UNCOVERING INTRA-EXPERT ACTIVATION SPARSITY FOR EFFICIENT MIXTURE-OF-EXPERT EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture of Experts (MoE) architecture has become the standard for state-of-the-art large language models, owing to its computational efficiency through sparse expert activation. However, sparsity through finer expert granularity is becoming increasingly difficult to achieve due to fundamental training challenges such as expert collapse and load imbalance. In this work, we explore and leverage **intra-expert activation sparsity** as a complementary and underexplored dimension of sparsity in MoE models. Surprisingly, substantial intra-expert sparsity is readily available in off-the-shelf pre-trained MoE models, providing up to **98% sparsity** on top of existing inter-expert sparsity of the routed experts without significant accuracy loss. We explore intra-expert activation sparsity across six open-source MoE models ranging from 1B to 397B parameters, and extend the MoE execution pipeline of vLLM to leverage intra-expert activation sparsity while preserving the existing MoE layer optimizations, achieving up to 2.0× speedup in MoE layer execution and 1.2× end-to-end speedup compared to the dense vLLM baseline.

1 INTRODUCTION

Mixture of Experts (MoE) architecture (Shazeer et al., 2017; Fedus et al., 2022) has become the de facto standard for state-of-the-art Large Language Models (LLMs) (Qwen Team, 2026; OpenAI, 2025; Meta Llama Team, 2025), owing to its superior computational efficiency over traditional dense LLM architectures. MoE models use a sparse feed-forward network (FFN) composed of multiple independent sub-networks called *experts*. Each expert typically specializes in a distinct domain of knowledge (Zhang et al., 2022), and during inference, the model selectively activates only those experts most relevant to the input token. This sparse activation characteristic of MoE models enables reduced computation while maintaining high model capacity, and MoE models have evolved to maximize sparsity by increasing the number of experts and reducing the percentage of active experts.

However, training these sparse MoE models is notoriously difficult. Without careful regularization, MoE models are prone to expert collapse (Fedus et al., 2022; Zoph et al., 2022), where the model converges to using a few general experts while leaving the majority of experts untrained, or load imbalance (Fedus et al., 2022; Zhou et al., 2022), where hotspots in expert activation degrade both model quality and hardware utilization. Additionally, MoE models suffer from representation collapse where experts converge to similar specializations (Chi et al., 2022) and reduced per-expert gradient updates (Panda et al., 2026). These challenges only compound as the number of experts grows, posing fundamental difficulties for further increasing *inter-expert sparsity* in MoE models.

In this paper, we instead focus on *neuron-level sparsity within each activated expert*, or **intra-expert activation sparsity**, as a complementary source of sparsity in MoE models at inference time. Activation sparsity has been widely studied in model architectures with ReLU activations that explicitly zero neurons with negative activations (Glorot et al., 2011), and recent studies reveal that activation sparsity exists in newer GLU activation-based Shazeer (2020) LLM architectures as well (Lee et al., 2024; Liu et al., 2024). However, despite the dominance of MoE models in modern LLM architectures, only a few existing works explore the use of activation sparsity within MoE models (Cheng et al., 2025; Szatkowski et al., 2025), and none provide a solution for applying activation sparsity to MoE models for efficient inference in high-performance serving systems.

We find **two major obstacles** that inhibit the widespread use of intra-expert activation sparsity for efficient MoE inference. First, many existing activation sparsity works target older dense architectures

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

and therefore lack characterization of MoE models under activation sparsity. Secondly, there is no *end-to-end integration* of intra-expert activation sparsity in highly optimized execution systems. Production systems such as vLLM (Kwon et al., 2023) include sophisticated MoE kernels and specialized techniques such as expert parallelism (DeepSeek-AI, 2024; Lepikhin et al., 2021) to maximize performance. However, existing works only implement a naive research-level baseline, leaving a critical gap in evaluating intra-expert activation sparsity in high-performance systems.

To this end, we first conduct a thorough study of intra-expert activation sparsity on off-the-shelf MoE models ranging from 1B to 397B parameters and report a surprising finding: many existing MoE models already possess a substantial intra-expert activation sparsity. Only a few critical tokens are sensitive to sparsification, and by balancing neurons across critical and non-critical tokens, we measure up to **90% additional sparsity** within active experts, while retaining 95% of the model accuracy. We also find that neurons from shared and routed experts have markedly different impacts on accuracy, to the point that when all shared expert neurons are active, as few as 2% of the active routed expert parameters suffice to maintain model capacity for large models such as Qwen3.5-122B-A10B.

Based on this study, we extend the fused MoE execution pipeline of the popular production-grade inference system vLLM to leverage intra-expert activation sparsity by skipping the up- and down-projection computations for inactive neurons. Our implementation builds on top of the expert-batched MoE kernel design of vLLM, retaining **full support for existing optimizations** such as HIP Graph (AMD, 2024a) and CUDA Graph (NVIDIA, 2019), kernel fusion (AMD, 2024b), and expert parallelism. We also share the insights we gained during our implementation to clearly illustrate the benefits and limitations of intra-expert sparsity in an end-to-end production setting.

In summary, we make the following contributions. (1) We explore intra-expert activation sparsity across six off-the-shelf MoE models, with thorough analysis of their activation sparsity characteristics. (2) We redesign the MoE execution pipeline of vLLM to efficiently exploit both inter- and intra-expert sparsity on GPUs. (3) We integrate our sparse execution pipeline into the full vLLM engine and demonstrate up to 2× speedup in MoE layer execution and 1.2× speedup in end-to-end inference. Together, these contributions establish intra-expert activation sparsity as a practical optimization technique for MoE model execution.

2 Background and Related Work

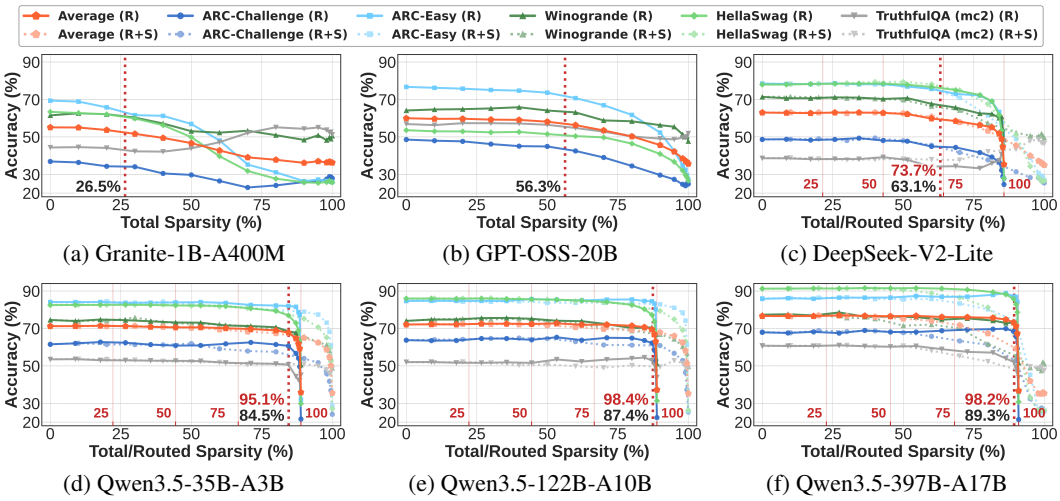
Mixture of Experts: Mixture of Experts (MoE) FFN architecture leverages the specialization of neurons by replacing the single FFN with multiple independent expert FFNs, where each expert learns to specialize on a distinct subset of inputs (Shazeer et al., 2017; Fedus et al., 2022). During inference, a router network determines which experts should be activated for a given input token, and the vast majority of FFN neurons belonging to unrelated experts are efficiently bypassed, providing coarse-grained *inter-expert sparsity*. However, training such MoE models is known to be extremely difficult (Fedus et al., 2022; Zoph et al., 2022; Chi et al., 2022), as the model must simultaneously specialize each expert toward distinct domains while ensuring equal importance of the domains for balanced expert utilization. Since expert specialization must also emerge *implicitly* during training rather than being explicitly prescribed, MoE training is highly unstable without careful regularization, making models prone to expert collapse (Zoph et al., 2022) or load imbalance (Zhou et al., 2022).

Activation Sparsity: Sparsity in neural networks can also be achieved through activation sparsity. Unlike inter-expert sparsity, which is built into the MoE model architecture, activation sparsity exploits the tendency of *activation functions* to suppress a large fraction of neuron outputs. For example, the hard zero threshold of ReLU activation causes a significant fraction of neurons to be completely inactive (Glorot et al., 2011), allowing computations to be skipped entirely (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014). Recent studies confirm the existence of activation sparsity in models with leaky activations such as GEGLU or SwiGLU (Shazeer, 2020), despite their lack of exact zeros (Szatkowski et al., 2025; Cheng et al., 2025; Lee et al., 2024; Liu et al., 2024). Sparsing Law (Luo et al., 2024) establishes that the fraction of activated neurons decreases as the model parameter size increases, regardless of the activation used. This observation is empirically shown to be true in LLMs using leaky activations (Szatkowski et al., 2025), such as Gemma3 (Gemma Team, 2025) and Qwen 2.5 (Yang et al., 2024), and also MoE models such as Qwen3-30B-A3B (Yang et al., 2025) and Mixtral-8x7B (Jiang et al., 2024). These works collectively establish that even FFN neurons have varying importance across tokens and can be leveraged for activation sparsity.

Table 1: Architectural details of the tested MoE models

Model Name	Params (Active / Total)	Experts	Active Experts	FFN Dim.	Shared Experts
Granite-1B-A400M (Granite Team, IBM, 2024)	400M / 1B	32	8	512	False
GPT-OSS-20B (OpenAI, 2025)	3B / 20B	32	4	2880	False
DeepSeek-V2-Lite (DeepSeek-AI, 2024)	2.4B / 16B	64	6	1408	True
Qwen3.5-35B-A3B (Qwen Team, 2026)	3B / 35B	256	8	512	True
Qwen3.5-122B-A10B (Qwen Team, 2026)	10B / 122B	256	8	1024	True
Qwen3.5-397B-A17B (Qwen Team, 2026)	17B / 397B	512	10	1024	True

"Experts" and "Active Experts" refer to the number of experts and active experts *excluding* shared experts.



(a) Granite-1B-A400M (b) GPT-OSS-20B (c) DeepSeek-V2-Lite

(d) Qwen3.5-35B-A3B (e) Qwen3.5-122B-A10B (f) Qwen3.5-397B-A17B

Figure 1: Accuracy benchmark of MoE models with varying levels of sparsity. Sparsity is applied only to routed experts in (R) results, and sparsity is applied to both routed and shared experts in (R+S) results. For models with shared experts, routed sparsity values that calculate sparsity only using routed experts are provided in red. The sparsity cutoff of the model is annotated with a vertical red dotted line, with total sparsity cutoff values in black and routed sparsity cutoff values in red.

3 Intra-Expert Activation Sparsity

3.1 Methodology

We hypothesize that when a token is routed to an expert, it only needs to activate a small subset of that expert's neurons, not the expert as a whole. We refer to this as **intra-expert activation sparsity**. To validate this claim, we take pre-trained MoE LLMs and measure their benchmark scores at different levels of intra-expert activation sparsity, by sorting each expert's post-activation outputs and zeroing out the lowest-scoring neurons. We do not modify the model architecture or parameters in any way.

We test six MoE models as shown in Table 1 against five benchmarks, namely ARC-Challenge (Clark et al., 2018), ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), Winogrande (Sakaguchi et al., 2019), and TruthfulQA-mc2 (Lin et al., 2021), to cover a wide variety of models and benchmarks. We also use GSM8K Cobbe et al. (2021) and BIG-Bench-Hard Suzgun et al. (2022) benchmarks for generative reasoning, on the lm-eval-harness (Gao et al., 2024) with vLLM (Kwon et al., 2023) backend, and evaluate both with and without applying intra-expert sparsity to the shared experts for models with shared experts. We define the maximum sparsity at which the model retains 95% of its baseline average benchmark score as the representative *sparsity cutoff* value of the model.

3.2 Observation and Analysis

Figure 1 presents the accuracy benchmark of the tested models. As noted in previous studies, we observe the trend that larger models generally have increased robustness to activation sparsity, with the smallest Granite-1B-A400M having the lowest sparsity cutoff at 26.5% and the largest Qwen3.5-397B-A17B having the highest cutoff at 89.3% total sparsity. Unlike previous studies that only covered models up to 30B parameters (Szatkowski et al., 2025; Cheng et al., 2025), our experiment extends the model size up to 397B and confirms that intra-expert activation sparsity scales well

Table 2: Raw benchmark accuracy (%) of Qwen3.5-35B-A3B under uniform static thresholds and learned predictor of Section 4.1. with the Predictor's average achieved neuron sparsity on each benchmark shown in parentheses.

		Routed Expert Sparsity (Static)						Threshold Predictor
Benchmark	Dense	10%	30%	60%	90%	95%	100%	Accuracy (sparsity)
ARC-Challenge	61.5	63.7	61.4	58.4	60.2	59.8	21.6	59.4 (92.7%)
Winogrande	74.7	71.2	70.2	67.2	69.2	69.5	50.3	71.0 (93.1%)
HellaSwag	82.4	82.2	80.5	77.8	77.9	78.0	29.8	78.0 (93.3%)
GSM8K	90.6	88.3	84.5	76.0	76.6	76.0	1.2	89.8 (92.8%)
BIG-Bench-Hard	48.8	48.6	36.7	33.9	34.7	32.4	26.6	48.4 (93.9%)

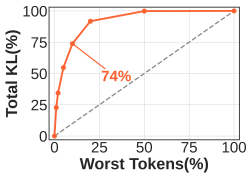


Figure 2: Dense-sparse KL divergence share by the worst $x\%$ tokens.

beyond models with 100B parameters, achieving even higher sparsity levels in these models. From this observation, we conclude that intra-expert activation sparsity is not only readily available in existing models, but also **increasingly useful** in larger models.

Shared Expert Neurons: We find that shared experts have higher per-neuron impact on accuracy compared to neurons of routed experts. For example, in Figure 1c, DeepSeek-V2-Lite is shown to retain its accuracy over a wider range of sparsity in its routed expert-only (R) results compared to GPT-OSS-20B in Figure 1b, despite the smaller parameter count of 16 billion. However, in the dotted (R+S) results, where the sparsity is also applied to the shared experts, its accuracy curve becomes similar to GPT-OSS-20B, with a wider arc and reduced accuracy retention. Similar trends are also observed in other models with shared experts, where the (R) results show a much sharper knee and consistently higher accuracy against the (R+S) results across all sparsity levels.

This phenomenon is more pronounced in larger models, to the point where **less than 2% of the routed expert neurons is enough to retain model capacity** for each token when combined with the full shared expert, as shown by the sparsity cutoff of the Qwen3.5-122B-A10B and Qwen3.5-397B-A17B models. Considering that these models already have 97% and 98% sparsity on the inter-expert level, the total amount of routed expert sparsity available through both inter- and intra-expert sparsity for these Qwen3.5 models is equivalent to *99.95%*. As such, intra-expert sparsity has the potential to bring significant optimizations to both compute and memory usage of MoE models.

Critical Neurons: Despite these results, we find that the intra-expert sparse generation often diverges from dense execution, especially during long, extended generations. Table 2 presents the raw accuracy values of Dense and varying levels of intra-expert activation sparsity of Qwen3.5-35B-A3B. We observe that the discriminative benchmarks of ARC-Challenge, Winogrande, and Hellaswag maintain reasonable accuracy until 95% routed sparsity, and degrade completely at 100% sparsity. In contrast, the two generative reasoning benchmarks of GSM8K and BIG-Bench-Hard suffer *a large drop in accuracy early* and maintain that accuracy until 95% sparsity, before degrading.

Upon further inspection, we find that there exist **critical tokens**, which are especially sensitive to sparsification, and often derail long generation for a few select problems of GSM8K and BIG-Bench-Hard. To quantify this behavior further, we feed the dense outputs of GSM8K to the 95% sparse Qwen3.5-35B-A3B model and calculate the KL divergence of the output logits against a dense model's logits. Figure 2 clearly shows the long-tailed distribution of the offending tokens. The top 5% worst offending tokens are responsible for 55% of the total KL divergence, the top 10% are responsible for 74%, and the top 20% are responsible for 92%.

The existence of critical tokens is also recognized by previous studies in different parts of the MoE architecture (Raposo et al., 2024; Lin et al., 2024). However, our paper is the first to recognize this problem in intra-expert activation sparsity, and therefore no solutions exist at the neuron level. The established approach is to provide more compute to the critical tokens, which translates to neuron budgeting, or making the critical token executions more dense and non-critical token executions more sparse. As such, we design a per-neuron, per-expert sparsity threshold predictor in the next section.

4 Solution Design

4.1 Sparsity Threshold Predictor Training

To build a **sparsity threshold predictor** that can optimally allocate neuron budget across different tokens and experts, we first look into existing expert budgeting studies (Raposo et al., 2024; Lu et al.,

2024; Lin et al., 2024). These works maintain accuracy in MoE models while using fewer experts by allocating more *experts* for the critical tokens and fewer for others based on a lightweight expert budget predictor. Similarly, we propose a lightweight per-token, per-expert neuron budget predictor. However, unlike expert-level predictors that either turn on or off an expert, our solution outputs a continuous threshold value for each expert, which allows more fine-grained control of compute for each expert at the neuron level.

We design our predictor as a two-layer FFN. For layer l, token t with MoE input x_t and expert e, it emits the threshold

$$s_{t,e} = W_2[e] \cdot \mathrm{ReLU}(\mathrm{LN}(x_t)\, W_1 + b_1 + \mathrm{emb}[e]) + b_2[e], \tag{1}$$

Let D be the model hidden size, E the routed experts per layer, L the number of MoE layers, and H the FFN width. LayerNorm and projection $W_1 \in \mathbb{R}^{D\times H}$ are shared by all E experts. Per-expert projection by $W_2 \in \mathbb{R}^{E\times H}$, $b_2 \in \mathbb{R}^E$, and $\mathrm{emb} \in \mathbb{R}^{E\times H}$ maps per-expert. Therefore, there are $L\,(2D + DH + H + 2EH + E)$ parameters in total.

The predictor is distilled against the dense model logits to recover the loss $\mathcal{L}$ from the critical tokens. Let p_d, p_s be the dense and sparse next-token distributions, $y^\star = \arg\max p_d$, and

$$m_s = \log p_s(y^\star) - \max_{j\neq y^\star} \log p_s(j), \qquad m_d = \log p_d(y^\star) - \max_{j\neq y^\star} \log p_d(j), \tag{2}$$

$$\mathcal{L} = w_{\mathrm{KL}} \cdot \mathrm{KL}(p_d \,\|\, p_s) + w_{\mathrm{crit}} \cdot \mathrm{relu}(\min(\delta,\, m_d) - m_s)\,, \tag{3}$$

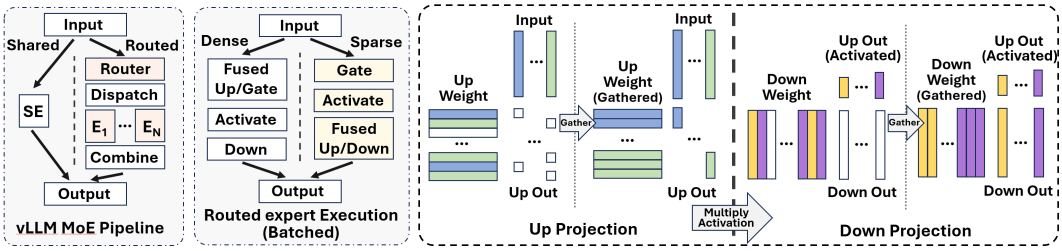
with $w_{\mathrm{KL}} = 0.1$, $w_{\mathrm{crit}} = 1.0$, $\delta = 4$. The hinge term penalizes every critical token equally regardless of its divergence, aiming to recover any divergence in the output, and by capping at m_d we avoid the loss being dominated by logits where dense itself is nearly tied across outputs. For sparsity feedback, we first calculate soft sparsity feedback $\overline{\mathrm{sprs}}_{\mathrm{soft}}$, using sigmoid over the gate output subtracted by the threshold. By adding a forward identity operator on each MoE layer which outputs $\partial\mathcal{L}/\partial\overline{\mathrm{sprs}}_{\mathrm{soft}} = -\lambda$ during backward, our training follows the gradient of $\mathcal{L} - \lambda \cdot \overline{\mathrm{sprs}}_{\mathrm{soft}}$. We update $\lambda \leftarrow \mathrm{clip}(\lambda + \eta(\mathrm{target} - \overline{\mathrm{sprs}}_{\mathrm{hard}}),\, 0,\, \lambda_{\max})$, $\eta = 0.5$ using the exact sparsity $\overline{\mathrm{sprs}}_{\mathrm{hard}}$ of the given step, where target is the target sparsity of the training. As the gradients naturally try to reduce sparsity, this uniform negative feedback to the loss across different experts allows low sparsity for sensitive experts and high sparsity for insensitive experts, independently for each input token.

Due to limited resources, we train the predictor for Qwen3.5-35B-A3B only. The Qwen3.5-35B-A3B predictor has 6.7M parameters at $H = 64$, and the training dataset is a mixture of 50% of the dense model's output on the GSM8K and MATH training sets and 50% from the WikiText-103 training set. We set our target sparsity to 95% and adjust the biases so that the predictor starts from the known 95% static sparsity threshold. We train using the AdamW optimizer at 10^{-3}, no weight decay, 100-step warmup, 1 epoch over the training set for 2500 steps, on a single MI350x GPU for 20 hours. As seen in Table 2, our lightweight predictor is able to recover most of the GSM8K and BIG-Bench-Hard scores, providing an effective solution to critical tokens in intra-expert activation sparsity.

4.2 vLLM Integration

We integrate intra-expert activation sparsity in vLLM as a *separate activation-sparse execution codepath*, while leaving other parts of the system intact for compatibility and performance. As shown in Figure 3a, we modify the expert router and routed per-expert execution code highlighted in orange without changing shared expert execution, dispatch, and combine functions that surround the per-expert execution. Applying activation sparsity to the shared expert has little to no gain, as it is executed in parallel to the routed expert path in the vLLM engine, and its neurons have a high impact on accuracy. Similarly, router, dispatch, and combine functions in routed execution are critical to the performance of expert parallelism (EP) and often use hand-crafted all-to-all communication libraries such as RCCL (AMD, 2020), NCCL (NVIDIA, 2015), or DeepEP (Zhao et al., 2025), which cannot be easily modified without impacting parallelism performance.

Figure 3b visualizes the changes we made to the routed expert execution. We add the activation sparse execution path on the right, highlighted in yellow. The execution path can be dynamically switched between the original dense expert execution and our sparse execution path, depending on the step batch size, as explained later. Operators such as up, gate, activate, and down projections are batched across the experts E_1 to E_N in the EP rank for efficiency in both dense vLLM and our sparse execution, which is skipped in the figure for brevity. Our sparse code has **four key characteristics**



(a) MoE pipeline (b) Sparse codepath (c) Fused up-down projection (Transposed for brevity)

Figure 3: Overview of intra-expert activation sparsity integration in vLLM MoE execution pipeline.

that differ from the dense execution: (1) Standalone gate projection, (2) fused sparsity threshold predictor, (3) Custom sparse activation kernel, and (4) fused sparse up-down projection kernel.

Gate projection must be executed before all sparse operations, as its output is necessary to compute the neuron activations and sparsity mask. We reuse the *batched matrix multiplication* (BMM) kernel of the vLLM dense path, which provides high-throughput expert-major computation and can be easily repurposed by providing the gate weights as input. As the gate projection uses a dense BMM kernel, it limits the scope of sparse computation to roughly two-thirds of the execution (up and down), and maximum speedup to $3\times$. However, we find that overall execution still yields substantial performance improvements, as shown in Section 5.1. Nevertheless, further studies on fast approximation or activation sparsity mask reuse, similar to those in sparse KV cache (Zhang et al., 2023; Li et al., 2024), may allow for a fully sparse expert execution and enable higher performance.

The sparsity threshold predictor $s_{t,e} = W_2[e] \cdot \mathrm{ReLU}(\mathrm{LN}(x_t)\, W_1 + b_1 + \mathrm{emb}[e]) + b_2[e]$ is effectively a two-layer FFN. We fuse the first $x_t\, W_1$ into the expert router projection by concatenating W_1 with the weights of the expert router. This allows us to compute the first FFN term for nearly free as later shown in Table 3, as router projections are small matrix multiplications with low compute utilization, and the small amount of additional computation introduced by W_1 barely affects the execution time. The remaining predictor terms depend on the expert index e. We create a *predictor tail* kernel in Triton (Tillet, 2020) that takes in $x_t\, W_1$ as input and batch processes the remaining predictor computations in a fused kernel across e. We launch it in parallel with the Gate computation to hide its overhead. As the gate computation is a dense matrix multiplication, it can hide the largely memory-bound predictor tail execution.

The per-expert, per-token sparsity threshold from the predictor $s_{t,e}$ is then passed to our custom sparse activation kernel with the gate projection outputs. Our sparse activation kernel fuses the activation calculation and masking of the deactivated neurons into a single kernel using Triton. The kernel computes neuron activations using the desired activation function of the model from the gate projection output and applies threshold-based single-pass masking (Liu et al., 2024; Lee et al., 2024) to create a sparsity mask of the neurons. The activation kernel then collects the activated neuron indices using the mask and returns them in a continuous buffer to be used as the gather index of our token-major up-down kernel.

Finally, we create a fused sparse up-down projection kernel based on the Triton BMM kernel provided by vLLM. As shown in Figure 3c, we first use batched vector load to efficiently *gather the up weight vectors* of each activated neuron for a given input token, using the active neuron index buffer generated by the activation kernel. This creates a packed, dense up weight matrix in the shared memory of the GPU, which can be efficiently computed using a dense matrix-vector multiplication against the input token to generate the up projection output. The output is multiplied by the activation and used as an input to the matrix-vector multiplication with the down-projection weights, which are also gathered and packed into a dense matrix using batched vector loads.

To maintain compatibility with Hip Graph and CUDA Graph, which require reuse of the same GPU memory buffers and kernel launch parameters across graph invocations, the gathered active neuron index buffer is padded to the full neuron size, and the Triton kernels are launched with a tile granularity of a fixed 64 neurons per Triton program. Each Triton program first checks its assigned neuron tile from the active neuron index vector, and exits early if its assigned tile only contains padding indices. This allows for efficient implementation of sparse kernel execution while keeping compatibility with graph capture optimizations, and the fixed 64-neuron tile granularity

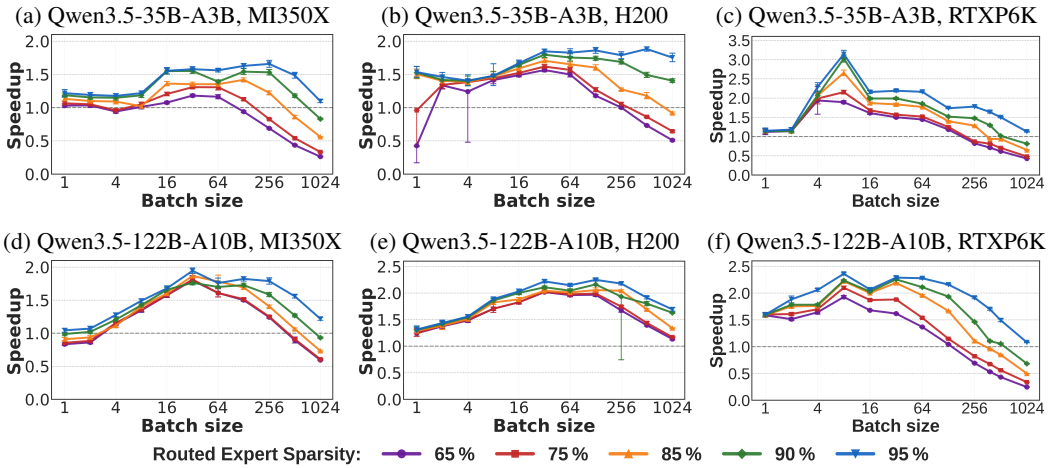


Figure 4: MoE layer speedup of intra-expert activation sparse execution against dense vLLM, averaged across 100 runs with a 96% confidence interval. RTXP6K is RTX Pro 6000 GPU. Note that Qwen3.5-35B-A3B and Qwen3.5-122B-A10B have a sparsity cutoff of over 95%.

enables utilization of GPU hardware accelerators for batched memory access and high-throughput computation instructions.

This approach has a limited maximum throughput compared to the dense BMM in the compute-bound region with large batch sizes. This is a fundamental limitation of a gather-based sparse execution solution, as GPU hardware provides the highest throughput when executing dense operations with contiguous memory access. To overcome this limitation, we profile the tipping batch size between the sparse execution path and the dense execution path during engine startup. Based on this, we switch between sparse and dense execution paths based on the batch size of the running step. This allows us to use the low-latency sparse path when there are fewer workloads, and switch back to the high-throughput dense path when there are more workloads.

5 Evaluation

5.1 MoE Layer Performance

We integrate our intra-expert activation sparse MoE execution into a development build (0.1.dev15965+g6506aec54) of vLLM (Kwon et al., 2023) and evaluate its benefits by comparing its speedup against the original dense vLLM baseline. We use AMD MI350X, NVIDIA H200, and NVIDIA RTX Pro 6000 GPUs for our experiments. For the per-layer evaluations of intra-expert activation sparsity, we use Qwen3.5-35B-A3B and Qwen3.5-122B-A10B models (Yang et al., 2025), with samples from the WikiText-2 (Merity et al., 2016) dataset as input to our experiments.

Figure 4 presents the speedup of intra-expert activation sparse MoE execution over the dense vLLM baseline, measured while executing the first MoE layer of the given model. The sparse execution achieves up to 1.5 to 3.0× speedup depending on configuration. As expected, speedup is highly dependent on batch size. At smaller batch sizes, static overhead such as kernel launch cost offsets the computation savings of sparse execution and gives flat speedup. At larger batch sizes, execution approaches the compute-bound regime, where the higher peak throughput of the dense execution outweighs the benefits of sparsity. Together, these two effects cause the speedup curve to form a convex shape across batch sizes.

Interestingly, the speedup differences across sparsity levels are more pronounced as the batch size increases. When batch size is small, the absolute reduction in computation is small, and static overheads such as the dense gate projection and activation kernel dominate. As batch size increases and execution approaches the compute-bound regime, the degree of sparsity plays a more decisive role: lower sparsity levels of 65 to 75% see a sharper decline in speedup, while higher sparsity levels of 90 to 95% remain more robust across a wider batch size range. Hardware and model size also greatly affect the efficacy of sparse execution, as sparse execution provides greater speedup when

Table 3: Execution time breakdown of Dense, Sparse, and Predictor-based routed expert layers.

Config	Kernel (μs)									Total (μs)
	Router	Top-k	FFN	Gate	Activation	Predictor Tail	Up-Down	Accum.	Copy/Cast	
Dense	5.7	4.6	180.5	–	6.1	–	–	2.5	8.9	208.3
Sparse (90%)	5.6	4.3	–	59.6	9.0	–	60.0	3.7	4.7	146.9
Sparse (95%)	5.5	4.3	–	59.7	9.0	–	33.6	3.4	4.7	120.2
Predictor	5.5	4.4	–	59.6	8.7	2.7	34.9	3.4	5.7	124.9

Table 4: End-to-end Qwen3.5-35B-A3B inference speedup of activation sparse MoE execution against dense vLLM baseline.

GPU	Input Tokens	Output Tokens	Number of Requests						Average Speedup
			1	4	16	64	256	512	
H200	1024	1	1.11×	1.09×	1.08×	1.09×	1.08×	1.10×	1.09×
	1024	128	1.17×	1.15×	1.13×	1.08×	1.06×	1.06×	1.11×
	1	1024	1.18×	1.19×	1.19×	1.19×	1.19×	1.13×	1.18×
	128	1024	1.19×	1.18×	1.19×	1.19×	1.17×	1.13×	1.18×
	1024	1024	1.20×	1.20×	1.20×	1.19×	1.14×	1.13×	1.18×
MI350X	1024	1	1.01×	1.01×	1.01×	1.01×	1.00×	1.00×	1.01×
	1024	128	1.10×	1.08×	1.10×	1.06×	1.03×	1.00×	1.06×
	1	1024	1.04×	1.03×	1.05×	1.06×	1.07×	1.00×	1.04×
	128	1024	1.07×	1.07×	1.08×	1.07×	1.10×	1.00×	1.08×
	1024	1024	1.09×	1.09×	1.09×	1.07×	1.05×	1.00×	1.07×

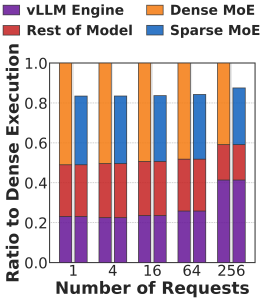


Figure 5: End-to-end execution time ratio.

resource bottlenecks are more severe. Comparing the MI350X, H200, and RTX Pro 6000, it is clear that the speedup is inversely related to the order of compute and memory availability, as the MI350X reaches up to 2.0×, the H200 up to 2.3×, while the RTX Pro 6000 achieves the highest speedup of near 3.0×, close to the theoretical maximum.

Table 3 provides per-kernel execution time breakdown of a single Dense, Sparse, and Predictor-based routed expert layer of Qwen3.5-35B-A3B MoE on H200 at batch size 64. Note that FFN is the sum of the fused Gate-Up and Down projection of the dense execution, and CUDA Graph is turned off for the breakdown measurements. It is clear that the main speedup of the sparse execution comes from reduction in the *Up-Down projection* time. As the gate execution of our implementation is dense, it takes 33% of the FFN time as expected. However, Up-Down at 90% sparsity only takes 50% of the remaining dense FFN time, and 95% and predictor only 25%, providing large speedup.

Remaining kernels largely take insignificant amount of time, for both dense and sparse kernels. Noticeably, the expert router kernel of the predictor-based execution, which has the predictor's W_1 projection fused in it, has virtually no difference in execution time compared to dense or other sparse executions, due to the memory-bound nature of the expert router matrix multiplication. The remaining portion of the predictor is visible as a separate *predictor tail* fused Triton kernel in the breakdown, but it's computation is largely insignificant and are often hidden behind the dense gate projection during the normal CUDA graph enabled executions.

5.2 End-to-end Evaluation

We execute **end-to-end inference and serving benchmarks** using the predictor-based intra-activation sparse Qwen3.5-35B-A3B model. Note that batch-size-based switching between dense and sparse MoE execution is enabled in the vLLM engine for these benchmarks, guaranteeing speedup.

Table 4 presents the inference speedup of the sparse MoE execution over the dense vLLM baseline. We observe up to 1.2× speedup on H200 and up to 1.1× on MI350X, with a minimum speedup of 1.0× as expected. Sparse execution primarily benefits latency-bound workloads, such as decode-heavy configurations with large numbers of output tokens or small numbers of requests. In compute-bound scenarios such as prefill-heavy workloads or executions with large numbers of requests, the benefits are reduced, as the per-step batch size frequently exceeds the switching threshold and the dense path is utilized more often.

Figure 5 shows the execution time ratio of each end-to-end component compared to dense, on an H100 running requests with 1024 input and output tokens. "Dense MoE" and "Sparse MoE" refer to routed expert execution time; "Rest of Model" is the remaining model components such as the attention or embedding layers. "vLLM Engine" refers to the serving overhead of vLLM, including

Table 5: End-to-end serving latency and token batch sizes, running Qwen3.5-35B-A3B on a H200.

Mode	Arrival Rate/s	p50 / p95 / p99			Token batch size per step					Sparse Steps (%)
		TTFT (ms)	TPOT (ms)	E2EL (s)	1–15	16–63	64–255	256–1023	1024+	
Dense	1	160 / 198 / 231	56.3 / 64.8 / 73.4	10.1 / 39.0 / 46.0	316,880	65,440	4,040	7,480	16	0
	4	161 / 189 / 196	58.9 / 61.9 / 65.4	10.5 / 40.7 / 48.8	27,560	76,400	11,720	7,680	160	0
	16	197 / 499 / 690	71.4 / 96.3 / 120.6	13.2 / 44.5 / 53.2	14,184	12,520	19,760	6,440	1,240	0
Sparse	1	142 / 172 / 204	48.8 / 50.3 / 51.9	8.8 / 32.7 / 40.0	409,368	32,400	4,000	12,088	0	98.0
	4	148 / 177 / 213	51.4 / 52.9 / 54.4	9.1 / 34.8 / 41.5	29,984	91,640	6,600	13,040	240	92.3
	16	181 / 583 / 693	64.7 / 86.0 / 114.7	11.6 / 39.5 / 48.4	15,800	11,560	21,080	11,080	2,240	80.7

request scheduling, KV cache management, and output sampling. Across batch sizes, dense MoE layer execution time accounts for roughly 45% of total execution time. While we achieve a 1.5 to 1.8× speedup in the MoE layer, Amdahl's law (Amdahl, 1967) limits the end-to-end improvement to the 1.2× end-to-end speedup observed in Table 4. Nevertheless, the MoE layer constitutes the majority share of model execution time across all batch sizes, making it a high-impact target for optimization.

Table 5 presents the p50/p95/p99 TTFT (time to first token), TPOT (time per output token), and E2EL (end-to-end latency) using vLLM's serving benchmark with inputs from the ShareGPT dataset with Poisson arrival at rates of 1/4/16. We observe a constant 10 to 15% speedup in end-to-end latency. From the table, we clearly see that most batch sizes fall between 1 and 256 tokens across all request rates. Factors such as serving latency and KV cache limitations often bound the decode step execution token range, allowing more than 80% of the steps to be performed using intra-expert sparse execution, where our intra-expert sparse kernel provides good performance benefits.

6 Discussion

Sparse Gate Projection: The current design applies activation sparsity only on the up and down projections, which account for roughly two-thirds of the expert FFN computation. This imposes a theoretical upper bound of 3× on end-to-end speedup even under optimal up/down execution. Thus, future work could explore ways to allow a lightweight gate-activation prediction mechanism that estimates sparse activation patterns before computing the full gate projection, or reusing the sparsity pattern based on the input token. Achieving accurate prediction of the sparsity map before gate projection would allow the system to prevent full dense gate computation, reducing activation overhead and enabling performance gains closer to the theoretical upper bound of intra-expert sparsity, enabling higher end-to-end MoE speedups.

Grouped Intra-expert Activation Sparsity: Our results reveal that sparsity alone does not guarantee speedup, as mapping sparse computation efficiently onto GPU hardware introduces overhead due to irregular memory access, indexing, and limited hardware utilization. This gap between theoretical sparsity and realized speedup widens as batch size increases, underscoring the need for more optimized sparse kernel designs. The most promising direction is to increase the granularity of sparsity mapping: rather than assigning and gathering each individual active neuron, tokens and neurons can be grouped to fit the hardware accelerator width and perform sparsity mapping at this block level. Minimal group sizes between 4 and 32 would better align sparse computation with the regular tile structures expected by high-performance kernels, with only a minor tradeoff in precision, leading to improved memory coalescing and increased hardware utilization.

7 Conclusion

In this work, we demonstrated how intra-expert activation sparsity can improve the efficiency of MoE models. While conventional MoE architectures reduce computation by limiting the number of activated experts per token, our approach further exploits sparsity in the dense FFN of each selected expert. We identified substantial redundancy within the dense expert itself and demonstrated that MoE inference can be further accelerated at inference time. Intra-expert activation sparsity is a promising and practical complement to inter-expert sparsity for reducing the serving cost of large MoE models, with growing relevance as model scale and inference demand continue to rise.

AI USE STATEMENT

In this work, we used generative AI tools to debug the benchmarking and kernel-integration code, and on the paper for minor editing and grammar checks. We did not use generative AI tools to generate the research idea, the experimental claims, or the reported measurements. We have reviewed all AI-assisted work and their contents were validated by the authors.

REFERENCES

AMD. RCCL: ROCm collective communications library. https://github.com/ROCmSoftwarePlatform/rccl, 2020. Software library.

AMD. HIP graph API tutorial. https://rocm.docs.amd.com/projects/HIP/en/latest/tutorial/graph_api.html, 2024a. ROCm HIP Documentation.

AMD. Fusion API: Getting started. https://rocm.docs.amd.com/projects/MIOpen/en/latest/Getting_Started_FusionAPI.html, 2024b. MIOpen ROCm Documentation.

Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pp. 483–485. ACM, 1967. URL https://dl.acm.org/doi/10.1145/1465482.1465560.

Runxi Cheng, Yuchen Guan, Yucheng Ding, Qingguo Hu, Yongxian Wei, Chun Yuan, Yelong Shen, Weizhu Chen, and Yeyun Gong. Mixture of neuron experts. *arXiv preprint arXiv:2510.05781*, 2025.

Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, Xian-Ling Mao, Heyan Huang, and Furu Wei. On the representation collapse of sparse mixture of experts. In *Advances in Neural Information Processing Systems*, volume 35, pp. 34600–34613, 2022. URL https://arxiv.org/abs/2204.09179.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

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

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022. URL https://arxiv.org/abs/2101.03961.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Gemma Team. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pp. 315–323. PMLR, 2011. URL https://proceedings.mlr.press/v15/glorot11a.html.

Granite Team, IBM. Granite 3.0 Language Models. Technical report, IBM, 2024. URL https://github.com/ibm-granite/granite-3.0-language-models.

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.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012. URL https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Je-Yong Lee, Donghyun Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. CATS: Contextually-aware thresholding for sparsity in large language models. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2404.08763.

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, 2021. URL https://arxiv.org/abs/2006.16668.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods, 2021. URL https://arxiv.org/abs/2109.07958.

Zicheng Lin, Tian Liang, Jiahao Xu, Qiuzhi Lin, Xing Wang, Ruilin Luo, Chufan Shi, Siheng Li, Yujiu Yang, and Zhaopeng Tu. Critical tokens matter: Token-level contrastive estimation enhances llm's reasoning capability. *arXiv preprint arXiv:2411.19943*, 2024.

James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. Training-free activation sparsity in large language models, 2024. URL https://arxiv.org/abs/2408.14690.

Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2402.14800.

Yuqi Luo, Chenyang Song, Xu Han, Yingfa Chen, Chaojun Xiao, Xiaojun Meng, Liqun Deng, Jiansheng Wei, Zhiyuan Liu, and Maosong Sun. Sparsing law: Towards large language models with greater activation sparsity, 2024. URL https://arxiv.org/abs/2411.02335.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Meta Llama Team. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. 2025. Meta AI Blog.

NVIDIA. NCCL: NVIDIA collective communications library. https://github.com/NVIDIA/nccl, 2015. Software library.

NVIDIA. CUDA graphs. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-graphs, 2019. CUDA C++ Programming Guide.

OpenAI. gpt-oss-120b and gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Ashwinee Panda, Vatsal Baherwani, Zain Sarwar, Benjamin Therien, Sambit Sahu, Tom Goldstein, and Supriyo Chakraborty. Dense backpropagation improves training for sparse mixture-of-experts. *Advances in Neural Information Processing Systems*, 38:168625–168660, 2026.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

Noam Shazeer. GLU variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

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. URL https://arxiv.org/abs/1701.06538.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. URL https://arxiv.org/abs/1409.1556.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Filip Szatkowski, Patryk Będkowski, Alessio Devoto, Jan Dubiński, Pasquale Minervini, Mikołaj Piórczyński, Simone Scardapane, and Bartosz Wójcik. Universal properties of activation sparsity in modern large language models, 2025. URL https://arxiv.org/abs/2509.00454.

Philippe Tillet. *Triton Language API Documentation*, 2020. URL https://triton-lang.org/main/python-api/triton.language.html. Accessed: 2026-05-06.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

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*, pp. 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. MoEfication: Transformer feed-forward layers are mixtures of experts, 2022. URL https://arxiv.org/abs/2110.01786.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. DeepEP: An efficient expert-parallel communication library. https://github.com/deepseek-ai/DeepEP, 2025. Software library.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Quoc V. Le, James Laudon, and Roy Frostig. Mixture-of-experts with expert choice routing, 2022. URL https://arxiv.org/abs/2202.09368.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. ST-MoE: Designing stable and transferable sparse expert models, 2022. URL https://arxiv.org/abs/2202.08906.