, \ldots, K\}. \tag{5}$$

B Predictor Architectures

e formulate the expert selection problem in Mixture-of-Experts (MoE) as a multi-class classification task, where the objective is to predict the activated expert for each token in a sequence. Let $\mathcal{T}$ denote the set of input tokens, and let $E = \{e_1, \ldots, e_K\}$ denote the set of experts available in a given MoE layer. For each token $t \in \mathcal{T}$, the goal is to predict an expert label $y_t \in \{1, \ldots, K\}$ that will be used by the MoE routing mechanism.

We explore three modeling paradigms for this task: a global frequency-based model, a conditional frequency model, and neural network-based predictors.

Probability-Based Model. This baseline treats all tokens identically and assigns each token to the expert that is most frequently activated in the training data. Let n_i be the number of times expert e_i was selected across all tokens in the training corpus. The model estimates the global activation probabilities using maximum likelihood as:

$$\hat{p}_i = \frac{n_i}{\sum_{j=1}^{K} n_j}, \quad \forall i \in \{1, \ldots, K\}. \tag{6}$$

The predicted expert for any token is then:

$$\hat{y}_t = \arg\max_i \hat{p}_i. \tag{7}$$

This approach ignores token-specific context, providing a static prediction rule that reflects global expert utilization frequencies.

Conditional Probability Model. To improve over the static assignment, we consider a token- or position-conditioned frequency model. Let I_t be the token index or its absolute position in the sequence. For each token index i, we count how many times each expert e_k was selected and compute:

$$\hat{p}_{k|i} = \frac{n_{k,i}}{\sum_{j=1}^{K} n_{j,i}}, \tag{8}$$

where $n_{k,i}$ denotes the number of times token index i selected expert e_k. The model then predicts:

$$\hat{y}_t = \arg\max_k \hat{p}_{k|I_t}. \tag{9}$$

This conditional model captures per-token or per-position biases in expert activation.

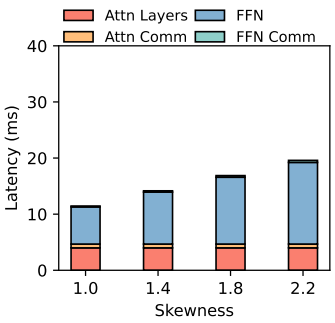
Neural Networks. To learn token-aware expert selection strategies, we train neural models that take token embeddings as input and predict the corresponding expert activation for each MoE layer. Each model is trained with cross-entropy loss and optimized using the Adam optimizer. Input sequences are padded to a fixed length of 512 tokens during training, and separate classifiers are maintained for each layer of the MoE model.

We experiment with the following two architectures:

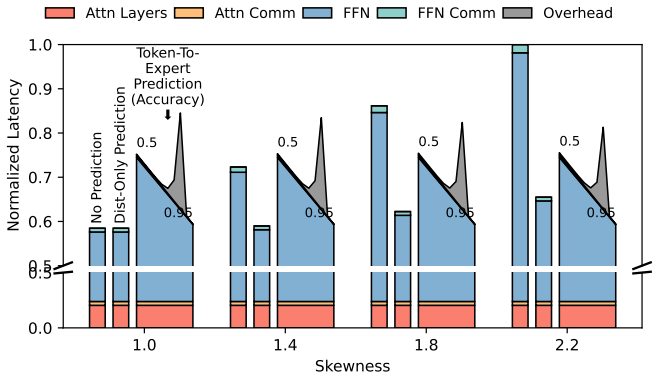
- **Feed-Forward Network (FFN).** The FFN model is a lightweight two-layer MLP. Each input token embedding (of dimension 4096 for Mixtral) is first passed through a linear projection to a 128-dimensional hidden space, followed by a ReLU activation. This is then followed by another linear layer of the same hidden size. Finally, for each target MoE layer, a separate classifier head is implemented as a linear layer mapping from the 64-dimensional hidden state to 8 expert logits. The FFN model is shared across tokens and layers, with layer-specific output heads.
- **LSTM with Sparse Attention.** To capture temporal dependencies, we also design a recurrent model based on an LSTM encoder augmented with sparse attention. The input token embeddings are first projected from dimension 4096 (Mixtral) to 128 using a linear compression layer, followed by a ReLU activation. These projected embeddings are passed through a 2-layer LSTM with hidden size 64, applied in a batch-first manner. To enhance contextual modeling, we incorporate a sparse attention mechanism over the LSTM outputs (i.e., attention is applied using the LSTM outputs as query, key, and value). A residual connection is then added between the attention output and a separate feedforward transformation of the compressed input. Finally, for each MoE layer, a dedicated linear classifier maps the resulting vector to expert logits (8 classes for Mixtral).

C Results on LlaMA-MoE and Switch Transformer

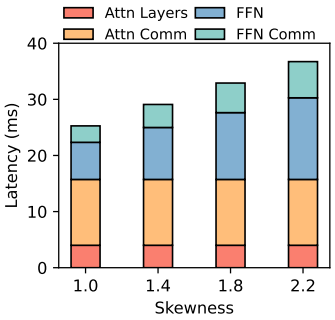
To generalize our claim, we evaluated the performance implications and ran our simulations on other model architectures beyond Mixtral. We show results obtained from the Llama-MoE model [28] in Figure 4 and the Switch Transformer model [4] in Figure C. We used the same datasets and the same hardware configurations as for the Mixtral experiments (MMLU, Alpaca Eval, and SST2; 4 A100 GPUs connected by NVLink or PCIe).



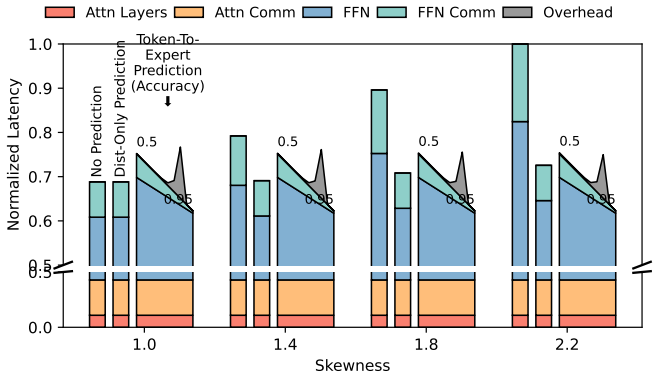
(a) Baseline latency with no prediction (interconnect = NVLink).



(b) Latency of different prediction strategies and accuracies (interconnect = NVLink).



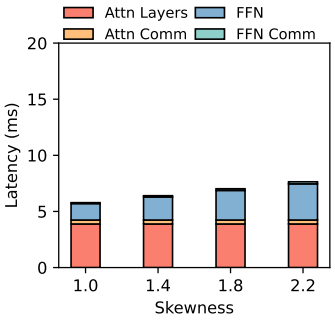
(c) Baseline latency with no prediction (interconnect = PCIe).



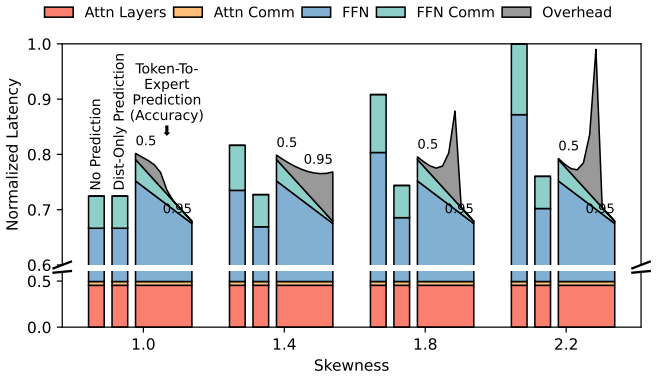
(d) Latency of different prediction strategies and accuracies (interconnect = PCIe).

Figure 4: Simulated prefill latency for a single layer of Llama-MoE model [28] under different prediction strategies and interconnect types. Workload sizes and hardware configurations are the same as Figure 2. For illustration purposes, overhead > 0.5 of original latency is omitted.

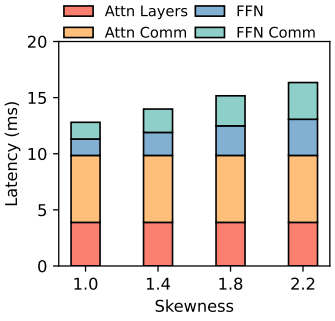
Overall, the trends and the insights are similar to those we derived from the Mixtral model. We observed that the datasets generally have higher skewness in both models compared to Mixtral due to different routing decisions. We also noticed that it is more difficult to obtain very high prediction accuracy, and the prediction complexity required when approaching perfect prediction grows exponentially. For illustration purposes, we have omitted results where the overhead latency is greater than half of the original latency (layer-wise latency without overhead).



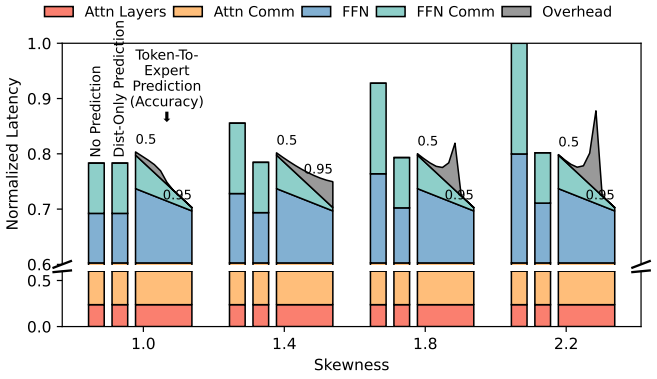
(a) Baseline latency with no prediction (interconnect = NVLink).



(b) Latency of different prediction strategies and accuracies (interconnect = NVLink).



(c) Baseline latency with no prediction (interconnect = PCIe).



(d) Latency of different prediction strategies and accuracies (interconnect = PCIe).

Figure 5: Simulated prefill latency for a single layer of Switch Transformer model [4] under different prediction strategies and interconnect types. Workload sizes and hardware configurations are the same as Figure 2. For illustration purposes, overhead > 0.5 of original latency is omitted.