HARDWARE-NATIVE JOINT SPARSE-QUANTIZATION FOR TRILLION-SCALE MIXTURE-OF-EXPERTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) architectures allow frontier language models to scale toward trillions of parameters, but their deployment is constrained by massive memory footprints and bandwidth limits. While modern accelerators feature Sparse Tensor Cores (SpTCs) to reduce weight storage and boost throughput via low-precision semi-structured sparsity, exploiting them in MoEs is hindered by severe degradation of model quality and a lack of grouped sparse GEMM primitives. We present **MoESQ**, an end-to-end hardware–software co-design framework that compresses expert weights into hardware-native low-precision sparse representations and accelerates their execution on SpTC. Algorithmically, **MoESQ** relaxes discrete semi-structured support selection via continuous reparameterization, enabling differentiable joint optimization with quantized weights under a router-weighted reconstruction objective scaled via expert-parallel compression. Systemically, we implement a custom grouped sparse GEMM kernel tailored for sparse, low-precision grouped GEMM on SpTC for MoE inference. Across MoE scales spanning 30B to frontier-scale trillion parameters, **MoESQ** advances state-of-the-art joint sparse-quantization by up to 4.35 percentage points in task accuracy, preserving 96.09% of original model's performance. System-level benchmarks on NVIDIA B200 GPUs show that our kernel outpaces vendor baseline by up to $1.65\times$, driving $1.18\times$ higher serving throughput and up to a $4.03\times$ drop in end-to-end latency. Ultimately, these results establish hardware–software co-design as a practical and necessary pathway for scalable, highly efficient MoE deployment.

1 INTRODUCTION

Recent advances in Mixture-of-Experts (MoE) architectures have emerged as a dominant paradigm for scaling foundation models, pushing parameter counts toward trillions while keeping per-token inference FLOPs manageable through dynamic expert routing (Kimi Team, 2026; DeepSeek-AI, 2026; Qwen Team, 2026b). However, this architectural efficiency introduces severe deployment bottlenecks in real-world serving. Despite activating only a fraction of weights per token, serving massive MoEs typically requires hosting the entire pool of experts in high-bandwidth device memory, leading to prohibitive memory footprints and memory bandwidth limitations under varying batch regimes. Consequently, the sheer storage and memory-bound computational demands of these billions-to-trillions of expert parameters severely limit the practical throughput and cost efficiency of MoE deployment.

A promising yet underutilized opportunity to overcome these memory and compute constraints lies in modern hardware accelerators. Modern architectures feature Sparse Tensor Cores (SpTC) that natively support fine-grained semi-structured ($N : M$) sparse low-precision matrix multiplications, offering a principled path to simultaneously compress weight storage and accelerate computational throughput (NVIDIA Corporation, 2021; 2022; 2024). Despite this capability, the prevailing body of research and deployment efforts for large-scale MoEs has disproportionately concentrated on dense, low-precision quantization and execution (Huang et al., 2025; Fu et al., 2025; Dadgarnia et al., 2026; Egiazarian et al., 2026; NVIDIA Corporation, 2026b;a). As a result, the synergistic potential of hardware-accelerated semi-structured sparsity remains largely unexplored in the context of massive MoE serving.

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.

Realizing these benefits in large-scale MoEs, however, poses two fundamental challenges. First, preserving model fidelity under the compound constraints of semi-structured sparsity and low-precision quantization is remarkably difficult. Prior work has explored joint sparsification and quantization (Frantar & Alistarh, 2023; Guo et al., 2024; 2026), but extending these techniques to trillion-parameter MoEs remains largely unproven, as the rigid semi-structured pattern combined with aggressive quantization exacerbates degradation across diverse expert representations. Second, even when an accurate sparse-quantized model is obtained, translating theoretical reductions into wall-clock speedups is fundamentally bottlenecked by the software execution stack. MoE inference relies on grouped GEMM to efficiently batch variable routing-dependent tokens across experts; however, existing grouped GEMM kernels exclusively target dense formats, leaving hardware-accelerated semi-structured sparse low-precision operations unsupported (NVIDIA Corporation, 2025a).

In this work, we propose **MoESQ**, an end-to-end framework that bridges algorithm design and hardware acceleration for sparse, low-precision MoE serving. At the algorithmic level, **MoESQ** overcomes the combinatorial complexity of rigid semi-structured support selection by relaxing discrete mask decisions through continuous reparameterization. This enables differentiable, joint adaptation of sparse supports and weights under a quantization-aware reconstruction objective. To scale seamlessly to trillion-parameter architectures, **MoESQ** weights expert reconstruction by dynamic routing significance and orchestrates the compression pipeline across distributed experts in parallel. Crucially, to turn these compact representations into tangible deployment gains, we develop a specialized grouped sparse GEMM kernel that executes native semi-structured low-precision operations on SpTC. By co-designing the compression objective with dedicated kernel execution, **MoESQ** achieves both high model fidelity and substantial end-to-end efficiency gains on modern hardware.

Extensive evaluations across MoE architectures spanning from 30B to frontier trillion-parameter scales demonstrate that **MoESQ** sets a new standard in compression fidelity. Specifically, **MoESQ** improves average task accuracy by up to 4.35 percentage points over the strongest joint sparsification and quantization baselines, retaining 96.09% of the uncompressed model's accuracy even at the trillion-parameter scale. At the systems level, our grouped sparse GEMM kernel for NVIDIA B200 GPUs achieves up to a $1.65\times$ speedup over its state-of-the-art dense counterpart. Under matched concurrency, this kernel acceleration translates into a $1.18\times$ higher end-to-end serving throughput over the fastest dense low-precision backends, up to a $4.03\times$ reduction in end-to-end latency compared to the INT4 Marlin backend (Frantar et al., 2025), and the lowest time-per-output-token across all evaluated systems. Taken together, these results demonstrate that our hardware–software co-design enables high-fidelity compression while reducing memory footprint and improving serving efficiency for large-scale MoE inference.

2 Background

A sparsely activated Mixture-of-Experts (MoE) layer replaces the dense feed-forward network with E experts and routes each token to only $k \ll E$ of them (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022), decoupling total parameter count from per-token compute (Ludziejewski et al., 2024; Abnar et al., 2025; Nakamura et al., 2026). Recent open-weight MoEs span roughly 750B–2.8T total parameters while activating only 32B–104B parameters per token (Z.ai, 2026; LG AI Research, 2026; Kimi Team, 2026; DeepSeek-AI, 2026; Qwen Team, 2026b; Moonshot AI, 2026). Although sparse routing limits per-token computation, deployment must store and make available the full expert pool, causing expert-weight memory to scale with total rather than active parameters. Kimi-K2.5 illustrates this disparity: the model contains 1T total parameters but activates only 32B per token, while its released packed INT4 checkpoint occupies approximately 595 GB (Kimi Team, 2026; vLLM Project, 2026).

Modern hardware accelerators provide an opportunity to reduce both memory and computation by operating directly on sparse low-precision representations. Sparse Tensor Cores (SpTCs) exploit semi-structured $N{:}M$ sparsity by encoding nonzero values with sparse metadata and skipping operations on structured zeros, while simultaneously supporting low-precision data formats (Mishra et al., 2021; Open Compute Project Foundation (MX Alliance), 2023; Rouhani et al., 2023; NVIDIA, 2025). Because this path reduces arithmetic, vendors commonly use sparse throughput to report the maximum performance of recent accelerators (NVIDIA Corporation, 2021; 2022). For instance, on

the NVIDIA Blackwell architecture, block-scaled FP4 formats (NVFP4 and MXFP4) can be coupled with *paired* 4:8 *sparsity* on fifth-generation SpTCs, providing up to 18 PFLOP/s for sparse NVFP4, twice the throughput of the dense NVFP4 path (NVIDIA Corporation, 2025a; 2024). Realizing this opportunity in routed MoE inference requires both expert weights that satisfy the exact paired sparsity, microscaling, and metadata constraints of the hardware representation and an execution path that can operate on this representation across dynamically routed experts.

The first requirement can be addressed by combining sparsity and quantization, an approach studied primarily to push model compression beyond standalone sparsification (Sun et al., 2024; Lee et al., 2026) or quantization (Frantar et al., 2023; Lin et al., 2024; Tseng et al., 2024; Dadgarnia et al., 2026). SparseGPT (Frantar & Alistarh, 2023) sparsifies LLMs through second-order error compensation and can subsequently quantize the remaining weights for further compression. The interaction between sparsification and quantization has also been analyzed (Harma et al., 2025; Kim et al., 2026), and more recent methods explicitly account for it: JSQ (Guo et al., 2024) uses a quantization-aware pruning metric and activation editing to reduce quantization error, while OBR (Guo et al., 2026) rotates, sparsifies, compensates, and then quantizes the weights. Because these methods were developed and evaluated primarily for compressing dense LLMs, their solution quality under the target hardware-native constraints and their scalability to trillion-parameter MoE models remain unclear.

Producing hardware-compliant sparse-quantized weights addresses only the first requirement; these weights must also be executed efficiently under routed MoE workloads. Modern GPU libraries provide single-problem GEMM kernels that combine semi-structured sparsity with low-precision arithmetic, including 2:4 INT8 on Ampere, 2:4 FP8 on Hopper, and paired 4:8 NVFP4 on Blackwell (Mishra et al., 2021; NVIDIA Corporation, 2022; 2025a). Routed MoE inference, however, relies on grouped GEMM to evaluate the expert products $Y_e = X_e W_e^\top$ in a single launch even though routing gives each expert a different token dimension $N_e = |X_e|$. Although low-precision grouped GEMM kernels, including NVFP4 grouped GEMM, are available, no publicly available sparse grouped GEMM path directly supports the target paired 4:8 NVFP4 representation (NVIDIA Corporation, 2025b). Hardware-native compression alone is therefore insufficient for efficient MoE execution; realizing its benefits also requires a sparse low-precision grouped GEMM kernel that directly processes the compressed expert weights.

Taken together, these gaps leave the capabilities of SpTCs disconnected from modern MoE deployment: no established path simultaneously preserves model fidelity for trillion-parameter scale MoE models and translates sparse low-precision hardware support into reduced expert storage and accelerated routed execution. Bridging this divide requires hardware–software co-design that aligns MoE compression and routed execution with hardware-native representation constraints.

3 METHOD

We propose **MoESQ**, which consists of an algorithmic procedure that learns hardware-native sparse-quantized representations for expert weights and a grouped sparse GEMM that executes the same representation on SpTC. We first describe how **MoESQ** jointly adapts sparse supports and quantized values for MoE models, then detail its implementation and corresponding execution kernel.

3.1 PROBLEM FORMULATION

We seek a compressed parameterization $\widetilde{W}$ that minimizes the reconstruction error of a targeted module f (e.g., a linear projection or an MoE expert) evaluated on calibration inputs X:

$$\min_{\widetilde{W}} \mathcal{L}(\widetilde{W}) := \left\| f(X; W) - f(X; \widetilde{W}) \right\|_F^2 \quad \text{s.t.} \quad \widetilde{W} \in \mathcal{C} := \mathcal{S} \cap \mathcal{Q}, \tag{1}$$

where $\mathcal{S}$ denotes the set of hardware-admissible semi-structured sparsity patterns (e.g., paired 4:8), and $\mathcal{Q}$ denotes the set of admissible quantized values accompanied by their associated shared scaling factors (e.g., NVFP4). Because support selection, quantized representation, and shared scale estimation are tightly coupled, Equation (1) constitutes a challenging discrete combinatorial optimization problem.

3.2 Learning Hardware-Native Sparse-Quantized Representations

To establish a differentiable formulation, we relax each discrete support selection into a continuous distribution over admissible patterns via the Gumbel–Softmax reparameterization trick (Jang et al., 2017; Maddison et al., 2017; Fang et al., 2024).

Consider a weight matrix $W \in \mathbb{R}^{N\times K}$ partitioned into R structured groups of size m. Each group $W_{[r]} \in \mathbb{R}^m$ is constrained to take a pattern from a predefined set of admissible binary vectors $\mathcal{P} = \{\rho_1, \ldots, \rho_P\} \subset \{0,1\}^m$. For example, under a paired 4:8 sparse format ($m = 8, P = 6$), each pattern ρ_p retains two non-zero pairs of weights (see Appendix A for full specifications). Let $A \in \mathbb{R}^{R\times P}$ represent a logit matrix scoring the candidate patterns, where $A_{r,p}$ measures the affinity of group r for pattern ρ_p. At optimization step t, we sample independent and identically distributed (i.i.d.) Gumbel noise to construct the relaxed support mask:

$$G_{t,r,p} \overset{\text{iid}}{\sim} \text{Gumbel}(0,1), \qquad \pi_{t,r,:} = \text{softmax}\left(\frac{A_{r,:} + G_{t,r,:}}{\tau_t}\right), \qquad Z_{t,[r]} = \sum_{p=1}^{P} \pi_{t,r,p}\rho_p, \quad (2)$$

where $\tau_t > 0$ denotes a temperature hyperparameter governing the softness of the categorical distribution $\pi_{t,r,:}$. This continuous relaxation provides a differentiable surrogate for support selection, enabling gradient-based updates to the pattern logits A. Annealing τ_t toward zero over training drives the relaxed support mask $Z_{t,[r]}$ toward a deterministic, one-hot selection of an admissible pattern. Although the Gumbel–Softmax relaxation renders support selection differentiable, it does not implicitly enforce the quantization constraint $\mathcal{Q}$. To address this, we introduce an auxiliary full-precision matrix $V \in \mathbb{R}^{N\times K}$ and couple it with the relaxed support mask to form the soft-masked weight matrix $Y_t := Z_t \odot V$.

A target quantization operator $\mathcal{Q}_w$ subsequently projects Y_t into $\mathcal{Q}$. Let $\mathcal{G}_w$ denote the admissible target value grid. For each quantization block $Y_{t,[b]}$, the operator $\mathcal{Q}_w$ derives a local scaling factor $s_{t,b}$ according to the target precision format and projects normalized weights onto $\mathcal{G}_w$:

$$s_{t,b} := \text{Scale}_w(Y_{t,[b]}), \qquad \mathcal{Q}_w(Y_t)_{[b]} := s_{t,b}\,\Pi_{\mathcal{G}_w}\left(\frac{Y_{t,[b]}}{s_{t,b}}\right), \quad (3)$$

where $\text{Scale}_w(\cdot)$ is the scale-computation function (e.g., maximum absolute value), and $\Pi_{\mathcal{G}_w}(\cdot)$ denotes the nearest-neighbor projection onto grid $\mathcal{G}_w$.

During backward passes, we treat scaling factors $s_{t,b}$ as constants and employ the Straight-Through Estimator (STE) (Bengio et al., 2013) to propagate gradients through the non-differentiable projection $\Pi_{\mathcal{G}_w}$. Consequently, the reconstruction loss jointly updates the support logits A and auxiliary weights V, ensuring that candidate supports are evaluated directly under their quantization-constrained realization.

3.3 Scalable Blockwise Expert Compression

To scale continuous optimization across MoE models, we follow a blockwise per-expert decomposition inspired by GSQ (Dadgarnia et al., 2026). At block l, the router assigns set of token representations $X_{l,e}$ to expert e and their corresponding gating coefficients into the aligned vector $w_{l,e}$. Evaluating the uncompressed expert on these inputs provides the reconstruction target $Y_{l,e} = f_{l,e}(X_{l,e}; \theta_{l,e})$. We reconstruct each expert as a unified gate–up–down block $\theta_{l,e} = (W_{\text{gate}}, W_{\text{up}}, W_{\text{down}})$, with separate support logits and latent weights for each projection and dynamic activation quantization $\mathcal{Q}_a$ inside the reconstruction loop. Writing $f_{l,e}^{\mathcal{Q}_a}(X; \widetilde{\theta})$ for activation-quantized expert execution, we optimize the router-weighted reconstruction objective:

$$\mathcal{L}_{l,e}^{\text{rw}}(\widetilde{\theta}) := \left\| \text{diag}(w_{l,e})\left(Y_{l,e} - f_{l,e}^{\mathcal{Q}_a}(X_{l,e}; \widetilde{\theta})\right)\right\|_F^2. \quad (4)$$

This objective scales each token residual by the same gate coefficient that weights the expert output in the routed MoE layer. As summarized in Algorithm 1, experts within a block are optimized independently across devices, after which the finalized block generates calibration inputs for block $l + 1$.

Algorithm 1 Hardware-Native Expert Compression for Mixture-of-Experts

Require: MoE model with L blocks and E experts; calibration data $\mathcal{D}$; pattern set $\mathcal{P}$; quantizers $\mathcal{Q}_w, \mathcal{Q}_a$; temperature schedule $\{\tau_t\}_{t=1}^{T}$; optimization steps T

$H_1 \leftarrow \mathrm{Embed}(\mathcal{D})$
for $l = 1, \dots, L$ **do**
  $\{X_{l,e}, w_{l,e}\}_{e=1}^{E} \leftarrow \mathrm{Route}_l(H_l)$
  for all $e \in \{1, \dots, E\}$ **in parallel do**
    $Y_{l,e} \leftarrow f_{l,e}(X_{l,e}; \theta_{l,e})$
    Initialize $\{A^{(j)}, V^{(j)}\}_{j \in \{\mathrm{gate,up,down}\}}$
    for $t = 1, \dots, T$ **do**
      for $j \in \{\mathrm{gate}, \mathrm{up}, \mathrm{down}\}$ **do**
        Sample noise $G_t^{(j)}$ and construct $Z_t^{(j)}$ via Equation (2)
        $Y_t^{(j)} \leftarrow Z_t^{(j)} \odot V^{(j)}$
        $\widetilde{W}_t^{(j)} \leftarrow \mathcal{Q}_w(Y_t^{(j)})$ via Equation (3)
      end for
      $\widetilde{\theta}_t \leftarrow \{\widetilde{W}_t^{(j)}\}_{j \in \{\mathrm{gate,up,down}\}}$
      Update parameters $\{A^{(j)}, V^{(j)}\}_j$ using $\nabla \mathcal{L}_{l,e}^{\mathrm{rw}}(\widetilde{\theta}_t)$
    end for
    $Z_{[r]}^{\star(j)} \leftarrow \rho_{\mathrm{argmax}_p A_{r,p}^{(j)}}$ for every group r and projection j
    $\theta_{l,e}^{\star} \leftarrow \{\mathcal{Q}_w(Z^{\star(j)} \odot V^{(j)})\}_{j \in \{\mathrm{gate,up,down}\}}$
  end for
  $H_{l+1} \leftarrow \mathrm{Block}_l(H_l; \{\theta_{l,e}^{\star}\}_{e=1}^{E})$
end for
return $\{\theta_{l,e}^{\star}\}_{l,e}$

3.4 Implementation Details

We keep the router and all non-expert parameters fixed during compression. For each block l, we route the current calibration states H_l once and cache the expert inputs $X_{l,e}$, routing weights $w_{l,e}$, and uncompressed targets $Y_{l,e}$ for reuse throughout refinement. The experts in a block then form independent optimization problems: we shard them across devices, optimize the assigned experts locally, and synchronize only after all experts have been finalized. The resulting sparse-quantized experts are assembled into the block and applied to H_l to produce the calibration states for block $l+1$, so each block observes the error accumulated by the compressed prefix.

We initialize the support logits and latent weights from an SGPTQ solution (Frantar & Alistarh, 2023; Frantar et al., 2023) and recompute quantization scales from the current soft-masked weights on every refinement forward pass. For NVFP4 (NVIDIA, 2025), the block and tensor scales are computed as

$$\mathrm{Scale}_w(Y_{t,[b]}) = \frac{\Pi_{\mathrm{E4M3}}\left(\gamma_t \|Y_{t,[b]}\|_\infty / 6\right)}{\gamma_t}, \qquad \gamma_t = \frac{448}{\max_b \|Y_{t,[b]}\|_\infty / 6}, \tag{5}$$

where 6 and 448 are the maximum magnitudes of the FP4-E2M1 value grid and FP8-E4M3 scale format, respectively. We stop gradients through these scales, while the dynamic activation quantizer independently computes scales for the block input and the intermediate SwiGLU activation. After refinement, we select the highest-logit pattern in every sparsity group, recompute the final scales, and pack the weights as E2M1 values, sparse metadata, and one UE4M3 scale per 32 logical weights. Full optimization settings are reported in Appendix B.

Grouped sparse GEMM execution on SpTC. A routed MoE layer evaluates a collection of independent expert products $Y_e = X_e W_e^\top$, whose token dimensions N_e vary with the routing decisions. Grouped GEMM generalizes batched GEMM by allowing these problems to have different matrix dimensions, transpositions, and scaling factors while executing them within one kernel launch (Hejazi, 2024). This organization avoids a separate launch for each expert while preserving its routing-dependent problem shape.

Table 1: OpenLLM Leaderboard v1 results across three MoE scales under the paired 4:8 NVFP4 W4A4 expert representation. Recovery (Rec.) is relative to each model's original average accuracy; bold marks the best compressed result.

Model	Method	ARC-c	GSM8K	MMLU	WG	HSwag	TQA	Avg	Rec.
Kimi-K2.5	Reference	74.06	94.39	89.53	81.93	91.96	62.69	82.43	100%
	Dense NVFP4	73.21	93.35	89.23	81.45	91.87	62.29	81.90	99.36%
	JSQ	59.22	68.92	71.87	73.88	80.95	50.58	67.57	81.97%
	SGPTQ	63.05	79.30	82.10	79.72	84.39	54.92	73.91	89.67%
	OBR	63.65	82.79	82.49	80.35	85.34	55.78	75.07	91.07%
	Ours	**69.97**	**88.70**	**85.34**	**82.40**	**88.57**	**60.28**	**79.21**	**96.09%**
Qwen3.5-397B-A17B	Reference	79.10	85.06	89.49	76.16	90.85	60.97	80.27	100%
	Dense NVFP4	79.15	78.85	89.49	75.95	90.75	59.59	78.96	98.37%
	JSQ	**75.51**	4.62	84.29	78.06	82.43	52.80	62.95	78.43%
	SGPTQ	74.74	49.81	85.67	78.77	81.02	55.15	70.86	88.27%
	OBR	75.00	65.28	86.00	**78.85**	81.89	**55.30**	73.72	91.84%
	Ours	74.49	**71.65**	**86.97**	78.30	**84.74**	55.10	**75.21**	**93.70%**
Qwen3-30B-A3B	Reference	69.45	89.54	79.34	69.61	77.92	53.78	73.27	100%
	Dense NVFP4	67.80	88.32	78.04	70.38	77.15	52.52	72.37	98.77%
	JSQ	52.13	65.88	60.70	62.19	51.68	43.14	55.96	76.37%
	SGPTQ	56.14	72.86	66.51	65.98	56.33	48.72	61.09	83.37%
	OBR	56.23	72.40	66.60	65.04	57.50	48.76	61.09	83.37%
	Ours	**59.73**	**79.45**	**69.89**	**67.80**	**63.60**	**52.15**	**65.44**	**89.30%**

To extend grouped execution to support paired 4:8 NVFP4 weights, we combine the Blackwell block-scaled sparse MMA mainloop with the grouped-problem abstraction and persistent scheduler that CUTLASS exposes separately (NVIDIA Corporation, 2025a;b). Because the sparse MMA requires the sparse matrix as its left operand A, we place $W_e \in \mathbb{R}^{M\times K}$ on A and compute $Y_e^\top = W_e X_e^\top$ (NVIDIA Corporation, 2025a). This transformation is zero-copy: row-major $X_e \in \mathbb{R}^{N_e\times K}$ has the same memory layout as the column-major operand B, and the column-major $M \times N_e$ result has the row-major layout expected for Y_e.

We pack the exported weights offline as E2M1 values, sparse metadata, and one UE4M3 scale per 32 logical weights, and quantize expert activations at run time into corresponding 32-element scale blocks using $\mathcal{Q}_a$. The sparse MMA applies the block scales of both operands, while the epilogue combines their tensor-wide scales into a per-expert factor α_e before casting the FP32 accumulator to BF16. Further details on kernel configuration, activation quantization, and serving integration are provided in Appendix D.1.

4 Experiments

We evaluate whether **MoESQ** addresses both requirements for hardware-native sparse MoE deployment: preserving model fidelity under the target representation and translating this representation into kernel and end-to-end serving gains. We then additionally compare its accuracy–memory–serving tradeoff with other state-of-the-art expert-compression methods.

4.1 Setup

Models & compression format. We compress experts of Qwen3-30B-A3B, Qwen3.5-397B-A17B (Qwen Team, 2026a), and Kimi-K2.5 (Kimi Team, 2026) with paired 4:8 NVFP4 W4A4 format (see Appendix A for format details) supported by NVIDIA's Blackwell architecture. The fidelity of compressed checkpoint is then measured on general knowledge and reasoning tasks, with original released checkpoint (BF16 for Qwen and native INT4 for Kimi), denoted Reference, and NVIDIA's dense NVFP4 checkpoints (NVIDIA Corporation, 2025c; 2026b;a) provided as a baseline.

Compression baseline. For joint sparsification and quantization baseline, JSQ (Guo et al., 2024), SGPTQ (Frantar & Alistarh, 2023; Frantar et al., 2023), and OBR (Guo et al., 2026) is adapted to MoE

Table 2: Kimi-K2.5 reasoning results under paired 4:8 NVFP4 W4A4 compression. Acc. is mean pass@1 over N runs; Trunc. is the fraction exhausting the 65,536-token budget, and Rec. is relative to the reference.

	AIME25 $N=10$		GPQA Diamond $N=5$		MATH500 $N=5$			
Method	Acc.	Trunc.	Acc.	Trunc.	Acc.	Trunc.	Avg	Rec.
Reference	95.67	3.7	89.49	0.7	96.36	0.1	93.84	100%
JSQ	7.33	88.3	13.43	72.0	27.76	67.0	16.17	17.24%
SGPTQ	68.33	15.3	68.48	2.5	94.24	1.5	77.02	82.07%
OBR	74.67	12.0	71.62	1.0	94.64	0.6	80.31	85.58%
Ours	**84.67**	6.3	**76.46**	0.3	**96.28**	0.4	**85.80**	**91.44%**

compression; Appendix B contains details on baseline implementation, compression hyperparameters. Note that these one-shot baselines do not provide a corresponding iterative stage to which the same optimization budget can be allocated, whereas our differentiable formulation enables additional refinement from the SGPTQ solution at a higher compression cost; Appendix C.1 isolates the design choices within this refinement. Broader comparison with expert compression includes 2-bit GSQ (Dadgarnia et al., 2026) and REAP expert pruning (Lasby et al., 2026), with the similar compression ratio to **MoESQ**.

Grouped GEMM kernels. We compare **MoESQ** against dense grouped GEMM baselines: precision-matched NVFP4 W4A4 CuTeDSL (NVIDIA Corporation, 2025a), and mainstream weight-only kernels including INT4 W4A16 Marlin (Frantar et al., 2025) and INT2 W2A16 Humming (inclusionAI, 2026). Implementation details, kernel profiling, and serving configurations can be found in Appendix D.

4.2 Accuracy Evaluation

General knowledge. To evaluate general knowledge preservation, we benchmark compressed models on the OpenLLM Leaderboard v1 suite (Hugging Face, 2024) (Table 1). Across all scales, **MoESQ** consistently outperforms joint sparse-quantization baselines, demonstrating that continuous reparameterization successfully shields expert representations from compound compression noise. This resilience is particularly evident on multi-step arithmetic tasks: on Qwen3.5-397B-A17B (GSM8K), **MoESQ** achieves 71.65% accuracy compared to 65.28% for OBR and a near-total collapse to 4.62% for JSQ, where decoupled compression disrupts expert computation. Overall, **MoESQ** retains 93.70% of reference performance on Qwen3.5-397B-A17B, while leading OBR by 5.93 recovery points (89.30% vs. 83.37%) on Qwen3-30B-A3B and 5.02 points (96.09% retention) on trillion-parameter Kimi-K2.5.

Reasoning. Next, we assess whether complex reasoning capabilities are preserved by evaluating compressed Kimi-K2.5 on AIME25 (Zhang & Math-AI, 2025), GPQA Diamond (Rein et al., 2024), and MATH500 (Hendrycks et al., 2021) under a constrained token generation budget (pass@1 averaged over N trials; Appendix B). As summarized in Table 2, **MoESQ** outperforms competing baselines across all tasks, attaining a dominant overall performance recovery of 91.44%. Crucially, these benchmarks reveal that multi-step reasoning is vulnerable to compression noise, which often corrupts generation trajectories and triggers non-terminating loops that exhaust token budgets (Appendix C.3). This degradation severely impacts baselines: JSQ suffers an 88.3% truncation rate on AIME25, collapsing its overall performance recovery to just 17.24%. In contrast, **MoESQ** maintains minimal truncation rates comparable to the uncompressed baseline.

Together, these results confirm that **MoESQ** effectively mitigates compound distortion by unifying continuous sparse support relaxation with joint weight quantization, safeguarding both factual knowledge and generation stability in large-scale MoEs.

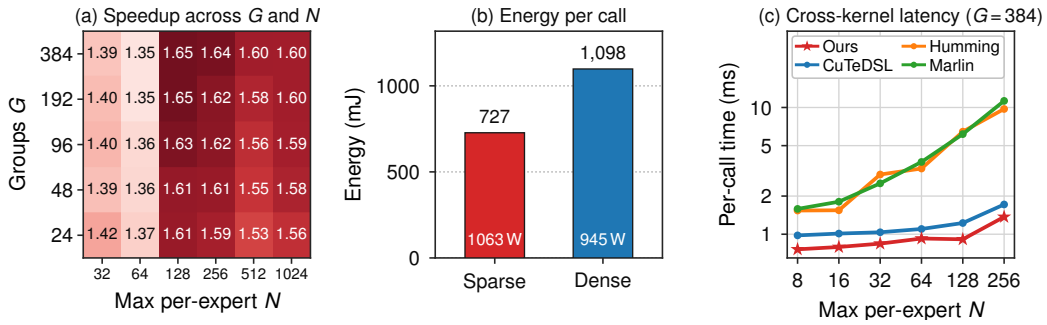

Figure 1: Kernel-level evaluation of the grouped sparse GEMM on the Kimi-K2.5 gate/up layer. (a) Median runtime speedup of the grouped sparse GEMM kernel over dense (CuTeDSL) baseline. (b) Energy to solution at iso-frequency, where the label above each bar indicates the energy per call (mJ) and the label inside each bar indicates the mean board power (W) of each kernel; measured on 1xB300. (c) Median per-call latency of the deployment kernels across a range of token counts.

4.3 Efficiency Evaluation

Kernel-level efficiency. We first evaluate whether the custom grouped sparse GEMM kernel of **MoESQ** effectively unlocks SpTC compute density under dynamic MoE routing. To isolate the performance impact of semi-structured sparsity, we benchmark **MoESQ** against vendor dense grouped GEMMs (CUTLASS NVFP4 CuTeDSL) using Kimi-K2.5 expert projection shapes across varying group counts G and per-expert batch sizes N (Appendix D.2). As shown in Figure 1(a), **MoESQ** achieves a 1.35–$1.65\times$ speedup over the dense baseline across all configurations, demonstrating robust performance across various configurations. This acceleration directly translates into superior energy efficiency: at peak speedup, shortened execution time reduces total energy consumption from $1{,}098$ to 727 mJ ($1.51\times$ improvement; Figure 1(b)), showing that higher SpTC compute throughput offsets transient board power by accelerating execution to finish faster. Finally, across the spectrum from decode ($N = 8$) to prefill ($N = 256$) regimes (Figure 1(c)), **MoESQ** consistently outpaces both dense FP4 and weight-only backends (INT4 Marlin, INT2 Humming).

End-to-end serving. We next evaluate whether these kernel-level advantages translate into system-wide performance gains during production serving. Using vLLM on an $8\times$ NVIDIA B200 system, we serve Kimi-K2.5 across isolated prefill and decode workloads, varying only the expert execution backend for a fair comparison (see Appendix D.3 for setup details). In prefill-dominated workloads (4096 in/1 out tokens), **MoESQ** consistently yields the highest throughput across all concurrency levels, reaching $55{,}654$ tokens/s at a concurrency of 128—outperforming the fastest baseline (CuTeDSL) by $1.18\times$–$1.25\times$ (Figure 2(a)). Under fully saturated decode workloads (384 in/128 out tokens, 500 in-flight requests), **MoESQ** achieves $8{,}983$ output tokens/s, reducing mean Time Per Output Token (TPOT) to 32.2 ms and total request latency to 6.2 s (Figure 2(b)–(d)). This translates to a $1.18\times$ boost in decode throughput along with 5.9 ms faster TPOT and a 1.1 s drop in end-to-end latency compared to dense CuTeDSL.

Together, these system- and kernel-level evaluations demonstrate that co-designing hardware-native SpTC execution with sparse low-precision compression alleviates the execution bottlenecks of trillion-parameter MoEs, delivering concurrent gains in throughput, latency, and energy efficiency.

4.4 Expert Compression Comparison

Having evaluated accuracy and efficiency independently, we next compare **MoESQ** against state-of-the-art expert pruning (Lasby et al., 2026) and dense quantization (Dadgarnia et al., 2026) on Kimi-K2.5 under comparable 2–3-bit-per-expert weight budgets to examine the joint accuracy–memory–serving tradeoff (Table 3). Within this low-bit regime, **MoESQ** recovers 96.09% of reference accuracy, outperforming dense 2-bit GSQ by 1.87 percentage points while trailing structural pruning via REAP by 1.90 percentage points. Importantly, **MoESQ** delivers substantial serving improvements: it achieves the highest output throughput ($8{,}983$ tok/s), lowest mean TTFT ($2{,}120$ ms), and lowest

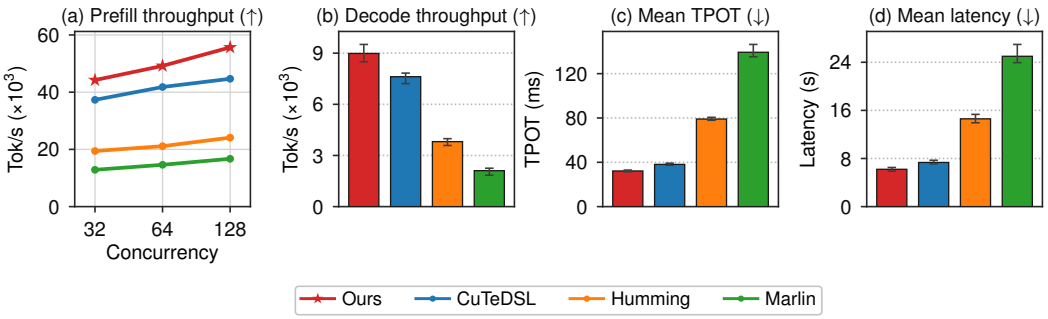

Figure 2: End-to-end serving of Kimi-K2.5 on 8×B200 with a shared vLLM stack, varying only the expert representation and corresponding grouped-GEMM backend (Table 12b contains full numbers). (a) Prefill throughput (total tokens/s) across concurrency, with (4096 in/1 out) requests. (b)–(d) Decode (384 in/128 out) throughput (output tokens/s), mean Time Per Output Token (TPOT), and mean end-to-end request latency, with all 500 requests in flight. Bars in (b)–(d) are the mean of three consecutive repeats, and whiskers show their range.

Table 3: Comparison with alternative expert-compression methods on Kimi-K2.5. Serving metrics are measured with vLLM on 8×B200 under the same decode workload and matched concurrency (384 input/128 output tokens per request, all 500 requests in flight) and averaged over three repeats. Bit counts include quantization scales and sparse metadata, KV cache is the engine-reported capacity, and bold marks the best compressed result.

			OpenLLM v1		Serving			
Method	Expert format	bits/expert weight	Avg. acc. ↑	Recovery ↑	KV cache (tokens) ↑	Output tok/s ↑	TTFT (ms) ↓	E2E (s) ↓
Reference	Dense INT4 W4A16	4.50	82.43	100%	1,055k	2,108	7,306	25.0
GSQ	Dense INT2 W2A16	2.13	77.66	94.22%	1,594k	3,811	4,560	14.6
REAP	50% expert pruning + INT4	2.25	**80.77**	**97.99%**	**1,632k**	3,190	5,366	17.1
Ours	Paired 4:8 NVFP4 W4A4	2.75	79.21	96.09%	1,433k	**8,983**	**2,120**	**6.2**

end-to-end latency (6.2 s), surpassing the fastest baseline by 2.36× in throughput and reducing TTFT by 2.15×. Although GSQ and REAP offer slightly larger KV-cache capacities due to lower expert storage footprints, both baselines yield lower throughput and higher serving latencies under the same workload and concurrency constraints. These results indicate that reducing expert memory footprint alone does not necessarily yield efficient serving, whereas hardware-native algorithm–hardware co-design provides a favorable balance among accuracy, memory overhead, and end-to-end efficiency.

5 Conclusion

In this work, we address the critical deployment bottleneck of Mixture-of-Experts models by exploiting hardware-accelerated Sparse Tensor Cores—an essential opportunity previously overlooked in large-scale MoE serving. To bridge this gap, we propose **MoESQ**, an end-to-end hardware–software co-design framework that aligns hardware-native sparse low-precision representations with their efficient, routed execution. By combining differentiable joint sparsification and quantization with a dedicated grouped sparse GEMM kernel, **MoESQ** achieves high-fidelity compression and superior serving performance at the trillion-parameter scale. Evaluations on recent hardware demonstrate that **MoESQ** retains 96.09% of reference accuracy while delivering up to a 1.65× kernel speedup, translating to a 1.18× throughput increase and up to a 4.03× latency reduction over standard baselines. These findings establish that hardware–software co-design provides a highly practical paradigm to overcome memory constraints and unlock scalable MoE deployment.

AI USE STATEMENT

In this work, we used generative AI tools for creating scientific figures or images, editing software code, edit a paper to improve readability, format references. We have reviewed all AI-assisted work: checking LLM-generated reference formats, code implementations of methods were verified by lead author and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Samira Abnar, Harshay Shah, Dan Busbridge, Alaaeldin El-Nouby, Joshua M. Susskind, and Vimal Thilak. Parameters vs FLOPs: Scaling laws for optimal sparsity for mixture-of-experts language models. In *International Conference on Machine Learning (ICML)*, 2025. URL https://proceedings.mlr.press/v267/abnar25a.html.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. URL https://arxiv.org/abs/1308.3432.

Roberto L. Castro and Dan Alistarh. Qutlass: Cutlass-powered quantized blas for deep learning. https://github.com/IST-DASLab/qutlass, 2025.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2302.06675.

Alireza Dadgarnia, Soroush Tabesh, Mahdi Nikdan, Michael Helcig, Eldar Kurtic, Maximilian Kleinegger, and Dan Alistarh. GSQ: Highly-accurate low-precision scalar quantization for LLMs via Gumbel-Softmax sampling. *arXiv preprint arXiv:2604.18556*, 2026. URL https://arxiv.org/abs/2604.18556.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026. URL https://arxiv.org/abs/2606.19348.

Vage Egiazarian, Roberto Castro, Denis Kuznedelev, Andrei Panferov, Eldar Kurtic, Shubhra Pandit, Alexandre Marques, Mark Kurtz, Saleh Ashkboos, Torsten Hoefler, et al. Bridging the gap between promise and performance for microscaling fp4 quantization. In *International Conference on Learning Representations (ICLR)*, volume 2026, 2026.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. Maskllm: Learnable semi-structured sparsity for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2409.17481.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research (JMLR)*, 2022. URL https://arxiv.org/abs/2101.03961.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning (ICML)*, 2023. URL https://proceedings.mlr.press/v202/frantar23a/frantar23a.pdf.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.17323.

Elias Frantar, Roberto L. Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. Marlin: Mixed-precision auto-regressive parallel inference on large language models. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, 2025. URL https://doi.org/10.1145/3710848.3710871.

Zhongqian Fu, Tianyi Zhao, Ning Ding, Xianzhi Yu, Xiaosong Li, Yehui Tang, and Yunhe Wang. EAQuant: Enhancing post-training quantization for MoE models via expert-aware optimization. *arXiv preprint arXiv:2506.13329*, 2025. URL https://arxiv.org/abs/2506.13329.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2506.04178.

Hang Guo, Luca Benini, and Yawei Li. Optimal brain restoration for joint quantization and sparsification of LLMs. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=VQIvBpL5ag.

Jinyang Guo, Jianyu Wu, Zining Wang, Jiaheng Liu, Ge Yang, Yifu Ding, Ruihao Gong, Haotong Qin, and Xianglong Liu. Compressing large language models by joint sparsification and quantization. In *International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/guo24g.html.

Simla Harma, Ayan Chakraborty, Elizaveta Kostenok, Danila Mishin, Dongho Ha, Babak Falsafi, Martin Jaggi, Ming Liu, Yunho Oh, Suvinay Subramanian, et al. Effective interplay between sparsity and quantization: From theory to practice. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2405.20935.

Babak Hejazi. Introducing grouped GEMM APIs in cuBLAS and more performance updates. NVIDIA Technical Blog, June 2024. URL https://developer.nvidia.com/blog/introducing-grouped-gemm-apis-in-cublas-and-more-performance-updates/. Published June 12, 2024; accessed 2026-09-22.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks Track*, 2021. URL https://arxiv.org/abs/2103.03874.

Wei Huang, Yue Liao, Jianhui Liu, Ruifei He, Haoru Tan, Shiming Zhang, Hongsheng Li, Si Liu, and Xiaojuan Qi. Mixture compressor for mixture-of-experts LLMs gains more. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.06270.

Hugging Face. Open LLM leaderboard v1. Hugging Face Leaderboards documentation, 2024. URL https://huggingface.co/docs/leaderboards/en/open_llm_leaderboard/archive.

inclusionAI. Humming: A JIT-compiled GEMM kernel library for quantized inference. https://github.com/inclusionAI/humming, 2026.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1611.01144.

Minjun Kim, Jaehyeon Choi, Hyunwoo Yang, Jongjin Kim, Jinho Song, and U Kang. Prune-then-quantize or quantize-then-prune? Understanding the impact of compression order in joint model compression. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2603.18426.

Kimi Team. Kimi K2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026. URL https://arxiv.org/abs/2602.02276.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023. URL https://arxiv.org/abs/2309.06180.

Mike Lasby, Ivan Lazarevich, Nish Sinnadurai, Sean Lie, Yani Ioannou, and Vithursan Thangarasa. Reap the experts: Why pruning prevails for one-shot moe compression. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/ed93b2b5722acc2341d421b8916404a1-Paper-Conference.pdf.

Kwanhee Lee, Hyeondo Jang, Dongyeop Lee, Dan Alistarh, and Namhoon Lee. The unseen frontier: Pushing the limits of LLM sparsity with surrogate-free ADMM. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2510.01650.

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. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

LG AI Research. K-EXAONE 2.0. Hugging Face model card, 2026. URL https://huggingface.co/LGAI-EXAONE/K-EXAONE-2.0-750B-A37B.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for LLM compression and acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024. URL https://arxiv.org/abs/2306.00978.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Ryan Lucas, Kayhan Behdin, Zhipeng Wang, Qingquan Song, Shao Tang, and Rahul Mazumder. Reasoning models can be accurately pruned via chain-of-thought reconstruction. In *International Conference on Learning Representations (ICLR)*, 2026. doi: 10.48550/arXiv.2509.12464. URL https://arxiv.org/abs/2509.12464.

Jan Ludziejewski, Jakub Krajewski, Kamil Adamczewski, Maciej Pióro, Michal Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygózdz, Piotr Sankowski, et al. Scaling laws for fine-grained mixture of experts. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.07871.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1611.00712.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021. URL https://arxiv.org/abs/2104.08378.

Moonshot AI. Kimi K3 tech blog: Open frontier intelligence. Moonshot AI blog, 2026. URL https://www.kimi.com/blog/kimi-k3.

Taishi Nakamura, Satoki Ishikawa, Masaki Kawamura, Takumi Okamoto, Daisuke Nohara, Jun Suzuki, and Rio Yokota. Optimal sparsity of mixture-of-experts language models for reasoning tasks. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2508.18672.

Neural Magic. LLM Compression Calibration. Hugging Face dataset, June 2024. URL https://huggingface.co/datasets/neuralmagic/LLM_compression_calibration.

NVIDIA. Pretraining large language models with NVFP4. *arXiv preprint arXiv:2509.25149*, 2025. URL https://arxiv.org/abs/2509.25149.

NVIDIA Corporation. NVIDIA A100 tensor core GPU: Unprecedented acceleration at every scale. NVIDIA datasheet, June 2021. URL https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf.

NVIDIA Corporation. NVIDIA H100 tensor core GPU datasheet. NVIDIA datasheet, 2022. URL https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306.

NVIDIA Corporation. NVIDIA DGX B200 datasheet. NVIDIA datasheet, 2024. URL https://resources.nvidia.com/en-us-dgx-systems/dgx-b200-datasheet.

NVIDIA Corporation. CUTLASS functionality on NVIDIA Blackwell architecture. NVIDIA developer documentation, 2025a. URL https://docs.nvidia.com/cutlass/latest/media/docs/cpp/blackwell_functionality.html.

NVIDIA Corporation. Blackwell Grouped GEMM with Block-Scaled Narrow-Precision Inputs. NVIDIA CUTLASS example 75, 2025b. URL https://github.com/NVIDIA/cutlass/blob/main/examples/75_blackwell_grouped_gemm/75_blackwell_grouped_gemm_block_scaled.cu.

NVIDIA Corporation. Qwen3-30B-A3B-NVFP4. Hugging Face model card, 2025c. URL https://huggingface.co/nvidia/Qwen3-30B-A3B-NVFP4.

NVIDIA Corporation. Kimi-K2.5-NVFP4. Hugging Face model card, 2026a. URL https://huggingface.co/nvidia/Kimi-K2.5-NVFP4.

NVIDIA Corporation. Qwen3.5-397B-A17B-NVFP4. Hugging Face model card, 2026b. URL https://huggingface.co/nvidia/Qwen3.5-397B-A17B-NVFP4.

Open Compute Project Foundation (MX Alliance). OCP microscaling formats (MX) specification, version 1.0. Open Compute Project specification, 2023. URL https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf.

Qwen Team. Qwen3.5-397B-A17B. Hugging Face model card, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-397B-A17B.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, et al. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, 2023. URL https://arxiv.org/abs/2310.10537. OCP Microscaling (MX) format specification.

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 (ICLR)*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=PxoFut3dWW.

Lintang Sutawika, Hailey Schoelkopf, Leo Gao, Baber Abbasi, Stella Biderman, Jonathan Tow, ben fattori, Charles Lovering, farzanehnakhaee70, Jason Phang, Anish Thite, Fazz, Aflah, Niklas Muennighoff, Thomas Wang, sdtblck, nopperl, gakada, tttyuntian, researcher2, Julen Etxaniz, Chris, Hanwool Albert Lee, Zdeněk Kasner, Khalid, LSinev, Jeffrey Hsu, Anjor Kanekar, KonradSzafer, and AndyZwei. Eleutherai/lm-evaluation-harness: v0.4.3, July 2024. URL https://doi.org/10.5281/zenodo.12608602.

Albert Tseng, Qingyao Sun, David Hou, and Christopher De Sa. QTIP: Quantization with trellises and incoherence processing. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2406.11235. Spotlight.

vLLM Project. Kimi-K2.5 deployment recipe. vLLM model recipe, 2026. URL https://github.com/vllm-project/recipes/blob/main/models/moonshotai/Kimi-K2.5.yaml.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025. URL https://arxiv.org/abs/2501.01005.

Z.ai. GLM-5.2. Hugging Face model card, 2026. URL https://huggingface.co/zai-org/GLM-5.2.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

A Sparse-Quantized Representation

Feasible set. Partition each row of W into sparsity groups $W_{[r]} \in \mathbb{R}^m$ and scaling blocks $W_{[b]} \in \mathbb{R}^d$, and let the kernel prescribe the sparsity patterns $\mathcal{P} \subset \{0,1\}^m$, value grid $\mathcal{G}_w$, and scale grid $\mathcal{G}_s$. The target feasible set in Equation (1) is

$$\begin{aligned} \mathcal{S} &= \Big\{ W : \forall r,\ \exists \rho \in \mathcal{P} \text{ s.t. } W_{[r]} \odot (\mathbf{1} - \rho) = \mathbf{0} \Big\}, \\ \mathcal{Q} &= \Big\{ W : \forall b,\ \exists s_b \in \mathcal{G}_s \text{ s.t. } W_{[b]} \in \big(s_b \mathcal{G}_w\big)^d \Big\}, \\ \mathcal{C} &= \mathcal{S} \cap \mathcal{Q}. \end{aligned} \tag{6}$$

Because $0 \in \mathcal{G}_w$, pruned positions satisfy the quantization constraint, and each shared scale governs the retained values of its block. For Blackwell paired 4:8 sparse NVFP4, $m = 8$, $\mathcal{P}$ contains the six patterns that retain two adjacent pairs, $\mathcal{G}_w$ is the E2M1 grid, $\mathcal{G}_s$ contains UE4M3 scales up to a per-matrix constant, and $d = 32$. Figure 3 illustrates this instantiation.

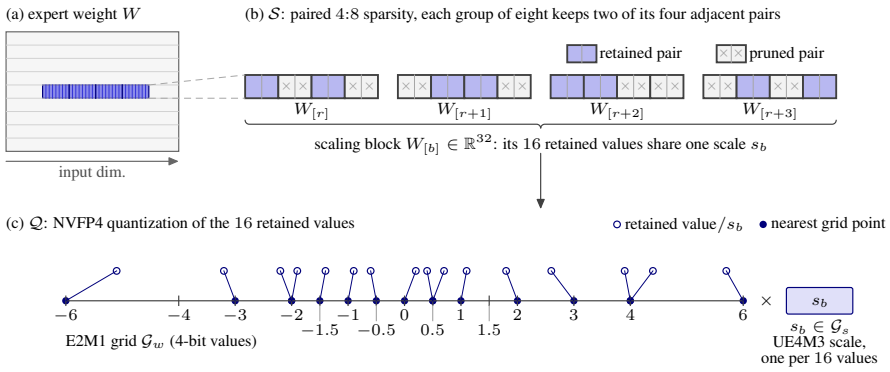

Figure 3: Equation (6) instantiated for the Blackwell sparse NVFP4 instruction ($m = 8$, $d = 32$). (a) An expert weight matrix; 32 consecutive weights along the input dimension are magnified. (b) Each group of eight retains two of its four adjacent pairs, one of the six paired 4:8 patterns in $\mathcal{P}$. (c) The 16 retained values of a block lie on the E2M1 grid $\mathcal{G}_w = \{0, \pm 0.5, \pm 1, \pm 1.5, \pm 2, \pm 3, \pm 4, \pm 6\}$ and share one UE4M3 (unsigned FP8) scale $s_b \in \mathcal{G}_s$.

Storage layout. The packed representation stores, per sparsity group of $m = 8$ weights, the four retained E2M1 values and the index of the retained pattern as sparse metadata, and one UE4M3 scale

per block of $d = 32$ weights:

$$\underbrace{4 \cdot \tfrac{4}{8}}_{\text{E2M1 values}} + \underbrace{\tfrac{4}{8}}_{\text{sparse metadata}} + \underbrace{\tfrac{8}{32}}_{\text{UE4M3 scale}} = 2.75 \text{ bits per weight.} \tag{7}$$

B Experimental Setup

Compression scope. Our method and the sparse-quantized baselines compress only the routed expert FFN weights; attention, shared experts, routers, embeddings, and the LM head remain at the reference checkpoint precision. The target representation is paired 4:8 sparsity with NVFP4 weights and activations (W4A4), using 32-element scale blocks as required by the Blackwell sparse NVFP4 path. Details of the feasible set and storage layout are given in Appendix A.

Baselines. For sparse-quantized reconstruction, we adapt the official implementations of JSQ (Guo et al., 2024), SGPTQ (Frantar & Alistarh, 2023; Frantar et al., 2023), and OBR (Guo et al., 2026) to MoE experts under the same paired 4:8 NVFP4 W4A4 target. SGPTQ transfers directly, applying SparseGPT support selection followed by GPTQ with the support fixed. The SGPTQ comparison is not compute-matched to **MoESQ**: SGPTQ uses 512 calibration sequences for its one-shot pass, while **MoESQ** starts from the resulting SGPTQ solution and performs a further 10 refinement epochs over 8,192 sequences. We report this difference explicitly because the comparison isolates the solution quality reached by the two optimization recipes, not their quality at equal compression cost. For OBR, we omit the rotation step because the corresponding rotated experts cannot be executed by the grouped sparse GEMM path evaluated here; we refer to this executable adaptation simply as OBR throughout the empirical evaluation. For the broader expert-compression comparison, GSQ (Dadgarnia et al., 2026) uses dense INT2 W2A16 weights, and REAP (Lasby et al., 2026) prunes 50% of routed experts while retaining the released checkpoint's INT4 weights. These methods are compared jointly in Section 4.4.

Calibration data. All compression stages use a single calibration mixture combining LLM compression calibration (Neural Magic, 2024), OpenThoughts-114k (Guha et al., 2026), and FineWeb-Edu (Lozhkov et al., 2024) in a 10:45:45 ratio. The mixture is chosen to cover a broad range of data, spanning curated calibration text, long-form reasoning traces, and filtered web text. Documents are packed into sequences of 4,096 tokens. SGPTQ initialization draws 512 sequences, and support–value refinement draws 8,192 sequences, with a further 128 held out for validation. The same mixture is used for every model and for every reported arm.

Compression hyperparameters. Table 4 gives the per-model optimization configuration of the router-weighted objective used for the main results.

Table 4: Per-model compression hyperparameters. Learning rates are for the support logits A and latent weights V; the Gumbel temperature τ_t of Equation (2) is annealed over the schedule shown.

	Kimi-K2.5	Qwen3.5-397B-A17B	Qwen3-30B-A3B
MoE blocks compressed	60	60	48
Routed experts per block	384	512	128
Calibration sequences	8,192	8,192	8,192
Batch size (micro-batch)	64 (8)	64 (8)	64 (16)
Epochs per block	10	10	10
Optimizer		Lion (Chen et al., 2023)	
Support-logit learning rate (A)	2×10^{-4}	2×10^{-4}	2.5×10^{-4}
Latent-weight learning rate (V)	5×10^{-5}	2.5×10^{-5}	5×10^{-5}
Gumbel temperature τ_t		$2.0 \rightarrow 0.05$	
SGPTQ damping (percdamp)	0.1	0.1	0.01
Devices	8×B200	8×B200	4×B200
Wall-clock	≈27.6 h	≈18.9 h	≈13 h

Evaluation protocol. All accuracy evaluations are served through vLLM with the KV-cache dtype fixed within each model family. We evaluate OpenLLM Leaderboard v1 (Hugging Face, 2024) with `lm-eval-harness` (Sutawika et al., 2024) on all three models. For reasoning, we evaluate Kimi-K2.5 on AIME25 (Zhang & Math-AI, 2025), GPQA Diamond (Rein et al., 2024), and MATH500 (Hendrycks et al., 2021) following the GSQ protocol (Dadgarnia et al., 2026). Pass@1 is averaged over 10, 5, and 5 runs, respectively, with temperature 1.0, top-p 0.95, and a 65,536-token generation budget.

C Additional Accuracy Results

C.1 Ablations

SGPTQ is a one-shot procedure and does not provide an iterative stage to which additional optimization budget can be allocated, whereas our differentiable formulation enables its sparse-quantized solution to serve as the initialization for further refinement at a higher compression cost. We hold this added budget fixed and ablate the two design choices enabled by the refinement stage on Qwen3-30B-A3B: the router-weighted objective and joint support–value adaptation. All arms use the same SGPTQ initialization, calibration setup, and 10-epoch refinement budget, and the full **MoESQ** configuration is the Qwen3-30B-A3B column of Table 4, with per-task results in Table 5.

Router-weighted reconstruction objective. Replacing the uniform special case $w_{l,e} = \mathbf{1}$ with the router-weighted objective of Equation (4) increases the OpenLLM average from 64.68 to 65.44 and recovery from 88.26% to 89.30%, with improvements on all six tasks. We tune the learning rates separately for the uniform and router-weighted objectives rather than forcing matched rates across objectives with different gradient scales. The selected support-logit and latent-weight learning rates are 5×10^{-4} and 1×10^{-4} for the uniform objective, and 2.5×10^{-4} and 5×10^{-5} for the router-weighted objective, respectively.

Support–value adaptation. With the initial SGPTQ support fixed, adapting only the latent weights V reaches 64.44 average accuracy, whereas jointly adapting the support logits A and latent weights V under the router-weighted objective reaches 65.44, an improvement of 1.00 percentage point.

Table 5: Ablations on Qwen3-30B-A3B under the paired 4:8 NVFP4 W4A4 target. $\mathcal{L}^{\text{unif}}$ denotes Equation (4) with $w_{l,e} = \mathbf{1}$, and "V only" keeps the initial SGPTQ support fixed. Avg. is the unweighted mean across the six OpenLLM tasks, Rec. is relative to the BF16 reference in Table 1, and bold marks the best result in each column.

Objective	Updated	ARC-c	GSM8K	MMLU	WG	HSwag	TQA	Avg	Rec.
$\mathcal{L}^{\text{unif}}$	(A, V)	59.30	77.18	69.84	66.93	63.25	51.55	64.68	88.26%
$\mathcal{L}^{\text{rw}}$	V only	57.42	79.15	**70.18**	66.38	63.41	50.08	64.44	87.95%
$\mathcal{L}^{\text{rw}}$ (**MoESQ**)	(A, V)	**59.73**	**79.45**	69.89	**67.80**	**63.60**	**52.15**	**65.44**	**89.30%**

C.2 Per-Task Expert-Compression Comparison

This section provides the per-task accuracy results underlying the expert-compression comparison of Section 4.4 and Table 3. Unlike Tables 1 and 2, which compare methods under the same sparse W4A4 target, the methods here use different compression mechanisms and deployment formats.

C.3 Non-Terminating Generations under JSQ

The reasoning degradation of JSQ in Table 2 is driven largely by non-terminating generations rather than by ordinary answer errors. Under the same paired 4:8 NVFP4 W4A4 target, JSQ often enters increasingly repetitive reasoning trajectories that continue until the 65,536-token budget is exhausted, whereas OBR and **MoESQ** typically remain substantially more concise and terminate normally. Figure 4 illustrates this failure mode on representative prompts.

Table 6: OpenLLM Leaderboard v1 comparison with alternative expert-compression methods on Kimi-K2.5. GSQ uses dense INT2 W2A16 weights, REAP prunes 50% of routed experts while retaining the released INT4 weights, and **MoESQ** uses paired 4:8 NVFP4 W4A4. Recovery is relative to the reference row, and bold marks the best compressed result.

Method	Expert format	ARC-c	GSM8K	MMLU	WG	HSwag	TQA	Avg	Rec.
Reference	Dense INT4 W4A16	74.06	94.39	89.53	81.93	91.96	62.69	82.43	100%
GSQ	Dense INT2 W2A16	64.51	91.96	84.35	80.90	86.01	58.24	77.66	94.22%
REAP	50% expert pruning + INT4	**72.35**	**93.71**	**87.72**	82.00	**89.25**	59.60	**80.77**	**97.99%**
Ours	Paired 4:8 NVFP4 W4A4	69.97	88.70	85.34	**82.40**	88.57	**60.28**	79.21	96.09%

(a) MATH500, problem 457. *"Determine the remainder of 54 (mod 6)."*

MoESQ (281 characters, terminated, correct)

```
To determine the remainder of 54 when divided by 6, we can perform the division:

$$54 \div 6 = 9 \text{ with remainder } 0$$

This means:
$$54 = 6 \times 9 + 0$$

Therefore, the remainder is 0.

ANSWER: $0$
```

JSQ (260,474 characters, unterminated; opening, then the final 260 characters)

```
We need to determine the remainder of 54 mod 6, which means the remainder when 54
is divided by 6.

Recall that "mod 6" means dividing by 6 and finding the remainder. The remainder
when dividing by 6 is the leftover after the integer division.

Let's perform the division: [...]

The remainder is 0 because 54 is exactly divisible by 6 [...]

 the the the the the the the the the the the the the the the the the the the the
the the the the the the the the the the the the the the the the the the the the
the the the the the the the the the the the the the the the the the the the the
```

(b) AIME25, problem 14. All three methods exhaust the budget on this problem; final characters of each generation:

MoESQ (174,615 characters, unterminated)

```
However, the uniform distribution assumption is likely correct because H is a
"random" subset in the additive sense, or due to the group action.

Given the symmetry, it is plausible that the number of pairs summing to any
particular d depends only on the valuation of d.

Thus, for d with v_3 = 3, N
```

OBR (175,294 characters, unterminated)

```
Alternatively, note that H is the set of cubes. [...] Then the sum is
```

JSQ (194,373 characters, unterminated)

```
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
```

Figure 4: Representative reasoning trajectories under the same paired 4:8 NVFP4 W4A4 target. In (a), **MoESQ** terminates with the correct answer, whereas JSQ reaches the same answer early but continues into a degenerate repetitive tail until the generation budget is exhausted. In (b), all methods exhaust the budget on the same difficult prompt, but **MoESQ** and OBR remain in coherent reasoning while JSQ has collapsed into repetition. Text is verbatim from the evaluation records, with `[...]` marking elisions and line breaks inserted for readability.

To quantify the effect, Table 7 separates answer quality from termination behavior. Accuracy on terminated generations measures how often a method answers correctly once it does terminate, while

the matched-subset comparison restricts all methods to problems for which JSQ terminates at least once. This avoids attributing the full gap solely to the generation cutoff.

Table 7: Termination behavior of sparse-quantized Kimi-K2.5 models. Accuracy on terminated generations isolates answer quality after successful termination, while the matched-subset comparison controls for JSQ terminating on substantially fewer problems. Coverage is the number of problems for which JSQ terminates at least once out of the benchmark total.

	Accuracy on terminated generations			Matched subset				Median terminated (chars)	
Benchmark	Ours	OBR	JSQ	Coverage	Ours	OBR	JSQ	Ours	JSQ
AIME25	90.39	84.85	62.86	11/30	100.00	100.00	20.00	42,580	83,504
GPQA Diamond	76.70	72.35	48.01	123/198	75.12	71.06	21.63	43,810	105,104
MATH500	96.63	95.21	84.12	379/500	96.89	95.41	36.62	5,426	34,486

The qualitative and quantitative results tell the same story. JSQ is heavily truncated on all three benchmarks, and even among generations that terminate, its accuracy remains substantially below OBR and MoESQ. Restricting evaluation to the matched subset does not close the gap. Thus, under this aggressive sparse-quantized target, poor compression can degrade long-horizon generation dynamics, producing rambling or repetitive trajectories that fail to terminate rather than simply increasing the frequency of conventional wrong answers (Lucas et al., 2026). We restrict this observation to the evaluated JSQ adaptation and do not claim that non-termination is universal to JSQ or sparse quantization in general.

D Grouped Sparse GEMM Kernel and Serving

D.1 Implementation Details

Operand layout and scheduling. Weights are packed offline into E2M1 values, sparse metadata, and one UE4M3 scale per block of 32 logical weights, as detailed in Appendix A. Activations are quantized at run time into the corresponding 32-element scale blocks by $\mathcal{Q}_a$ of Equation (4). We store activations and outputs in a batched layout with a fixed per-expert capacity $\bar{N}$, so every operand of expert e lies at a fixed stride and no pointer arrays are needed. The token counts remain on the device, and the persistent scheduler assigns expert e exactly $\lceil N_e/T_N \rceil$ tiles along the token dimension; only its last partial tile can touch padded rows, whose outputs are discarded downstream.

Tile configurations. The MMA tile is $T_M \times T_N \times 256$, with $T_M \in \{128, 256\}$ in one- or two-SM mode and $T_N \in \{128, 256\}$ (NVIDIA Corporation, 2025a). The UE4M3 block scales of both operands are applied inside the MMA, while their per-tensor scales are folded into one factor α_e per expert that the epilogue applies to the FP32 accumulator before writing BF16. The four tile shapes and admissible cluster shapes produce 30 kernel variants. At startup, the serving backend times these variants once per expert GEMM geometry on the loaded weights and freezes the selected variant before CUDA-graph capture.

Serving integration. We expose the kernel to vLLM (Kwon et al., 2023) through the same fused-MoE interface as the dense NVFP4 backend built on FlashInfer (Ye et al., 2025). The sparse backend implements fused dispatch–scatter with activation quantization, the two grouped GEMMs, and fused SwiGLU with requantization, while routing, dispatch planning, batched preparation and finalization, and token combination remain shared. Unlike the dense NVFP4 path, which uses 16-element scale blocks, our activation-quantization kernel forms 32-element blocks and emits the scale layout required by the sparse MMA. We fuse this quantization with token dispatch and scatter for the first expert GEMM and with SwiGLU output preparation for the second.

D.2 Kernel Benchmark

We benchmark our grouped sparse NVFP4 GEMM on a single B200 against the CUTLASS dense NVFP4 grouped GEMM, which we run in two operand layouts. Table 8 summarizes the layer shapes and sweep ranges. Per-expert token counts follow the deterministic imbalanced pattern $\{N, 3N/4, N/2, N/4\}$ repeated across groups.

Operand layout. Section 3.4 describes the operand orientation of the sparse kernel. The dense baseline, Dense (native), keeps the activation-stationary layout used by vLLM; the speedups in Figure 1(a) and Figure 5 are relative to it. The dense control, Dense (transposed), runs the same kernel in the orientation of the sparse kernel, so that it differs from our kernel only in sparsity.

Table 8: Grouped-GEMM benchmark shapes. (M, K) are the output and contraction dimensions; N is the maximum per-expert token count and G the expert-group count.

Model	Layer	(M, K)	N sweep	G sweep
Kimi-K2.5	gate/up down	$(4096, 7168)$ $(7168, 2048)$	$\{32, 64, 128, 256, 512, 1024\}$	$\{24, 48, 96, 192, 384\}$
Qwen3-30B-A3B	gate/up down	$(1536, 2048)$ $(2048, 768)$	$\{16, 32, 64, 128, 256, 512, 1024\}$	$\{8, 16, 32, 64, 128\}$

Timing and autotuning. We time pure on-device kernel execution with CUDA-graph replay following QUTLASS (Castro & Alistarh, 2025); allocation, operand preparation, sparse compression, and kernel initialization are excluded. We report median replay latency. Each problem/kernel pair is autotuned independently over SM mode, MMA tile shape, and Blackwell cluster configuration (Table 9), and the fastest valid variant is selected. Effective throughput and speedup are

$$\mathrm{TFLOP/s}_{\mathrm{eff}} = \frac{2MK\sum_{g=1}^{G} N_g}{t \cdot 10^{12}}, \qquad \mathrm{speedup} = \frac{t_{\mathrm{dense}}}{t_{\mathrm{sparse}}}.$$

Table 9: Autotuner search space. Tile-M is fixed by the SM mode and tile-K=256 throughout; supported tile-N values differ across kernels.

Kernel	SM mode (tile-M)	tile-N	tile-K
Sparse (ours)	1-SM (128), 2-SM (256)	$\{128, 256\}$	256
Dense (native)	1-SM (128), 2-SM (256)	$\{128, 192, 256\}$	256
Dense (transposed)	1-SM (128), 2-SM (256)	$\{128, 192\}$	256

Speedup across shapes. Across the full $G \times N$ sweep, the sparse kernel is faster in every configuration. On Kimi-K2.5, geometric-mean speedups are $1.52\times$ for gate/up and $1.58\times$ for down; on Qwen3-30B-A3B they are $1.33\times$ and $1.20\times$, respectively. The gain depends primarily on per-expert token count and contraction size: it is smallest for short, memory-dominated problems and approaches the sparse-MMA advantage as arithmetic intensity increases. Figure 5 gives the full sweep.

Energy measurement. We measure whole-board energy with NVML over sustained CUDA-graph replay on an otherwise idle, exclusive B300, using 200 warmup iterations and a pinned SM clock for the iso-frequency comparison. We report energy per call and average board power. Because the sparse kernel draws more instantaneous power but finishes sooner, energy-to-solution is the relevant metric; Table 10 reports both iso-frequency and power-capped measurements.

Cross-kernel comparison. Figure 1(c) compares the deployment kernels used in Figure 2 on the Kimi-K2.5 gate/up layer at G=384, spanning decode-like to prefill-like token counts. Timings include activation quantization for the W4A4 kernels and dequantization for the W4A16/W2A16 kernels, from BF16 input activations to BF16 output. Our sparse kernel is fastest across the full range, including against INT4 Marlin and INT2 Humming; Table 11 reports the measured latencies.

D.3 End-to-End Serving Benchmark

We serve Kimi-K2.5 on $8\times$B200 with vLLM under a shared DP=8 $\times$ EP topology using DeepEP low-latency all-to-all. Across all backends, routing, dispatch planning, batched preparation and finalization, batching, memory controls, token combination, and output layout are held fixed; only the expert representation and its associated compute path differ. The number of concurrent requests

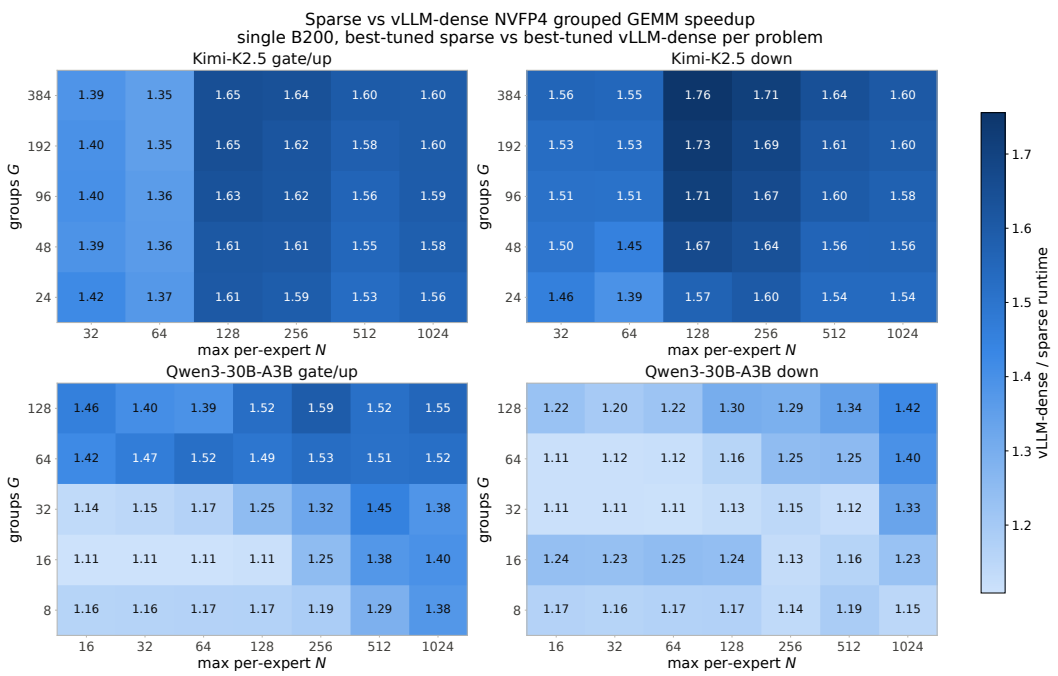

Figure 5: Autotuned speedup of the grouped sparse GEMM over the dense NVFP4 grouped GEMM (native layout) across the four benchmarked MoE layers on B200. Rows are expert-group count G and columns are maximum per-expert token count N.

Table 10: Energy-to-solution for the Kimi-K2.5 gate/up grouped GEMM (M=4096, N=128, K=7168, G=384) on B300. Iso-frequency is the primary comparison; the power-capped regime is included for completeness. Dense columns are the two operand layouts of the same CUTLASS kernel (Appendix D.2); speedup and energy reduction are those of the sparse kernel over each dense column.

	Iso-frequency (1702 MHz)			Power-capped (≈ 1095 W)		
	Sparse	Dense (transposed)	Dense (native)	Sparse	Dense (transposed)	Dense (native)
Avg. power (W)	1063	865	945	1095	1096	1092
SM clock (MHz)	1702	1702	1702	1725	1897	1867
Latency (ms)	0.687	1.348	1.164	0.679	1.183	1.131
Throughput (TFLOP/s)	2627	1339	1550	2657	1525	1595
Energy/call (mJ)	727	1166	1098	745	1298	1237
Efficiency (TFLOP/J)	2.48	1.55	1.64	2.42	1.39	1.46
Speedup	–	1.96×	1.69×	–	1.74×	1.67×
Energy reduction	–	1.60×	1.51×	–	1.74×	1.66×

is pinned with `--max-concurrency`, so every backend is measured at matched concurrency. Table 12a gives the shared configuration and Table 12b the full results corresponding to Figure 2.

Choice of comparators. We compare our grouped sparse GEMM (paired 4:8 NVFP4 W4A4) with dense NVFP4 W4A4 CuTeDSL, INT4 W4A16 Marlin, and INT2 W2A16 Humming. We use the batched expert variants of all kernels so that they share the same dispatch path and output layout.

Workloads. We report two workloads, each of which isolates one serving phase, and every measurement issues 500 requests. The prefill workload uses 4096 input tokens and 1 output token per request at concurrencies of 32, 64, and 128, where concurrency denotes the number of concurrent

Table 11: Cross-kernel grouped-GEMM latency (μs) on the Kimi-K2.5 gate/up layer (M=4096, K=7168, G=384) on B200. Timings include activation quantization/dequantization and use the same ragged token pattern as the main sweep.

	bits/p	N=8	N=16	N=32	N=64	N=128	N=256
Ours (paired 4:8 W4A4)	2.750	**760**	**793**	**843**	**925**	**912**	**1,370**
NVFP4-W4A4 CuTeDSL	4.500	979	1,012	1,036	1,100	1,226	1,717
INT4-W4A16 Marlin	4.500	1,584	1,805	2,519	3,717	6,163	11,272
INT2-W2A16 Humming	2.125	1,538	1,545	2,959	3,299	6,446	9,733

requests summed over the eight DP engines. Since each request emits a single token, we report total token throughput for this workload. The decode workload uses 384 input and 128 output tokens per request with a concurrency cap of at least 500, so that all 500 requests are in flight. At 512 tokens per request, this requires 256K KV-cache tokens, well below the capacity of every backend in Table 12b, so no backend is limited by KV-cache capacity. The decode workload is measured three consecutive times within the same session, with concurrency caps of 512, 1024, and 2048, all of which exceed the number of issued requests and therefore yield the same operating point. Throughput varies by up to 22% across these repeats, generally increasing as the server warms up, so we report the mean of the three repeats together with their range. The first measurement of each session is excluded for all backends, since it absorbs server warmup.

Table 12: End-to-end serving on Kimi-K2.5 (8× B200, matched concurrency); the results of Figure 2 in full.

(a) Serving configuration, shared by all four backends; only the expert kernel and weight format differ.

Hardware	8× NVIDIA B200
Engine	vLLM, DP=8 × EP, DeepEP low-latency all-to-all
Fixed controls	gpu-mem-util 0.97, max-model-len 4608, max-batched-tokens 256
KV cache dtype	`bfloat16`, pinned for all backends
Prefill workload	4096 in / 1 out, concurrency ∈ {32, 64, 128}, 500 requests
Decode workload	384 in / 128 out, all 500 requests in flight, 3 repeats

(b) Full serving results corresponding to Figure 2. Best per row in **bold**. Prefill rows are measured at each concurrency of the prefill workload. Decode and latency rows are the mean over three repeats of the decode workload with all 500 requests in flight, and the range row gives the minimum and maximum decode throughput across the repeats. *Maximum concurrency is reported by vLLM at engine start relative to a full-length request. †Prefill emits one token per request, so prefill rows report total token throughput; decode rows report output token throughput.

	Ours (Paired 4:8 W4A4)	NVFP4-W4A4 CuTeDSL	INT4-W4A16 Marlin	INT2-W2A16 Humming
Throughput (↑)				
Prefill, concurrency 32 (tok/s)†	**44,223**	37,314	12,864	19,430
Prefill, concurrency 64 (tok/s)†	**49,184**	41,802	14,615	21,120
Prefill, concurrency 128 (tok/s)†	**55,654**	44,678	16,715	24,116
Decode (tok/s)	**8,983**	7,619	2,108	3,811
Decode, range (tok/s)	8,494–9,512	7,213–7,831	1,843–2,252	3,586–3,985
Latency, decode workload (↓)				
End-to-end mean (s)	**6.2**	7.3	25.0	14.6
End-to-end p99 (s)	**7.0**	8.3	30.1	16.6
TPOT mean (ms)	**32.2**	38.1	139.3	79.0
TPOT median (ms)	**32.8**	38.8	140.3	80.2
TTFT mean (ms)	**2,120**	2,505	7,306	4,560
Capacity (↑)				
KV cache (tokens)	1,432,608	1,098,720	1,054,848	**1,593,760**
Max concurrency*	310.9×	238.4×	228.9×	**345.9**×