HETEROLORA-MOE: HETEROGENEOUS LOW-RANK EXPERTS WITH DIVERSE PROJECTION PATTERNS AND EFFICIENT GPU KERNELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Expert differences in a pretrained mixture-of-experts (MoE) model exhibit low-rank structure, motivating compact expert representations based on shared weights and low-rank updates. Motivated further by variation in this structure across projections and experts, we present HeteroLoRA-MoE, which distributes expert-specific low-rank updates across four balanced projection patterns over shared dense SwiGLU weights. A single token-level router selects complete experts and combines their outputs. This compact design introduces small matrix products and projection-dependent scheduling, so parameter savings do not automatically translate into lower latency. To address these execution costs, we develop HeteroKernel, a structure-aware GPU implementation that reuses shared gate and up results and aggregates expert intermediates before applying the shared dense down projection. Projection-aware grouping, fused reductions, and pipelined weight loading organize the remaining adapter computation, skipping absent branches while preserving the mixture algebraically. In four-seed mixed-SFT experiments initialized from Qwen2.5-0.5B/1.5B-Instruct, HeteroLoRA-MoE achieves higher mean scores than a parameter-matched MixLoRA-style baseline on multiple evaluation tasks at both model scales. On NVIDIA L20, HeteroKernel reduces the overhead of routed low-rank computation, achieving 1.85–$6.79\times$ speedups over the PyTorch implementation across the tested prefill and one-token decode settings.

1 INTRODUCTION

Mixture-of-experts (MoE) architectures are widely used to scale neural models through conditional computation (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). By routing each token to a small subset of experts, they expand model capacity without activating all parameters for every input. This design supports applications ranging from multilingual translation to language modeling, mathematical reasoning, and code generation (Lepikhin et al., 2021; Jiang et al., 2024). Models such as Mixtral and DeepSeekMoE demonstrate the value of combining a large expert pool with sparse activation (Jiang et al., 2024; Dai et al., 2024). How these experts are represented determines how efficiently the model uses its parameter budget.

The differences among experts contain low-rank structure that makes a fully independent parameterization potentially redundant. Our analysis of Phi-tiny-MoE-instruct (Li et al., 2025) finds structured low-rank components in expert weight differences relative to an average reference. On held-out routed activations, activation-aware rank-32 approximations retain median residual output energies of 63.02%, 56.20%, and 47.89% for the gate, up, and down projections, respectively. In comparison, random rank-32 approximations retain only 7.15%, 7.15%, and 7.37%, respectively, showing that selected low-dimensional subspaces capture much more of the expert differences than random ones. Their relative concentration across projections also varies among experts. Storing a complete feed-forward network (FFN) for each expert therefore misses the opportunity to share common weights and encode structured differences compactly. Sparse activation reduces the number of experts evaluated per token, but does not remove this parameter redundancy.

Existing methods reduce redundancy through weight sharing and compact expert updates. LoRA represents adaptation with low-rank factors (Hu et al., 2022), routed LoRA mixtures construct

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.

multiple experts over a shared backbone (Dou et al., 2024; Li et al., 2024), and parameter-efficient expert representations retain common weights while reducing expert-specific storage (Wu et al., 2024; Huang et al., 2025). A homogeneous low-rank mixture, however, gives every expert the same adaptation sites, even when the observed structure of expert differences varies across projections. This restriction matters in SwiGLU because gate and up updates modify the nonlinear intermediate activation, while down updates act on its output mapping. Heterogeneous adapter families and module-wise expert and rank allocation offer other forms of flexibility (Cao et al., 2026; Chen et al., 2025). These design choices do not directly express coordinated projection combinations within each routed expert. What remains is to represent expert diversity through both the low-rank factors and the locations at which they modify the complete FFN transformation.

We introduce HeteroLoRA-MoE, a heterogeneous low-rank MoE architecture that shares dense SwiGLU weights and forms experts through different combinations of low-rank updates to the gate, up, and down projections. A single token-level router selects complete experts and combines their outputs. Experts therefore differ in both their learned updates and their adaptation locations. This compact structure introduces small matrix products and projection-dependent scheduling, so parameter savings do not automatically translate into faster execution. We address these costs with HeteroKernel, a structure-aware GPU implementation that reuses shared computations, aggregates expert intermediates before a single shared down projection, and organizes the remaining adapter operations through grouping, fusion, and pipelined loading. Our contributions are summarized below.

- We propose HeteroLoRA-MoE, which uses projection combinations to construct heterogeneous low-rank experts over shared dense weights. This reduces expert parameter storage while allowing structurally diverse adaptation paths.
- We develop HeteroKernel, tailored to projection-heterogeneous low-rank experts, reducing redundant computation, scheduling overhead, and intermediate-memory traffic.
- Across four seeds, HeteroLoRA-MoE achieves higher mean scores than the parameter-matched MixLoRA-style baseline on multiple evaluation tasks with both Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct.
- We evaluate inference performance on NVIDIA L20. The native kernels achieve one-token decode speedups of 1.85–$2.07\times$ over PyTorch and 1.18–$1.27\times$ over the fastest measured Triton variant at each tested context length.

2 MOTIVATION

2.1 MIXTURE-OF-EXPERTS

A sparse mixture-of-experts (MoE) layer consists of E experts and a router. For an input token state x, it combines expert outputs using the standard MoE formulation (Shazeer et al., 2017)

$$y(x) = \sum_{e=1}^{E} p_e(x) f_e(x), \tag{1}$$

where $y(x)$ is the layer output, e indexes the experts, $f_e(x)$ is expert e's output, and $p_e(x)$ is its routing weight. In the SwiGLU MoE considered here, each expert is a feed-forward network with gate, up, and down projections. Sparse routing evaluates only a small subset of experts per token, while retaining the weights of all experts. Common MoE training objectives combine a language-modeling loss with a load-balancing term (Fedus et al., 2022), which encourages balanced expert utilization but does not explicitly enforce distinct expert transformations. These properties motivate examining shared structure and compact representations of expert differences.

2.2 MEASURING THE STRUCTURE OF EXPERT DIFFERENCES

We analyze Phi-tiny-MoE-instruct (Li et al., 2025), with 32 layers and $E = 16$ experts per layer. Within each layer, let $W_{e,q}$ be the weight matrix of expert e at projection $q \in \{G, U, D\}$, corresponding to gate, up, and down. We define

$$\overline{W}_q = \frac{1}{E} \sum_{e=1}^{E} W_{e,q}, \qquad D_{e,q} = W_{e,q} - \overline{W}_q, \tag{2}$$

Table 1: Low-rank approximation of expert differences. Retention is reported in percent.

	Rank 8			Rank 32			Rank 128		
Approximation	Gate	Up	Down	Gate	Up	Down	Gate	Up	Down
Activation-aware	44.70	38.14	22.22	63.02	56.20	47.89	82.97	79.30	66.51
Weight SVD	34.93	29.67	12.13	54.00	46.80	22.53	76.55	71.23	50.94
Random	1.79	1.77	1.84	7.15	7.15	7.37	28.59	28.56	29.30

Table 2: Projection orderings by activation-aware retention at rank 32.

Ordering	$G>U>D$	$G>D>U$	$U>G>D$	$U>D>G$	$D>G>U$	$D>U>G$	Total
Pairs	381	16	65	7	33	10	512

where $\overline{W}_q$ is the mean weight across experts and $D_{e,q}$ is expert e's deviation from this mean.

Using FineWeb-Edu (Penedo et al., 2024) and the unchanged model's Top-2 router, we collect 50,000 calibration and 10,000 test tokens per layer and expert from disjoint sequence sets. Let x_i be the i-th test input to the projection and $n = 10{,}000$ the test count. The matrix $Y = [D_{e,q}x_1, \ldots, D_{e,q}x_n]$ contains the residual outputs, and $\widehat{Y}_r$ contains their approximations using a residual weight matrix of rank at most r. We measure residual-output retention as

$$R_r = 1 - \frac{\|Y - \widehat{Y}_r\|_F^2}{\|Y\|_F^2}, \tag{3}$$

where $\|\cdot\|_F^2$ sums squared matrix entries. Thus R_r measures the fraction of residual-output energy recovered relative to predicting zero, with larger values indicating better approximation.

We compare activation-aware approximation fitted to calibration outputs, weight SVD fitted to $D_{e,q}$, and random orthonormal subspaces averaged over five trials, using the same test inputs. We report median retention across 512 layer-expert pairs per projection. The main text presents ranks 8, 32, and 128; Section A provides computation details and results at the intermediate ranks 16 and 64.

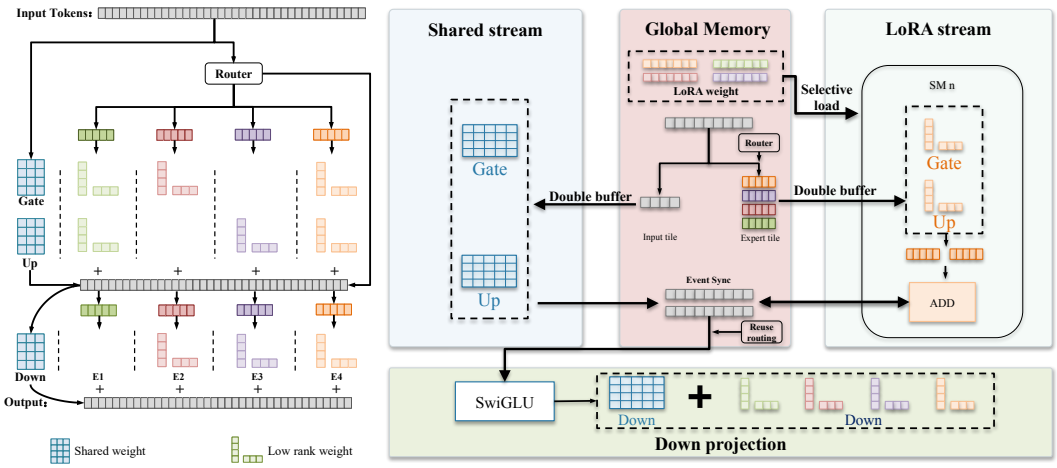
2.3 Low-rank structure across experts and projections

Table 1 shows that expert differences contain structured low-rank components. At rank 32, activation-aware approximation retains 63.02%, 56.20%, and 47.89% of residual-output energy for gate, up, and down, respectively, compared with about 7% for random subspaces. Weight SVD lies between the two at all three displayed ranks, indicating that accounting for actual inputs improves the approximation. Even rank 8 captures substantial residual-output energy, while rank 128 recovers more of the expert differences. Together, these results motivate representing expert-specific variation with compact low-rank updates around shared weights. We next examine whether the ordering of projection retention at rank 32 is consistent across experts. Table 2 shows that the ordering of gate, up, and down by rank-32 retention differs among experts. Across the 512 layer-expert pairs, gate has the highest retention in 397 pairs, up in 72, and down in 43. This variation also occurs within individual layers, with 29 of the 32 layers containing experts with different projection orderings.

These observations motivate shared weights with low-rank updates at diverse projection locations. They provide empirical motivation for projection-pattern diversity. However, routing tokens to low-rank experts creates small matrix multiplications, while different projection combinations complicate scheduling, so fewer parameters do not necessarily translate into faster execution. This motivates GPU kernels to reduce overhead for heterogeneous low-rank experts.

3 Heterogeneous Low-Rank Experts

HeteroLoRA-MoE represents each expert using shared dense weights and its own low-rank updates. Experts differ in both their update factors and the projections to which those updates are applied. A single router selects experts and combines their complete outputs. Figure 1(a) shows the architecture.



a) Heterogeneous experts **b)** HeteroKernel execution

Figure 1: HeteroLoRA-MoE architecture and execution. (a) Experts combine shared projections with independent low-rank updates following four projection patterns. (b) The illustrated HeteroKernel execution uses separate CUDA streams for shared and adapter computations, exchanging global-memory results with event synchronization. Routing and projection masks select the adapter work. Double shared-memory buffers overlap tile loading and computation. SwiGLU preserves expert-specific intermediates for down adapters, while their routing-weighted aggregate feeds one shared down projection.

3.1 Expert Representation

Full-parameter MoE experts use separate gate, up, and down weights. Our experts share base weights and apply low-rank updates to different projection combinations. The expert output is

$$f_e(x) = \widetilde{W}_{e,D}[\sigma(\widetilde{W}_{e,G}x) \odot \widetilde{W}_{e,U}x], \tag{4}$$

where e indexes experts, σ denotes SiLU, and $\odot$ is elementwise multiplication. For input $x \in \mathbb{R}^H$, the shared weights are $W_G, W_U \in \mathbb{R}^{I\times H}$ and $W_D \in \mathbb{R}^{H\times I}$, with hidden dimension H and intermediate dimension I. Expert weights are $\widetilde{W}_{e,q} = W_q + \Delta W_{e,q}$ for $q \in \{G, U, D\}$.

Different experts place low-rank updates at different projection combinations. Each expert has a fixed binary mask $\boldsymbol{m}_e = (m_{e,G}, m_{e,U}, m_{e,D}) \in \{0,1\}^3$, which determines its weights,

$$\Delta W_{e,q} = m_{e,q} B_{e,q} A_{e,q}, \qquad q \in \{G, U, D\}, \tag{5}$$

where $m_{e,q} = 1$ enables the LoRA module at projection q. Each enabled module has independent factors $A_{e,q} \in \mathbb{R}^{r\times d_{\text{in}}}$ and $B_{e,q} \in \mathbb{R}^{d_{\text{out}}\times r}$, where r is the adapter rank. d_{in} and d_{out} are its input and output dimensions. When $m_{e,q} = 0$, the low-rank term is omitted and no adapter parameters are allocated. The mask is fixed at model construction and does not depend on the input token.

Motivated by the observed variation in the low-rank structure of expert differences across projections and within layers, we construct heterogeneous experts with updates at different projection locations. Under a fixed adapter parameter budget, we reduce the total number of adapters and increase the rank of each enabled adapter, concentrating adaptation capacity at selected locations. Token-dependent routing allows the model to learn which expert projection patterns are suitable for each token. We use four masks, $(1,1,0)$, $(1,0,1)$, $(0,1,1)$, and $(1,1,1)$, corresponding to gate+up, gate+down, up+down, and gate+up+down. Each expert learns independent low-rank factors, and we assign equal numbers of experts to each pattern, balancing adapter parameter budgets across projections.

3.2 Multi-output Formulation

The shared weights allow us to express the expert layer as one multi-output transformation. For input x, let $g_0 = W_G x$ and $u_0 = W_U x$ denote the shared gate and up outputs. Expert-specific updates

produce the intermediate columns

$$h_e(x) = \sigma(g_0 + \Delta W_{e,G}x) \odot (u_0 + \Delta W_{e,U}x), \qquad V(x) = [h_1(x), \ldots, h_E(x)] \in \mathbb{R}^{I\times E}. \quad (6)$$

Each $\Delta W_{e,q} = m_{e,q}B_{e,q}A_{e,q}$ follows the fixed projection mask in Equation (5). Applying the shared down projection and the expert-specific down updates gives

$$F(x) = W_D V(x) + [\Delta W_{1,D}h_1(x), \ldots, \Delta W_{E,D}h_E(x)] = [f_1(x), \ldots, f_E(x)]. \quad (7)$$

Here $F(x) \in \mathbb{R}^{H\times E}$ collects the expert outputs as columns. Thus shared projections and heterogeneous low-rank branches define a single multi-output function, with each down update acting on its own expert's intermediate state.

The router masks and combines these output columns. A router $W_R \in \mathbb{R}^{E\times H}$ produces $\boldsymbol{\pi}(x) = \mathrm{softmax}(W_R x)$. Let $\mathcal{S}(x)$ contain the expert indices selected by Top-K routing, and let $z_e(x)$ be one for $e \in \mathcal{S}(x)$ and zero otherwise. The normalized masked weights and final output $y(x)$ are

$$p_e(x) = \frac{z_e(x)\pi_e(x)}{\sum_{j=1}^{E} z_j(x)\pi_j(x)}, \qquad y(x) = F(x)\boldsymbol{p}(x) = \sum_{e\in\mathcal{S}(x)} p_e(x)f_e(x), \quad (8)$$

where $\boldsymbol{p}(x) = [p_1(x), \ldots, p_E(x)]^\top$ contains the normalized routing weights, with zero entries for unselected experts. The fixed projection masks $\boldsymbol{m}_e$ define each expert's structure, while the input-dependent routing mask $\boldsymbol{z}(x)$ selects which expert outputs contribute. This formulation is equivalent to the routed mixture in Equation (1).

3.3 Initialization and training

HeteroLoRA-MoE supports training from scratch or initialization from a pretrained dense model. Shared weights are initialized randomly or inherited from the source model, while routers and $A_{e,q}$ are randomly initialized and $B_{e,q}$ are set to zero (Hu et al., 2022). With pretrained initialization, all experts initially reproduce the source FFN, preserving its function under normalized routing. Training can jointly optimize all parameters or freeze pretrained shared weights for parameter-efficient adaptation.

For L MoE layers, we combine token-level language-model cross-entropy $\mathcal{L}_{\mathrm{LM}}$ with per-layer router load balancing (Fedus et al., 2022),

$$\mathcal{L} = \mathcal{L}_{\mathrm{LM}} + \frac{\lambda}{L}\sum_{\ell=1}^{L}\mathcal{L}_{\mathrm{bal}}^{(\ell)}, \qquad \mathcal{L}_{\mathrm{bal}}^{(\ell)} = E\sum_{e=1}^{E}\rho_e^{(\ell)}\bar{\pi}_e^{(\ell)}. \quad (9)$$

where $\mathcal{L}_{\mathrm{bal}}^{(\ell)}$ is the load-balancing loss for layer ℓ, λ controls its strength, $\rho_e^{(\ell)}$ is the fraction of routing slots assigned to expert e in layer ℓ, and $\bar{\pi}_e^{(\ell)}$ is its mean routing probability over tokens in the batch before Top-K selection. The balancing loss encourages balanced expert utilization.

4 HeteroKernel: Structure-Aware GPU Execution

HeteroKernel is a structure-aware GPU implementation for HeteroLoRA-MoE. Since routing depends only on the input, expert selection is available before adapter computation. HeteroKernel uses routing and projection masks to skip inactive branches, avoiding unnecessary computation and memory accesses. It groups the remaining computations to reuse data and overlap memory loads with computation. Figure 1(b) shows the execution flow.

4.1 Early masking and shared computation

An adapter executes only when its expert is selected and its projection is enabled. Its activation condition is therefore

$$a_{e,q}(x) = z_e(x)m_{e,q}, \qquad q \in \{G, U, D\}, \quad (10)$$

where $a_{e,q}(x) = 1$ enables the corresponding adapter. Applying this condition before execution avoids computing the unselected columns of $F(x)$. Shared gate and up outputs are computed once per token and reused by the selected experts, which add only their enabled low-rank updates.

For the down projection, linearity allows the routing-weighted expert intermediates to be aggregated before applying the shared dense weight:

$$\begin{aligned} y(x) &= \sum_{e \in \mathcal{S}(x)} p_e(x)\big(W_D + \Delta W_{e,D}\big) h_e(x) \\ &= W_D \underbrace{\left(\sum_{e \in \mathcal{S}(x)} p_e(x) h_e(x) \right)}_{\bar{h}(x)} + \sum_{e \in \mathcal{S}(x)} p_e(x) \Delta W_{e,D} h_e(x). \end{aligned} \tag{11}$$

Thus the shared dense down projection is applied once per token to $\bar{h}(x)$. Each enabled down adapter still acts on its own expert intermediate $h_e(x)$, with its correction weighted and added to the shared output. This identity is exact in real arithmetic and also applies to homogeneous shared-base experts; projection masks additionally allow absent adapter branches to be skipped.

4.2 GPU Execution Mapping

We map the computations in Equation (11) to separate shared and adapter streams. Token–expert assignments are grouped and tiled by expert, with each block handling one expert's tile. The fixed mask $m_{e,q}$ uniformly skips disabled loads and products; blocks without valid assignments return immediately. Shared gate and up weights are stacked into one matrix product. Adapter warps use Tensor Core operations on token and rank tiles, keeping operand fragments and accumulators in registers. Gate and up adapters reuse input fragments with separate factors and accumulators.

Threads cooperatively stage activations and expert factors in shared memory for reuse across assigned tokens. In the larger-tile schedule, each tile is loaded once per block and shared across warps. Padded row strides reduce bank conflicts while maintaining alignment. Double buffering overlaps computation with asynchronous loading of the next tile, synchronizing before buffers exchange roles.

An inverse assignment map reconnects grouped results to their source tokens. The composition kernel fuses adapter addition, SiLU, elementwise multiplication, and routing-weighted reduction, avoiding separate activation and elementwise-product buffers. It produces the expert intermediates $h_e(x)$ and their aggregate $\bar{h}(x)$ required by Equation (11). The down adapter's multiplication by $B_{e,D}$ can be fused with weighted reduction and addition to the shared output, avoiding a full-width correction buffer; for larger rank and token tiles, a grouped path materializes corrections before reduction. For larger workloads, separate CUDA streams overlap shared gate/up computation with routing-plan construction followed by adapter products, synchronizing before composition. The shared down and down-adapter paths then overlap until their outputs are combined.

5 Experiments

We evaluate projection-pattern allocation under a controlled adapter budget and the inference behavior of the corresponding execution design. Auxiliary pretraining evaluates end-to-end trainability.

5.1 Experimental Setup

We initialize from Qwen2.5-0.5B-Instruct (Qwen et al., 2025), with 24 layers, hidden width 896, and intermediate width 4,864. We compare HeteroLoRA-MoE with the original Instruct checkpoint, dense SFT, Full-weight MoE, LoRA SFT, and a homogeneous baseline termed MixLoRA-style, which follows MixLoRA's shared-FFN expert design (Li et al., 2024). All MoE configurations in the main comparison use 16 Top-2 experts per MLP. HeteroLoRA-MoE assigns equal numbers of rank-32 experts to the four projection patterns, freezing inherited weights and training routers and adapters with $\lambda = 10^{-3}$. LoRA SFT uses rank 384, while MixLoRA-style uses rank 24 on all three projections, both matching our total adapter parameter budget. Additional results on Qwen2.5-1.5B-Instruct are reported in Appendix C.3.

Mixed SFT samples up to 10,000 examples per dataset from the training splits of GSM8K (Cobbe et al., 2021), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), MMLU (Hendrycks et al., 2020) (auxiliary training split), and MBPP (Austin et al., 2021); HumanEval (Chen et al., 2021) is

Table 3: Mixed-SFT quality (%, higher is better). All of the trained models report mean ± standard deviation over seeds 42–45. Original Instruct is evaluated without SFT. Bold marks the highest mean in each column, including ties.

Method	GSM8K	HellaSwag	PIQA	MMLU	MBPP	HumanEval
Original Instruct	31.31	28.44	52.61	35.34	22.00	23.78
Dense SFT	31.94 ± 1.09	52.63 ± 0.57	68.78 ± 0.49	46.81 ± 0.16	26.00 ± 0.85	22.41 ± 1.69
LoRA SFT	31.25 ± 0.80	55.04 ± 0.56	68.88 ± 1.66	45.27 ± 0.68	20.15 ± 0.96	24.09 ± 2.60
Full-weight MoE	31.84 ± 1.12	51.71 ± 0.49	68.00 ± 0.53	46.69 ± 0.20	25.70 ± 1.11	23.93 ± 1.80
Fixed gate+up ($r = 36$)	31.60 ± 0.53	54.30 ± 0.93	68.48 ± 0.36	$\mathbf{47.06} \pm 0.11$	27.00 ± 1.07	22.26 ± 0.53
Fixed gate+down ($r = 36$)	30.88 ± 0.90	53.74 ± 0.50	68.50 ± 0.75	46.79 ± 0.36	26.95 ± 1.26	22.10 ± 1.85
Fixed up+down ($r = 36$)	31.65 ± 1.44	54.29 ± 0.91	68.05 ± 0.97	46.93 ± 0.23	27.55 ± 1.20	21.49 ± 1.52
MixLoRA-style ($r = 24$)	30.93 ± 0.71	53.25 ± 1.05	68.00 ± 0.48	46.84 ± 0.10	27.05 ± 0.17	23.17 ± 1.36
HeteroLoRA-MoE	$\mathbf{32.18} \pm 0.56$	$\mathbf{55.50} \pm 0.82$	$\mathbf{68.89} \pm 0.48$	46.83 ± 0.36	$\mathbf{27.95} \pm 0.98$	$\mathbf{24.24} \pm 0.66$

evaluation-only. Evaluation uses held-out test sets, or validation sets for HellaSwag and PIQA. We train for one epoch with sequence length is 1,024 and global batch size 64. All mixed-SFT methods use a learning rate of 1×10^{-4}. All trained methods in the main comparison use seeds 42–45. All use the PyTorch reference implementation and the common evaluation protocol in Section B. We report percentage scores, with mean ± standard deviation for four-seed runs. Training hyperparameters are summarized in Appendix D.

5.2 Downstream quality and parameter efficiency

Table 3 compares downstream quality across adaptation methods. Under the same parameter budget, HeteroLoRA-MoE achieves higher mean scores than MixLoRA-style on multiple tasks, with nearly identical MMLU performance. It also exhibits lower standard deviations on GSM8K, HellaSwag, and HumanEval, indicating more consistent performance across training seeds on these tasks. These results support heterogeneous projection patterns as an effective allocation of a fixed adapter budget.

These controlled comparisons support the heterogeneous allocation motivated by our observations of expert differences: under the same total adapter budget, concentrating updates at selected projection locations raises each enabled adapter's rank from 24 to 32 and improves mean performance on multiple tasks. We conjecture that token-dependent routing helps direct this higher-rank capacity toward projections more relevant to each token, contributing to the observed gains.

With the same 16 experts and Top-2 routing, HeteroLoRA-MoE reduces stored and trainable parameters by 87.4% and 96.8% relative to Full-weight MoE, while achieving higher mean scores on multiple evaluation tasks, including a 0.31-point gain on HumanEval. It also has higher mean scores than dense SFT and LoRA SFT on multiple evaluation tasks. These results suggest that heterogeneous experts effectively exploit low-rank structure to maintain competitive downstream quality with substantially fewer parameters than full-weight experts.

5.3 GPU Inference Performance

We evaluate full-model latency on an NVIDIA L20 GPU with BF16 operands, PyTorch 2.11.0, and CUDA 12.8. HeteroKernel replaces the MoE MLP modules in Hugging Face Transformers 4.53.0, retaining its attention and KV cache. Measurements use identical token sequences, batch size one, and SDPA attention, reporting CUDA-event p50 latency over 20 repetitions after five warm-ups. Training-only routing statistics are disabled for all MoE implementations. Prefill measures a forward pass, and decode measures one step with an initialized KV cache. Table 4 compares inference latency across models and across implementations of HeteroLoRA-MoE.

Table 4 shows that HeteroKernel reduces small-matrix and heterogeneous scheduling overhead, achieving 1.85–$6.79\times$ speedups over the same model's PyTorch implementation. Compared with the measured Triton implementation, HeteroKernel achieves 1.33–$1.41\times$ speedups for prefill and 1.18–$1.27\times$ for one-token decode. With 16 routed experts, latency remains within 9% of unmerged LoRA SFT across all reported settings. It is also lower than the measured PyTorch implementations of Full-weight MoE and MixLoRA-style and the Triton implementation of HeteroLoRA-MoE. These results show that HeteroKernel alleviates the execution overhead of heterogeneous low-rank experts,

Table 4: Full-model p50 latency on L20 in ms (BF16, batch 1). Columns give prefill tokens and decode context lengths. LoRA SFT keeps adapter weights unmerged.

		Prefill				Decode			
Model	Backend	32	128	512	2048	32	128	512	2048
Dense SFT	PyTorch	16.92	16.75	16.89	27.53	21.61	21.58	21.64	21.68
LoRA SFT	PyTorch	17.12	18.06	18.32	27.66	22.35	22.26	23.44	23.42
Full-weight MoE	PyTorch	60.37	67.60	71.77	94.46	35.76	38.21	36.82	38.28
MixLoRA-style	PyTorch	107.44	128.74	130.80	159.22	57.01	57.38	45.10	45.63
MixLoRA-style	CUDA	19.24	18.94	19.12	40.80	23.87	23.92	23.99	23.83
Implementation comparison for HeteroKernel	PyTorch	98.52	113.95	120.86	138.11	46.64	44.07	44.53	44.22
	Triton	24.16	25.08	24.43	39.98	27.69	27.79	28.58	30.36
	CUDA	17.78	17.76	17.81	29.98	22.52	23.63	23.68	23.88

Table 5: Expert-count and rank ablations (%). All configurations report the mean over seeds 42–45.

Configuration	GSM8K	HellaSwag	PIQA	MMLU	MBPP	HumanEval
$E = 16, r = 32$	**32.18**	**55.50**	68.89	**46.83**	27.95	**24.24**
$E = 16, r = 64$	31.16	54.90	**70.84**	46.82	27.60	19.66
$E = 32, r = 32$	32.15	55.21	68.44	46.81	27.80	19.51
$E = 32, r = 64$	31.92	55.47	69.64	46.69	**28.00**	17.68

helping translate parameter savings into practical inference speedups. Matched-backend settings and measurements, and HeteroKernel quality checks are provided in Sections C.1 and C.2.

5.4 Ablation Studies

To test projection heterogeneity at a fixed total adapter budget, we compare our mixed-pattern experts with fixed gate+up, gate+down, and up+down experts at rank 36, alongside the all-projection rank-24 baseline. All use 16 Top-2 experts and the same four-seed SFT protocol (Table 3). HeteroLoRA-MoE achieves higher mean scores than all three fixed two-projection variants on multiple tasks, while fixed gate+up performs best on MMLU. These results support our choice of heterogeneous projection patterns over a uniform pattern for all experts.

Table 5 examines the effects of expert count and LoRA rank. The default configuration achieves competitive downstream quality. Increasing expert count or rank yields limited and inconsistent gains, with some larger configurations performing worse. These results are consistent with our low-rank motivation and suggest that selective capacity allocation can be more effective than simply increasing adapter rank. One possible explanation is that compact updates already capture useful low-rank structure, while larger configurations may be more prone to overfitting, harder to optimize, or more sensitive to hyperparameters.

Table 6 disables one optimization at a time from the full CUDA implementation. Removing early routing masks raises block latency to $3.53\times$ and $4.95\times$ the full implementation at 512 and 2,048 tokens, highlighting the benefit of skipping unselected adapter computation. Removing shared-down reuse increases latency by 21–56%, with the largest effect at one token. Separating the fused composition operations increases latency by 22–44% at 512–2,048 tokens, while disabling double buffering adds 21–28% across all three sizes. These results support complementary benefits from selective execution, shared computation, fusion, and overlapped loading.

5.5 Auxiliary end-to-end pretraining

We train four Phi-style configurations of HeteroLoRA-MoE from scratch on FineWeb-Edu (Penedo et al., 2024) for approximately one epoch. Figure 2 shows their training loss curves, smoothed over 100 logged records. The decreasing training loss across all four configurations provides auxiliary evidence of end-to-end trainability.

Table 6: CUDA kernel ablations. Block p50 latency (ms) at three token counts. Each variant disables one optimization.

CUDA variant	1 token	512 tokens	2048 tokens
Full CUDA	**0.2324**	**0.4455**	**1.1049**
w/o early routing mask	0.2499	1.5729	5.4671
w/o shared-down reuse	0.3637	0.5939	1.3343
w/o composition fusion	0.2365	0.5437	1.5933
w/o double buffering	0.2940	0.5690	1.3418

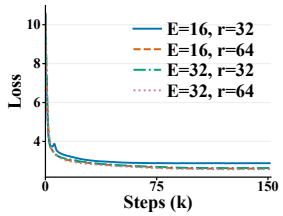


Figure 2: Auxiliary pretraining loss on FineWeb-Edu.

6 Related Work

6.1 Routed and heterogeneous low-rank experts

LoRAMoE, MoELoRA, and MixLoRA route among low-rank modules (Dou et al., 2024; Luo et al., 2024; Li et al., 2024). MixLoRA is particularly close, using Top-k LoRA experts in frozen FFNs and reusing shared gate and up computations, but retaining a dense down operation for each routed intermediate. We aggregate these intermediates before the shared down projection. This also benefits homogeneous mixtures, while heterogeneous masks expose additional adapter branches to skip. MoA varies adapter families and placements, GuiLoMo allocates projection-wise expert counts and ranks, and DMEP prunes experts independently across modules (Cao et al., 2026; Chen et al., 2025; Li et al., 2026). These methods mix adapters across Transformer locations or construct module-level expert pools. HeteroLoRA-MoE instead defines each expert by a gate, up, and down mask within one SwiGLU block, selected jointly by a single router.

6.2 MoE and LoRA execution

Tutel, MegaBlocks, and ScatterMoE optimize communication, block-sparse computation, and scattered token layouts for dense experts (Hwang et al., 2023; Gale et al., 2023; Tan et al., 2024). Punica and S-LoRA batch request-specific adapters and manage their memory (Chen et al., 2024; Sheng et al., 2024). Our measurements compare local PyTorch, Triton, and CUDA implementations rather than these serving systems. MeteoRA is the closest token-routed LoRA execution system, using per-module routers and a tailored Triton kernel (Xu et al., 2025). Our execution spans a complete SwiGLU block under one router, enabling cross-projection reuse, selective adapter loading, shared-down factorization, and direct or grouped schedules.

6.3 Sparse upcycling and efficient expert construction

Sparse upcycling initializes routed experts from dense checkpoints (Komatsuzaki et al., 2023), with extensions exploring routing, granularity, training scale, and expert re-initialization (Hui et al., 2025; Nakamura et al., 2025). LLaMA-MoE and DSMoE partition dense FFNs into smaller experts (Zhu et al., 2024; Lv et al., 2025). PESC introduces bottleneck adapters, while DeRS represents experts as a shared base with sparse or low-rank deltas (Wu et al., 2024; Huang et al., 2025). Building on shared-base representations, we define experts through balanced SwiGLU projection patterns and exploit their dependencies in GPU execution.

7 Conclusion and Future Work

HeteroLoRA-MoE combines shared dense weights with heterogeneous low-rank experts defined by different projection patterns. Its GPU kernels reuse shared computations, group adapter products, and overlap weight loading with computation. Across four seeds, adaptation of Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct yields higher mean scores than the parameter-matched homogeneous mixture on multiple evaluation tasks at both model scales, while the HeteroKernels improve full-model inference latency in a workload-dependent manner.

The fixed projection family does not adapt its patterns or capacity allocation to layers or tasks. Learning where to place adapters and how much rank to allocate is a natural extension of this design.

AI USE STATEMENT

Generative AI tools assisted with language polishing, manuscript and figure/table formatting, and portions of code implementation and debugging. The research ideas and design were developed by the authors, who take responsibility for the final manuscript, code, and reported results.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Jie Cao, Tianwei Lin, Bo Yuan, Rolan Yan, Hongyang He, Wenqiao Zhang, Juncheng Li, Dongping Zhang, Siliang Tang, and Yueting Zhuang. Moa: Heterogeneous mixture of adapters for parameter-efficient fine-tuning of large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21056–21073, 2026.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. In *Proceedings of Machine Learning and Systems*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mo Bavarian, Clemens S. Winter, Phil Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021. URL https://api.semanticscholar.org/CorpusID:235755472.

Xinrong Chen, Hengyuan Zhang, Yingmin Qiu, Xiao Liang, Ziyue Li, Guanyu Wang, Weiping Li, Tong Mo, Hayden Kwok-Hay So, and Ngai Wong. GuiLoMo: Allocating experts and ranks for LoRA-MoE via bilevel optimization with GuidedSelection vectors. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 7552–7567, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.399. URL https://aclanthology.org/2025.findings-emnlp.399/.

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

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1280–1297, 2024.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Wei Shen, Limao Xiong, Yuhao Zhou, Xiao Wang, Zhiheng Xi, Xiaoran Fan, Shiliang Pu, Jiang Zhu, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. LoRAMoE: Alleviating world knowledge forgetting in large language models via MoE-style plugin. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1932–1945, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi:

10.18653/v1/2024.acl-long.106. URL https://aclanthology.org/2024.acl-long.106/.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems*, 5:288–304, 2023.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *CoRR*, abs/2009.03300, 2020. URL https://arxiv.org/abs/2009.03300.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Yongqi Huang, Peng Ye, Chenyu Huang, Jianjian Cao, Lin Zhang, Baopu Li, Gang Yu, and Tao Chen. Ders: Towards extremely efficient upcycled mixture-of-experts models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10056–10066. IEEE, 2025.

Tingfeng Hui, Zhenyu Zhang, Shuohuan Wang, Yu Sun, Hua Wu, and Sen Su. Upcycling instruction tuning from dense to mixture-of-experts via parameter merging. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13012–13031, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.637. URL https://aclanthology.org/2025.acl-long.637/.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems*, 5:269–287, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=T5nUQDrM4u.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Dengchun Li, Yingzi Ma, Naizheng Wang, Zhengmao Ye, Zhiyuan Cheng, Yinghao Tang, Yan Zhang, Lei Duan, Jie Zuo, Cal Yang, and Mingjie Tang. Mixlora: Enhancing large language models fine-tuning with lora-based mixture of experts, 2024. URL https://arxiv.org/abs/2404.15159.

Weihang Li, Jianchun Liu, and Hongli Xu. Adaptive and fine-grained module-wise expert pruning for efficient lora-moe fine-tuning, 2026. URL https://arxiv.org/abs/2604.26340.

Zichong Li, Chen Liang, Zixuan Zhang, Ilgee Hong, Young Jin Kim, Weizhu Chen, and Tuo Zhao. Slimmoe: Structured compression of large moe models via expert slimming and distillation. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=oaCUsn391F.

Tongxu Luo, Jiahe Lei, Fangyu Lei, Weihao Liu, Shizhu He, Jun Zhao, and Kang Liu. Moelora: Contrastive learning guided mixture of experts on parameter-efficient fine-tuning for large language models. *arXiv preprint arXiv:2402.12851*, 2024.

Minxuan Lv, Zhenpeng Su, Leiyu Pan, Yizhe Xiong, Zijia Lin, Hui Chen, Wei Zhou, Jungong Han, Guiguang Ding, Wenwu Ou, et al. Dsmoe: Matrix-partitioned experts with dynamic routing for computation-efficient dense llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 19722–19733, 2025.

Taishi Nakamura, Takuya Akiba, Kazuki Fujii, Yusuke Oda, Rio Yokota, and Jun Suzuki. Drop-upcycling: Training sparse mixture of experts with partial re-initialization. In *International Conference on Learning Representations*, volume 2025, pp. 84370–84397, 2025.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: decanting the web for the finest text data at scale. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Qwen, :, 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, Tianyi Tang, 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, 2025. URL https://arxiv.org/abs/2412.15115.

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.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. S-lora: Serving thousands of concurrent lora adapters. In *Proceedings of Machine Learning and Systems*, 2024.

Shawn Tan, Yikang Shen, Rameswar Panda, and Aaron Courville. Scattered mixture-of-experts implementation. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=YDZ7GeFLxq.

Haoyuan Wu, Haisheng Zheng, Zhuolun He, and Bei Yu. Parameter-efficient sparsity crafting from dense to mixture-of-experts for instruction tuning on general tasks. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 737–749, 2024.

Jingwei Xu, Junyu Lai, and Yunpeng Huang. Meteora: Multiple-tasks embedded lora for large language models. In *International Conference on Learning Representations*, volume 2025, pp. 55049–55076, 2025.

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, 2019.

Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. Llama-moe: Building mixture-of-experts from llama with continual pre-training. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 15913–15923, 2024.

A Computation of the Low-Rank Analysis

For a fixed layer, expert, and projection, write the weight residual as $D = W_e - \overline{W}$. The analysis collects inputs selected by the original router, excluding padded tokens. Calibration and testing use disjoint sequences, and every expert receives the token counts reported in Section 2. The model remains unchanged throughout collection.

A.1 Gate and up projections

Let Y_{cal} contain the calibration residual outputs as columns. We accumulate the uncentered second-moment matrix

$$C_{\mathrm{cal}} = Y_{\mathrm{cal}} Y_{\mathrm{cal}}^{\mathsf{T}}. \tag{12}$$

The leading r eigenvectors form an orthonormal basis U_r. This basis minimizes

$$\min_{U^{\mathsf{T}}U=I} \|Y_{\mathrm{cal}} - UU^{\mathsf{T}} Y_{\mathrm{cal}}\|_F^2. \tag{13}$$

The resulting rank-r residual approximation is $\widehat{D} = U_r U_r^{\mathsf{T}} D$. For independently collected test outputs, define $C_{\mathrm{test}} = Y_{\mathrm{test}} Y_{\mathrm{test}}^{\mathsf{T}}$. Orthogonal projection gives

$$R_r = \frac{\mathrm{tr}(U_r^{\mathsf{T}} C_{\mathrm{test}} U_r)}{\mathrm{tr}(C_{\mathrm{test}})}. \tag{14}$$

Directions are selected using calibration statistics, while retention is computed using test statistics.

A.2 Down projection

For down, we accumulate input statistics in its smaller, 448-dimensional input space. With $C_x = \sum_i x_i x_i^{\mathsf{T}}$ from calibration inputs, the reduced-rank regression objective is

$$\sum_i \|(D - \widehat{D})x_i\|_2^2 = \|(D - \widehat{D})C_x^{1/2}\|_F^2, \tag{15}$$

subject to $\mathrm{rank}(\widehat{D}) \le r$. We obtain the leading r eigenvectors V_r of $C_x^{1/2} D^{\mathsf{T}} D C_x^{1/2}$ and form

$$\widehat{D} = D C_x^{1/2} V_r V_r^{\mathsf{T}} C_x^{-1/2}. \tag{16}$$

The inverse square root uses a pseudoinverse, retaining input eigenvalues above 10^{-7} times the largest eigenvalue. The test score is computed directly from Equation (3) using Dx_i and $\widehat{D}x_i$ on held-out inputs.

A.3 Reference approximations and aggregation

The weight-based reference uses the rank-r truncated SVD of D. Random bases are generated from Gaussian matrices followed by QR orthogonalization. For gate and up, a random basis Q_r gives $\widehat{D} = Q_r Q_r^{\mathsf{T}} D$; for down, it gives $\widehat{D} = D Q_r Q_r^{\mathsf{T}}$. We average the test scores of five random bases for each expert and projection. Each reported median is then taken over the 512 layer-expert pairs, separately for each projection and rank. Projection orderings compare the three activation-aware scores within each pair.

A.4 Additional ranks

Table 7 reports ranks 16 and 64, complementing Table 1 under the same protocol.

B Downstream Evaluation Protocol

The quality comparisons in Tables 3 and 5 use the same task splits, prompts, and scoring rules for all models, including Original Instruct. We evaluate GSM8K, MMLU, MBPP, and HumanEval on their test sets, and HellaSwag and PIQA on their validation sets. MMLU SFT uses the separate auxiliary

Table 7: Median held-out residual-output retention (%) at ranks 16 and 64 on Phi-tiny-MoE-instruct.

	Rank 16			Rank 64		
Approximation	Gate	Up	Down	Gate	Up	Down
Activation-aware	53.38	46.61	28.62	72.88	67.47	50.01
Weight SVD	43.77	37.44	15.88	64.63	57.50	33.39
Random	3.57	3.57	3.70	14.29	14.28	14.72

training split, and HumanEval is evaluation-only. Task prompts are wrapped in the tokenizer's chat template with an assistant-generation prefix.

HellaSwag, PIQA, and MMLU present the question and letter-labeled options. We select the answer letter with the highest conditional log-likelihood and report accuracy. MMLU accuracy is micro-averaged: the total number of correct answers across all 14,042 test questions is divided by 14,042. GSM8K uses greedy generation and numeric exact match after answer extraction and normalization. Extraction prioritizes the final `####` answer or boxed answer, then falls back to the last numeric value. Generative tasks use at most 512 new tokens, with input prompts truncated to 2,048 tokens.

MBPP evaluation uses all 500 problems in the test split of the `full` configuration. MBPP prompts include the task description and its provided example tests; HumanEval prompts ask for a complete implementation of the supplied function. We generate one greedy candidate per problem and report execution-based pass@1. The evaluator extracts the last fenced code block, or uses the full response when no code fence is present. MBPP checks the provided tests and challenge tests, while HumanEval uses its supplied test harness. Each candidate has a five-second execution timeout. Scores are reported as percentages, with means and standard deviations across training seeds where indicated. For all tasks, the reported standard deviation is the population standard deviation across the four training seeds (42–45), computed with a divisor of four (`ddof=0`).

C Additional Controlled Comparisons

C.1 Matched-backend latency

We evaluate MixLoRA-style and HeteroLoRA-MoE with both the PyTorch reference and our CUDA execution path, using their trained seed-42 checkpoints. Both models use 16 Top-2 experts and the same total adapter budget. The comparison reuses identical input sequences, BF16, batch size one, SDPA attention, and the timing protocol of the main latency experiment. The CUDA implementations retain each model's native adapter rank, using rank 24 for homogeneous experts and rank 32 for heterogeneous experts. Adapter factors and intermediate tensors are stored at their native ranks.

Table 8: Full-model p50 latency on L20 in ms, reproducing the MixLoRA-style and HeteroLoRA-MoE measurements from Table 4 for comparison within each backend. CUDA uses native rank-24 and rank-32 paths for homogeneous and heterogeneous experts, respectively.

		Prefill				Decode			
Model	Backend	32	128	512	2048	32	128	512	2048
MixLoRA-style	PyTorch	107.44	128.74	130.80	159.22	57.01	57.38	45.10	45.63
MixLoRA-style	CUDA	19.24	18.94	19.12	40.80	23.87	23.92	23.99	23.83
HeteroLoRA-MoE	PyTorch	98.52	113.95	120.86	138.11	46.64	44.07	44.53	44.22
HeteroLoRA-MoE	CUDA	17.78	17.76	17.81	29.98	22.52	23.63	23.68	23.88

Under the CUDA backend, HeteroLoRA-MoE has 6.2–26.5% lower measured prefill latency than MixLoRA-style in Table 8. Decode latency is 1.2–5.7% lower at context lengths 32–512 and 0.2% higher at context length 2,048. Both models benefit from specialized execution, with comparable decode latency despite their different projection patterns.

C.2 Downstream quality with CUDA execution

We evaluate the same seed-42 HeteroLoRA-MoE checkpoint with the PyTorch reference and final CUDA implementation on all six tasks using the protocol in Section B. Both runs retain FlashAttention-2 and use identical evaluation examples, prompts, and decoding settings. This paired evaluation measures the downstream effect of changing the execution backend without retraining. Table 9 reports the paired results, which differ by at most 1.22 percentage points. CUDA increases scores on GSM8K, HellaSwag, PIQA, and MMLU by 0.53, 0.10, 0.11, and 0.14 points, respectively, increases HumanEval by 1.22 points, and decreases MBPP by 0.60 points.

Table 9: Downstream quality of the same seed-42 checkpoint across backends (%). The difference is CUDA minus PyTorch in percentage points.

Backend	GSM8K	HellaSwag	PIQA	MMLU	MBPP	HumanEval
PyTorch	31.99	55.58	69.42	47.33	28.20	23.17
CUDA	32.52	55.68	69.53	47.47	27.60	24.39
Difference	+0.53	+0.10	+0.11	+0.14	-0.60	+1.22

C.3 Quality evaluation on Qwen2.5-1.5B-Instruct

This experiment examines adaptation at a larger model scale using Qwen2.5-1.5B-Instruct, which has 28 layers, hidden width 1,536, and intermediate width 8,960. We retain the mixed-SFT data, optimization settings, and six-task evaluation protocol of the main comparison (Sections B and 5). HeteroLoRA-MoE replaces all 28 MLPs with 16 Top-2 experts at rank 32, assigning four experts to each projection pattern. LoRA SFT uses rank-384 adapters on the gate, up, and down projections to match the total adapter parameter budget. Both methods freeze inherited weights. Dense SFT updates all parameters, while Full-weight MoE trains 16 full experts and the router in each layer with Top-2 routing, keeping inherited attention and embeddings frozen. All mixed-SFT methods use a learning rate of 1×10^{-4}, as in the 0.5B experiments. All fine-tuned models use training seeds 42–45, and we report mean $\pm$ standard deviation across the four runs. The original Instruct checkpoint is evaluated without fine-tuning. Table 10 provides the comparison across the six tasks. HeteroLoRA-MoE has higher mean scores than the parameter-matched MixLoRA-style baseline on multiple evaluation tasks, with mean-score gains ranging from 0.61 to 1.25 percentage points across the evaluated tasks.

The larger-scale comparison preserves the same rank-32 heterogeneous versus rank-24 homogeneous allocation at a matched adapter budget. Higher mean scores across all six tasks provide further empirical support for the effectiveness of this heterogeneous allocation at the larger model scale.

Table 10: Mixed-SFT quality on Qwen2.5-1.5B-Instruct (%, higher is better). Rows and tasks follow Table 3. Fine-tuned results report mean $\pm$ standard deviation over training seeds 42–45. Original Instruct is evaluated without SFT. Bold marks the highest mean in each column, including ties.

Method	GSM8K	HellaSwag	PIQA	MMLU	MBPP	HumanEval
Original Instruct	66.94	45.41	71.55	57.96	41.60	54.27
Dense SFT	70.87 $\pm$0.59	80.96 $\pm$0.26	81.39 $\pm$0.63	60.85 $\pm$0.30	49.40 $\pm$0.66	57.77 $\pm$0.51
LoRA SFT	69.20 $\pm$0.71	82.15 $\pm$0.38	82.26 $\pm$0.75	59.54 $\pm$0.42	49.40 $\pm$0.79	55.34 $\pm$0.26
Full-weight MoE	68.97 $\pm$0.14	80.18 $\pm$0.50	80.90 $\pm$0.17	60.51 $\pm$0.54	48.40 $\pm$0.20	54.42 $\pm$0.66
Fixed gate+up ($r = 36$)	70.22 $\pm$0.25	79.36 $\pm$0.62	81.81 $\pm$0.29	58.24 $\pm$0.66	50.10 $\pm$0.33	54.88 $\pm$0.75
Fixed gate+down ($r = 36$)	71.17 $\pm$0.37	82.72 $\pm$0.74	80.97 $\pm$0.41	59.89 $\pm$0.78	51.20 $\pm$0.45	58.38 $\pm$0.26
Fixed up+down ($r = 36$)	70.96 $\pm$0.49	**82.75** $\pm$0.16	82.48 $\pm$0.53	60.28 $\pm$0.20	51.00 $\pm$0.57	58.99 $\pm$0.51
MixLoRA-style ($r = 24$)	71.21 $\pm$0.61	81.54 $\pm$0.28	81.39 $\pm$0.65	60.87 $\pm$0.32	50.95 $\pm$0.70	58.99 $\pm$0.51
HeteroLoRA-MoE	**71.82** $\pm$0.73	82.62 $\pm$0.40	**82.56** $\pm$0.77	**61.88** $\pm$0.44	**52.20** $\pm$0.00	**60.21** $\pm$0.26

D Training Hyperparameters

Table 11 summarizes the common training hyperparameters for all mixed-SFT methods at both model scales, Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct. All methods use the same training data,

and evaluation uses the final checkpoint at the end of the single training epoch. LoRA scaling and dropout apply only to LoRA-based methods, and the router balancing coefficient applies only to MoE methods. Across all evaluation tasks, reported standard deviations are population standard deviations over seeds 42–45 (`ddof=0`); evaluation details are provided in Section B.

Table 11: Training hyperparameters for Qwen mixed SFT.

Hyperparameter	Value
Optimizer	AdamW
Adam $\beta_1, \beta_2, \epsilon$	$0.9,\ 0.999,\ 10^{-8}$
Learning rate (all methods)	1×10^{-4}
Weight decay	0.01
Learning-rate schedule	Cosine decay
Warmup ratio	0.03
Training epochs	1
Effective batch size	64
Maximum sequence length	1,024
Maximum gradient norm	1.0
Training precision	BF16
LoRA scaling α/r	1
LoRA dropout	0
Router balancing coefficient	10^{-3}