The Cost of Expertise: Understanding MoE Decode Performance

Sami Abuzakuk
EPFL
Lausanne, Switzerland
sami.abuzakuk@epfl.ch

Oana Balmau
McGill University
Montreal, Canada
oana.balmau@mcgill.ca

Jiaxuan Chen
McGill University
Montreal, Canada
jiaxuan.chen2@mail.mcgill.ca

Anne-Marie Kermarrec
EPFL
Lausanne, Switzerland
anne-marie.kermarrec@epfl.ch

Rafael Pires
EPFL
Lausanne, Switzerland
rafael.pires@epfl.ch

Ramya Prabhu*
EPFL
Lausanne, Switzerland
ramya.prabhu@epfl.ch

Martijn de Vos
EPFL
Lausanne, Switzerland
martijn.devos@epfl.ch

Abstract

Mixture-of-experts (MoE) architectures have emerged as a popular strategy for scaling large language models (LLMs), enabling substantial increases in parameter counts without proportional growth in per-token FLOPs. Existing analyses often model MoE inference as an attention backbone followed by a routing layer that is used to route tokens to multiple Feed-Forward Network (FFN). MoE-based LLMs expand the design space across sparsity and scaling dimensions, such as top-k routing, expert intermediate size and prompt-dependent expert activation whose systems implications remain underexplored especially for the decode-phase, despite decode dominating interactive inference workloads. We systematically study how MoE design choices influence inference latency during the decode phase, which is the primary latency bottleneck in LLM workloads. We analyze latency behavior across varying batch size, expert intermediate sizes, and numbers of activated experts. Our results reveal that expert intermediate size, batching behavior, and expert activation patterns interact in non-trivial ways: latency scales near-linearly with expert width, batching gains plateau for wider experts due to early GPU saturation, and activating additional experts increases latency through memory pressure rather than compute overhead. These findings expose critical system-level trade-offs in MoE design that are invisible to FLOP-centric analyses.

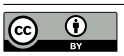
CCS Concepts

• **Computing methodologies** → **Artificial intelligence**.

Keywords

GPUs, Systems for ML , Mixture of Experts (MoE) inference, Large Language Models (LLMs)

*Corresponding author.



EuroMLSys '26, Edinburgh, Scotland Uk

ACM ISBN 979-8-4007-2605-7/2026/04
https://doi.org/10.1145/3805621.3807659

ACM Reference Format:
Sami Abuzakuk, Oana Balmau, Jiaxuan Chen, Anne-Marie Kermarrec, Rafael Pires, Ramya Prabhu*, and Martijn de Vos. 2026. The Cost of Expertise: Understanding MoE Decode Performance. In *Sixth European Workshop on Machine Learning and Systems (EuroMLSys '26), April 27–30, 2026, Edinburgh, Scotland UK*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3805621.3807659

1 Introduction

Transformer-based architectures have become a dominant choice for sequence modeling tasks due to their ability to efficiently capture long-range dependencies using self-attention [3, 25, 29]. Today, transformer-based architectures form the backbone of highly capable foundation models such as GPT-5 [27], Gemini [8], and Claude [1], which power a wide range of modern AI applications. As these models are scaled to improve performance, their parameter counts and associated resource requirements increase substantially. This presents challenges for inference, particularly in large-scale deployments that are constrained by hardware availability.

Mixture-of-experts (MoE) [6, 16] is a promising approach to address these inference challenges. MoE increases model capacity without proportionally increasing computational inference costs and are increasingly integrated in the architecture of machine learning (ML) models. By sparsely activating subsets of experts, MoE models reduce per-token compute cost, but introduce input-dependent execution behavior due to token routing and additional communication requirements between GPUs [9, 14]. As a result, the iteration time of MoE workloads exhibits significant variability, complicating performance modeling and system-level resource management [13].

While MoE architectures reduce per-token compute cost, they simultaneously introduce an additional layer of systems complexity. Unlike dense transformers, whose execution characteristics are largely static and predictable, the performance of MoE inference largely depends on routing decisions that determine which experts are activated for each token. Performance therefore not only depends on the batch size and model parameter count, but also on the size of each expert, the employed expert parallelism strategy, and the number of activated experts per token. These parameters

interact in non-trivial ways, influencing computation time, memory overhead, kernel efficiency, and load balance between GPUs.

With the growing adoption of MoE architectures, substantial effort has been devoted to optimizing MoE inference. Prior systems work has primarily focused on mitigating performance inefficiencies arising from sparse expert execution and runtime imbalance. For example, MegaBlocks [7] reduces kernel launch overheads by introducing block-sparse fused kernels that enable efficient parallel execution of expert computations on GPUs. HarMoEny [5] improves runtime efficiency by addressing expert load imbalance through dynamic token redistribution and asynchronous expert prefetching. DeepSpeed-MoE [26] explores expert placement, sharding strategies, and communication optimizations to improve distributed MoE execution. These approaches typically model MoE layers as attention followed by multiple FFN and target end-to-end latency by optimizing imbalance-driven bottlenecks such as skewed routing and uneven expert utilization.

However, our results suggest that even in the absence of routing imbalance, MoE inference exhibits inherent performance variability stemming from MoE-specific parameters. Architectural factors including expert intermediate size, routing sparsity, and expert activation diversity introduce dynamism that affects latency through hardware utilization, memory traffic, and saturation behavior. Furthermore, contemporary MoE architectures increasingly explore alternative scaling strategies, such as varying expert width and expert count. These design choices introduce new performance regimes whose interactions may alter or invalidate assumptions underlying prior optimizations. Consequently, understanding and optimizing MoE inference latency requires systematic characterization of fundamental parameters beyond imbalance-centric analysis.

In this work, we perform a controlled microbenchmark study to disentangle the effects of expert size, routing configuration, and batching on decode-phase MoE inference latency. We focus specifically on the decode phase because it dominates latency in interactive LLM deployments and operates under small, sequential batch sizes where hardware efficiency is most fragile [19, 28].

While prior work attributes inference performance to general GPU bottlenecks such as memory bandwidth and kernel efficiency, these explanations do not capture how the diversity of MoE architectural design choices shapes these bottlenecks in practice. Modern MoE models vary widely in expert intermediate size, number of experts, and routing configurations, yet the performance implications of these design dimensions remain underexplored. In particular, these architectural parameters directly influence memory traffic, cache locality, and hardware saturation behavior, leading to qualitatively different performance regimes across models. As a result, simply characterizing MoE inference as "memory-bound" is insufficient to predict performance across configurations.

To address this gap, we isolate expert MLP latency — defined as the execution time of the expert feed-forward computation excluding attention and routing overheads — and systematically study how it varies across configurations. We intentionally isolate the expert MLP layer to disentangle its contribution from other components such as attention, KV cache management, and routing imbalance. While end-to-end inference includes these interactions, isolating the expert computation allows us to attribute performance trends directly to architectural parameters such as expert width and activation patterns, which are otherwise difficult to study in full-system settings. Our results show that while expert MLP latency scales predictably with intermediate size, batching efficiency is highly configuration-dependent and may diminish or plateau for wider experts.

Our goal is not to show that MoE inference is memory-bound, but to identify which architectural parameters determine how and when these bottlenecks arise.

These findings demonstrate that optimizing MoE inference cannot rely on simple scaling rules derived from dense models. Instead, performance is governed by architectural parameters that directly modulate hardware behavior, requiring careful consideration of expert width, activation patterns, and batching dynamics. Overall, our study provides a step toward a more systematic understanding of decode-phase MoE inference, exposing configuration-dependent trade-offs that are not captured by existing imbalance- or compute-centric analyses. In summary, we make the following contributions:

- We conduct a targeted microbenchmark study that isolates decode-phase expert MLP latency in MoE LLMs across varying batch sizes, expert intermediate sizes, and routing configurations (Section 4).
- We show that expert MLP latency scales proportionally with expert intermediate size and that batching efficiency is highly sensitive to expert width (Section 4.1.1, Section 4.1.2).
- We analyze the impact of expert activation patterns on latency and discuss the implications of these findings for future MoE system design (Section 4.2).

2 Background and Preliminaries

We first provide a brief overview of transformer models and MoE layers, which together form the backbone of modern large language models (LLMs). We focus on aspects of these architectures that directly influence inference-time performance, including attention computation, feed-forward execution, and sparse expert activation.

2.1 Transformer Models

Transformer architectures have become the dominant backbone for a broad range of ML applications [29]. A standard transformer consists of a stack of identical layers, or transformer blocks, each refining token representations through attention and feed-forward computation. This architecture is visualized in Figure 1 (left).

A token denotes the intermediate representation of an input element, such as a word [2], subword unit [17], or character [11]. Tokens are encoded as vectors and processed jointly as a sequence. Each transformer block contains two primary components. The self-attention mechanism models dependencies across the sequence by computing interactions between tokens, producing contextualized representations. These representations are then passed to a Feed-Forward Network (FFN), typically composed of two linear layers separated by a nonlinear activation, which transforms features independently for each token.

These components exhibit distinct computational properties. Self-attention introduces sequence-length-dependent costs and significant memory access, while the FFN contributes substantial compute through dense matrix multiplications. As models scale, FFN computation often dominates FLOPs, motivating alternatives such

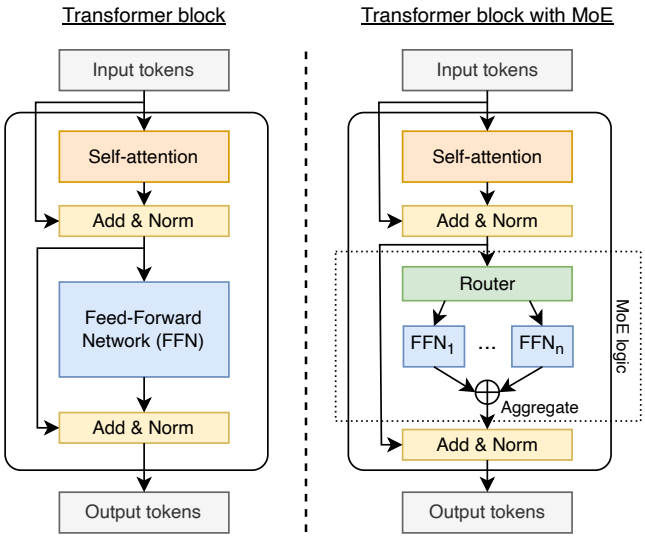


Figure 1: A transformer block (left) and a MoE block, containing the MoE logic (right). MoE-based models replace the FFN with a router and multiple experts implemented as FFNs.

as MoE layers. Understanding this division is important for inference analysis, as performance bottlenecks depend on sequence length, batch size, and architectural configuration.

2.2 Mixture-of-Expert Models

MoE models are a class of transformer architectures designed to improve computational efficiency by activating only a subset of model parameters during inference. Instead of applying fully dense feedforward layers to all tokens, MoE architectures replace these layers with multiple expert networks and a routing mechanism that assigns each token to a small number of experts. This sparse activation reduces computation while maintaining high model capacity. For a given layer, only the selected experts are executed, reducing computation compared to dense models with equivalent parameter counts. In practice, MoE layers are commonly configured with top-k routing, expert capacity limits, and load balancing objectives to mitigate expert overload. State-of-the-art MoE models, such as Mixtral [16], OpenAI GPT-4 [20], GPT-OSS [21] and DeepSeek v3 [4], exemplify this architecture.

From a systems perspective, MoE execution differs from dense layers in two key ways. First, expert activation is input-dependent, causing the amount of computation per iteration to vary across inputs. Second, expert execution requires communication to dispatch tokens to experts and to combine expert outputs, typically implemented using all-to-all collectives. These properties introduce variability in execution time that is absent in dense models.

Although MoE layers are commonly described as sparse alternatives to dense FFN, contemporary MoE-based LLMs span a rich design space. Expert width, expert count, routing sparsity (top-k), and expert-sharing mechanisms vary widely across models. These parameters alter not only theoretical FLOPs but also hardware utilization, memory pressure, routing overhead, and latency scaling.

2.3 Motivation

Existing performance intuitions derived from dense transformers do not necessarily extend to MoE inference. In particular, decode-phase execution introduces distinct bottlenecks where memory traffic and expert activation dynamics dominate. Understanding the latency implications of these design variations therefore requires targeted empirical analysis, motivating the study presented in this work.

3 Evaluation Setup

All experiments are conducted on a system equipped with 2× NVIDIA A100 GPUs with 80 GBs of DRAM We use PyTorch 2.10 and vLLM 0.12.0 as the inference framework. Our evaluation focuses on the decode phase, where latency directly impacts interactive inference performance. We isolate the cost of the MoE expert layer by timing vLLM's FusedMoE function. This measurement captures the latency of fused expert MLP execution, including routing-induced token permutation, expert computation, and output aggregation overheads. We benchmark a single decode step while sweeping the decode batch size over 8,16,32,64,128. We fix the hidden size to 2048 to perform our experiments. For each configuration, we vary MoE design parameters expert intermediate size and number of experts activated at runtime. Unless otherwise stated, we distribute tokens uniformly across active experts to remove load imbalance effects and isolate architectural influences on latency. Each experiment is timed using CUDA Events. We include warm-up iterations followed by repeated timed runs, and we report averaged latency values in milliseconds.

4 Baseline Expert MLP Latency Scaling

We characterize decode-phase latency of the MoE expert MLP using targeted microbenchmarks that isolate its execution while varying expert intermediate size, batch size, and expert activation patterns. We address three questions: (i) how expert size impacts latency, (ii) how batching efficiency depends on expert width, and (iii) how the number of activated experts affects runtime.

4.1 Effect of Expert Size Across Batch Sizes

4.1.1 Latency Scaling with Expert Intermediate Size. We examine how expert intermediate size affects expert MLP latency. Figure 2 shows that latency scales approximately linearly with the intermediate dimension across batch sizes. For example, at batch size 128, latency increases from 0.77 ms (size 512) to 6.35 ms (size 6144).

At small batch sizes, we observe a brief plateau (e.g., 512 to 1024 at batch size 8), indicating that fixed launch and memory overheads dominate before the kernel reaches sufficient arithmetic intensity.

Overall, expert intermediate size is a primary determinant of decode-phase latency: increasing expert width introduces predictable compute overhead while pushing the GPU toward saturation earlier.

Key Takeaway. Expert MLP latency scales approximately linearly with expert intermediate size.

4.1.2 Batching Efficiency Across Expert Intermediate Sizes. We next analyze how batching efficiency varies with expert width. Figure 3 compares latency across batch sizes for expert intermediate sizes of 512 and 6144.

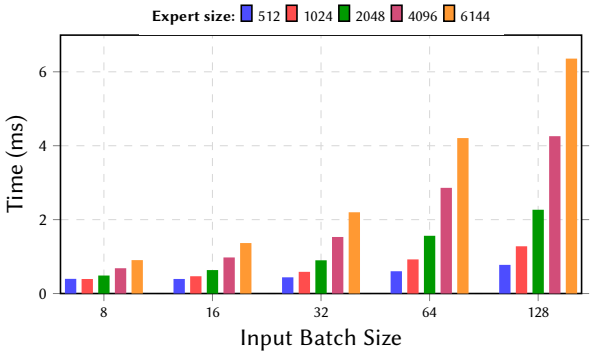


Figure 2: Expert MLP latency vs. expert intermediate size across decode batch sizes.

For smaller experts, increasing batch size reduces latency per token through improved utilization and amortization of fixed overheads. In contrast, larger experts exhibit diminishing batching benefits: latency improvements plateau early as the GPU approaches saturation.

This behavior arises because wider experts increase both compute and memory demands, exhausting hardware resources at smaller batch sizes. As a result, batching primarily improves throughput rather than reducing absolute latency for large experts.

These results show that batching efficiency depends strongly on expert width, and assumptions of consistent batching gains do not hold for large experts.

Key Takeaway. Batching benefits diminish for wider experts, which reach hardware saturation at smaller batch sizes.

4.2 Effect of Activated Experts

We analyze how the number of activated experts affects decode-phase latency by varying the number of active experts while keeping hidden size fixed and distributing tokens uniformly.

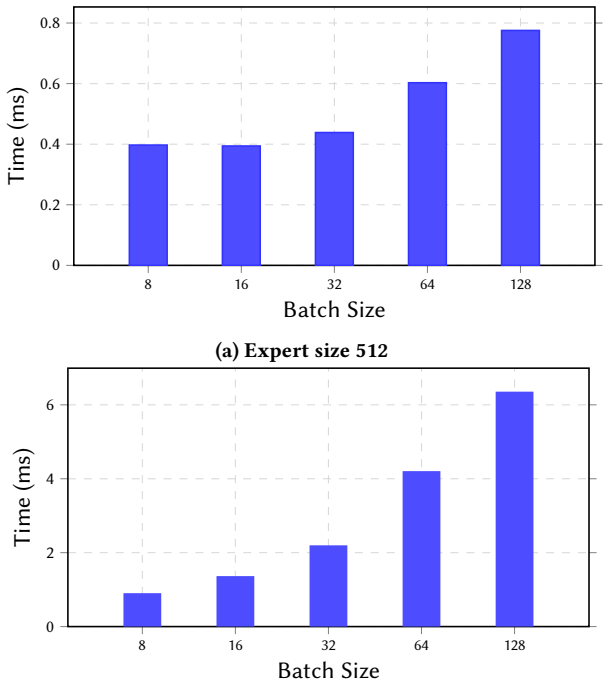
Figure 4 shows that increasing the number of activated experts consistently increases expert MLP latency. This effect is driven primarily by memory behavior rather than compute: activating more experts increases the volume of parameters accessed per step, reducing cache locality and increasing weight loading overhead.

These results suggest that decode-phase performance is often memory-bound under realistic configurations. To mitigate this, systems can improve locality by grouping tokens with similar routing patterns and reducing expert churn across decode steps.

Key Takeaway. Activating more experts increases latency due to memory pressure, even under uniform load.

5 Discussion

Our results show that expert MLP latency in MoE models is influenced by factors beyond those typically considered in dense architectures. While prior work largely attributes MoE performance variability to expert token routing imbalance, our measurements indicate that latency is also strongly governed by expert intermediate size, and the number of activated experts. These factors introduce



(a) Expert size 512

(b) Expert size 6144

Figure 3: Expert MLP latency across batch sizes for different expert widths.

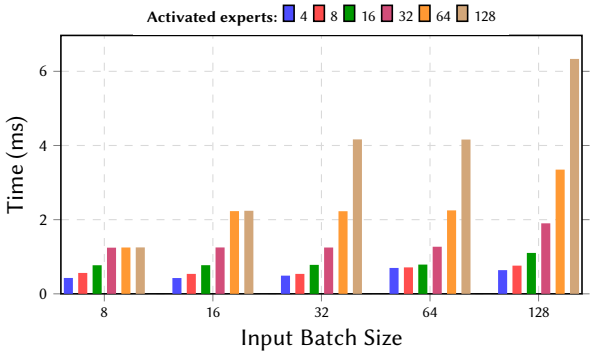


Figure 4: Expert MLP latency vs. number of activated experts.

performance effects that are not captured by imbalance-centric analyses. Although MoE architectures are often justified by constant theoretical FLOPs via sparse activation, the latency trends observed in fig. 2 demonstrate that hardware efficiency and saturation effects play a dominant role in determining the runtime behaviour of this architecture in practice. The contrast between small and large expert intermediate sizes reveals distinct execution regimes. Narrower experts benefit more consistently from batching, with modest latency growth as batch size increases. This suggests that batching effectively amortizes kernel overheads and improves GPU utilization when expert compute remains relatively lightweight.

Table 1: Architectural characteristics of representative MoE models.

Model	Experts	Top-k	Hidden	Expert Int. Size
Mixtral 8×7B[16]	8	2	4096	14336
Qwen1.5-MoE[24]	128	4	2048	1408
Qwen3-MoE[30]	128	8	2048	768

In contrast, wider experts exhibit sharply increasing latency, indicating that larger intermediate dimensions rapidly drive the GPU toward saturation. Beyond this point, batching yields diminishing returns and latency scaling becomes governed by compute throughput and memory bandwidth constraints. These findings highlight that expert width shapes not only raw MLP cost but also batching efficiency. Scaling expert capacity through larger intermediate sizes therefore introduces non-linear latency trade-offs, particularly during decode where batch sizes are inherently limited.

Implications for Inference Configuration. For latency-sensitive deployments, expert intermediate size should be treated as a systems-critical parameter. Excessively wide experts may degrade utilization efficiency and limit batching gains, while moderate widths may provide more favorable latency–throughput trade-offs.

Our findings can be interpreted in the context of evolving MoE design strategies. Table 1 lists the popular models today that have an MoE architecture. Mixtral [16] exemplifies a configuration with relatively wide experts and a small number of experts per layer. In such designs, expert MLP computation is sufficiently large that even modest decode batch sizes can drive high GPU utilization. While this can yield strong compute efficiency, it also causes earlier hardware saturation, reducing the marginal benefits of batching and increasing latency sensitivity to expert activation.

In contrast, more recent MoE architectures like Qwen3 [30] and Qwen1.5 [24] increasingly scale model capacity by expanding the number of experts while employing narrower expert intermediate sizes. This approach reduces per-expert compute intensity, potentially delaying saturation and preserving batching efficiency. However, increasing expert count introduces new systems challenges, including higher routing overhead, greater expert activation diversity, and amplified memory traffic due to more irregular expert weight accesses. Consequently, optimal MoE inference performance depends on jointly balancing expert width and expert count in accordance with hardware characteristics and workload properties.

Limitations. Our study relies on targeted microbenchmarks that isolate expert MLP behavior under controlled routing assumptions. End-to-end serving systems may introduce additional overheads, including scheduling effects and cross-layer interactions. Our evaluation does not model routing imbalance or cross-layer interactions such as attention and KV cache effects, which can influence end-to-end latency. In particular, scheduling effects and inter-layer dependencies may amplify or attenuate the trends observed in isolation. However, our goal is to characterize the intrinsic behavior of expert computation, which remains a dominant component of decode-phase latency in MoE models.

6 System-Level Implications and Future Directions

While the previous discussion focused on inference configuration trade-offs at the expert-layer level, we now consider implications for multi-GPU execution, scheduling, and system architecture. Beyond single-layer latency effects, our findings have broader implications for distributed MoE inference and emerging LLM serving architectures.

The latency sensitivities observed in this work have important implications for distributed MoE inference and multi-tenant deployments. In multi-GPU settings, expert execution relies on token dispatch and aggregation, typically implemented via all-to-all communication. Architectural choices that increase expert MLP latency — such as wider experts or higher expert activation — may therefore amplify communication overhead and synchronization delays. Decode-phase inference is particularly sensitive to latency variability. Fluctuations in expert activation and batching efficiency can introduce unpredictable per-step latency, which may propagate across devices and degrade tail performance. These effects complicate scheduling, performance modeling, and service-level objective (SLO) enforcement.

The insights from this study are especially relevant in the context of disaggregated serving architectures, which decouple prefill and decode execution across specialized resources. Systems such as Splitwise [22], TetriInfer [12] and DistServe [33] separate compute-intensive and latency-sensitive stages to improve utilization and responsiveness. However, decode-phase MoE variability, driven by expert activation patterns, memory traffic, and hardware saturation, may introduce additional scheduling complexity in such designs. Understanding how expert-layer latency interacts with disaggregated pipelines remains an important direction for future systems research.

In multi-tenant environments, MoE inference may generate irregular memory traffic and expert weight movement, producing interference patterns that differ from dense-model workloads. Incorporating MoE-aware metrics into scheduling and admission control policies may therefore improve latency stability and resource utilization. Collectively, these observations motivate the design of systems explicitly optimized for sparse expert execution, including routing-aware batching, expert locality optimization, and adaptive scheduling strategies.

7 Related Work

7.1 Systems Optimizations for MoE

MoE Inference is well studied in literature from the systems perspective. MegaBlocks formulates MoE computation in terms of block-sparse operations to avoid dropping tokens without increasing the computation cost during MoE training [7]. Tutel [14] and DeepSpeed-MoE [26] improve MoE model performance by combining expert parallelism, model parallelism, and tensor parallelism to significantly boost throughput and reduce latency. These systems also propose specialized communication primitives. Lina [18] improves MoE training and inference by prioritizing all-to-all communication and opportunistic resource scheduling when expert

activation patterns are skewed. Towards MoE-deployment [13] proposed dynamic gating, expert buffering and expert load balancing to help in deploying MoE models for inference. MoETuner[9] determines an expert-to-GPU assignment that reduces inter-GPU token routing overhead and balances token processing across devices, leading to lower tail latency and end-to-end execution time. While all these works look at expert imbalance, none of them analyze the runtime behaviour of MoE models at the granularity of number of experts activated.

7.2 Performance Modeling for LLMs

Recent work has begun to make LLM inference performance more predictable and interpretable by combining hardware-aware bounds with systems-level measurements: LLM-Viewer[31] synthesizes inference bottlenecks via a roofline view (notably separating prefill and decode) and positions common optimizations as shifts in compute–memory regimes; [10] provides an engine-centric account of how KV-cache management and scheduling translate these bottlenecks into realized throughput in production serving; moving from explanation to prediction, [15] uses roofline-derived structure (often with calibration) to estimate runtime, while [23] develops a more device-agnostic analytical model aimed at forecasting inference across heterogeneous hardware; finally, LLMCompass [32] offers a modeling/simulation framework for evaluating inference performance across hardware design points, serving as a complementary tool for prediction and bottleneck attribution beyond single-device roofline bounds. However, these approaches are largely developed and validated on dense transformers and can fall short for MoE inference, where token-dependent expert routing introduces highly nonuniform compute and memory demand, conditional execution changes effective arithmetic intensity on the fly, and dynamic sparsity reduces the fidelity of static roofline bounds; moreover, expert parallelism can amplify all-to-all communication and dispatch overheads, while load imbalance from skewed routing (or constrained top-k selection) can dominate latency and undercut predictors that assume smooth batching and homogeneous layer behavior, suggesting MoE-aware models must explicitly account for routing distributions, expert placement, dispatch/collect costs, and tail-latency effects.

8 Conclusion

In this work, we presented an analysis of the MLP layer of MoE transformer during the decode phase, focusing on how core architectural and routing parameters influence expert MLP performance. Through targeted micro-benchmarks, we characterized the impact of expert intermediate size, batching behavior, top-k routing, and runtime expert activation on inference latency. Our results demonstrate that decode-phase MoE latency is governed by more than theoretical FLOPs. Expert width scales MLP cost predictably but also alters batching efficiency and hardware saturation behavior. Larger experts exhibit diminishing batching benefits, while increased expert activation inflates memory traffic and weight movement overheads. These effects reveal distinct compute- and memory-bound regimes that challenge simplified performance assumptions inherited from dense transformers.

These findings highlight the need for MoE-aware performance modeling and system design. Expert configuration parameters, often chosen for model quality, have direct and sometimes non-linear implications for latency efficiency. Incorporating runtime signals such as expert activation diversity and utilization regimes may enable more robust scheduling, batching, and resource provisioning strategies. Future work should extend this characterization to end-to-end serving stacks, distributed MoE deployments, and multi-tenant environments, where routing variability and communication overheads further shape inference behavior. A principled understanding of decode-phase MoE performance will be essential for building predictable, efficient, and latency-aware LLM systems.

Acknowledgments

This work has been partially funded by the Swiss National Science Foundation, under the project "FRIDAY: Frugal, Privacy-Aware and Practical Decentralized Learning", SNSF proposal No. 10.001.796. We would like to thank the reviewers for their insightful comments

References

[1] Anthropic. Claude (claude 3 opus version), 2024. Large language model.
[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.
[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020.
[4] DeepSeek-AI. Deepseek-v3 technical report, 2025.
[5] Zachary Doucet, Rishi Sharma, Martijn de Vos, Rafael Pires, Anne-Marie Kermarrec, and Oana Balmau. Harmoeny: Efficient multi-gpu inference of moe models, 2025.
[6] 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.
[7] Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts, 2022.
[8] Gemini Team et al. Gemini: a family of highly capable multimodal models, 2023.
[9] Go et al. Moetuner: Optimizing mixture-of-experts inference via expert placement and routing-aware scheduling, 2025. arXiv preprint.
[10] Aleksa Gordic. Inside vllm: Anatomy of a high-throughput llm inference system, September 2025. Accessed: 2026-02-01.
[11] Rohit Gupta, Constantin Orăsan, and Ruslan Mitkov. Character-based neural machine translation with transformers. In *Proceedings of Recent Advances in Natural Language Processing (RANLP)*, 2019.
[12] Qizheng Hu, Ziyun Huang, et al. Inference without interference: Disaggregate llm inference for mixed downstream workloads, 2024.
[13] Huang et al. Towards efficient mixture-of-experts deployment: Dynamic gating and expert load balancing, 2023. arXiv preprint.
[14] Sehoon Hwang, Deepak Narayanan, Jongsoo Kim, et al. Tutel: Adaptive mixture-of-experts at scale. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023.
[15] Saki Imai, Rina Nakazawa, Marcelo Amaral, Sunyanan Choochotkaew, and Tatsuhiro Chiba. Predicting llm inference latency: A roofline-driven ml method. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
[16] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, et al. Mixtral of experts, 2024.
[17] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.
[18] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed moe training and inference with lina, 2024.
[19] NVIDIA. Mastering llm techniques: Inference optimization. https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/, 2023. Accessed: 2026-02-24.
[20] OpenAI. Gpt-4 technical report, 2024.
[21] OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025.
[22] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132. IEEE, 2024.

[23] Rajeev Patwari, Ashish Sirasao, and Devleena Das. Forecasting llm inference performance via hardware-agnostic analytical modeling, 2025.
[24] Qwen Team. Introducing qwen1.5, February 2024.
[25] Alec Radford, Jeffrey Wu, Rewon Child, et al. Language models are unsupervised multitask learners, 2019. OpenAI blog.
[26] Samyam Rajbhandari, Jeffrey Li, Zhewei Yao, et al. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International Conference on Machine Learning (ICML)*, 2022.
[27] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, et al. Openai gpt-5 system card, 2025.
[28] TNG Technology Consulting and Hugging Face. Llm performance: Prefill, decode, and concurrent requests. https://huggingface.co/blog/tngtech/llm-performance-prefill-decode-concurrent-requests, 2024. Accessed: 2026-02-24.
[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008, 2017.
[30] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, et al. Qwen3 technical report, 2025.
[31] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, et al. Llm inference unveiled: Survey and roofline model insights, 2024.
[32] 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 Computer Society, 2024.
[33] Yinmin Zhong, Shengyu Zhang, Siming Zhao, et al. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.