Steering-Conditioned LoRA via Sparse Autoencoders

Anonymous Authors[1]

Abstract

Parameter-efficient adaptation specializes large language models (LLMs) while preserving capabilities acquired during pretraining and instruction tuning. Low-Rank Adaptation (LoRA) and routed-adapter methods reduce the number of trainable parameters, but typically allocate updates from dense hidden states, whose entangled coordinates poorly separate overlapping domains. We propose Steering-Conditioned LoRA, which uses Sparse Autoencoder (SAE) features as the control space for low-rank adaptation. The method trains SAEs offline on base-model activations, freezes them, and uses their sparse features to route LoRA specialists and assign token-dependent effective rank. We evaluate Steering-Conditioned LoRA on Qwen3 models from 0.6B to 32B parameters across five domains, SAE-conditioning interventions, and sequential domain orders. Across scales, our method improves average instruction-following scores by $+0.016$ to $+0.052$ and average general-knowledge scores by $+0.017$ to $+0.023$ over base models, whereas gains on harder reasoning benchmarks, including MMLU-Pro and GSM8K, are marginal or negative for larger models. Causal controls show that the SAE signal drives these gains: replacing, randomizing, shuffling, or inverting it reduces target-domain scores by 33.6–65.6%, whereas removing dynamic rank while preserving SAE routing reduces scores by only 11.8%. In sequential adaptation, Steering-Conditioned LoRA reduces mean positive forgetting by 13.1%, and by 38.3% when broad instruction-following data precedes narrower domains. These results show that the main bottleneck in routed parameter-efficient adaptation is not adapter capacity alone, but the representation space used to control it.

1. Introduction

Parameter-efficient fine-tuning (PEFT) has emerged as a practical paradigm for specializing large language models (LLMs) on downstream tasks while preserving the broad capabilities acquired during pretraining and instruction tuning (Lialin et al., 2023; Ding et al., 2023). Full fine-tuning, although effective, updates every model parameter and frequently induces catastrophic forgetting or representation drift, particularly in sequential or multi-domain adaptation scenarios (Kirkpatrick et al., 2017b). Low-Rank Adaptation (LoRA) mitigates this risk by constraining weight updates to low-rank decomposition matrices (Hu et al., 2022). However, standard LoRA employs a static rank allocation across all tokens, layers, and domains. This uniformity is fundamentally misaligned with the heterogeneous nature of language processing: simple tokens, routine instructions, and factual recall require minimal adaptation, whereas complex reasoning, domain-specific jargon, and compositional tasks demand substantially higher capacity (Zhang et al., 2023; Sun et al., 2024). A fixed rank budget consequently underfits difficult instances, overfits trivial ones, and increases the risk of interference between unrelated adaptation objectives.

Conditional adaptation addresses this issues by dynamically modulating update capacity across parameters or inputs. Methods such as AdaLoRA reallocate rank based on parameter saliency (Zhang et al., 2023), while routed adapters and mixture-of-experts (MoE) frameworks dispatch inputs to specialized subnetworks (Jiang et al., 2024; Fedus et al., 2022). Despite their promise, nearly all conditional PEFT methods derive routing decisions from dense hidden states. This design choice introduces a critical bottleneck: dense activations in transformer layers are highly superposed, entangling semantic, syntactic, positional, and domain-specific factors into shared coordinate axes (Elhage et al., 2022; Templeton et al., 2024). A router operating on such entangled representations must infer control signals from mixed coordinates, which often leads to

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

ambiguous routing decisions, cross-domain expert sharing, and increased interference between adaptation pathways (Zhou et al., 2022; Liang & Li, 2024).

This limitation motivates a fundamental shift in how we condition parameter-efficient updates. If routing relies on entangled dense states, can we instead steer adaptation using a representation space that explicitly disentangles functional features? Sparse Autoencoders (SAEs) offer a promising answer. By training a sparse bottleneck over transformer activations, SAEs decompose dense superpositions into interpretable, monosemantic latent features that activate only for specific concepts, syntactic structures, or domain cues (Bricken et al., 2023; Cunningham et al., 2024; Templeton et al., 2024). To date, SAEs have primarily been leveraged for mechanistic interpretability and feature visualization (Turner et al., 2024). We instead repurpose them as a *control interface* for efficient adaptation.

We introduce Steering-Conditioned LoRA, a parameter-efficient framework that routes and scales LoRA updates through frozen SAE features. First, we train SAEs offline on base-model activations and freeze their weights, ensuring that the learned sparse basis remains invariant during downstream adaptation. Then, we use each token's sparse feature vector projected to a gating distribution for fine-tuning each Qwen3 models for medicine, code, mathematics, science and general instruction. This approach selects among multiple lightweight LoRA specialists and dynamically allocates token- and layer-dependent effective rank. The SAE features work as a semantic steering signal, determining not only which adaptation pathway to activate, but also how much low-rank capacity to deploy, without modifying the dense hidden states used for forward computation. Our experiments improves instruction-following and general-knowledge benchmarks over both base models and competitive PEFT baselines such as standard LoRA, AdaLoRA, and DoRA (Hu et al., 2022; Zhang et al., 2023; Liu et al., 2024). Gains on challenging compositional reasoning tasks (e.g., MMLU-Pro (Wang et al., 2024), GSM8K (Mirzadeh et al., 2025) diminish at larger scales, suggesting that sparse routing primarily benefits domains where SAEs expose clear structural boundaries.

Beyond single-domain adaptation, we demonstrate improved retention during sequential domain transitions. Routing updates through semantically localized SAE features, the method reduces overlap between adaptation subspaces, thereby limiting unnecessary parameter reuse across unrelated tasks. In ordered domain experiments, we reduces the mean catastrophic forgetting by 13.1% overall, and by 38.3% when broad instruction-following data is presented before narrower, specialized domains. While the approach does not eliminate forgetting in all regimes and shows limited impact on hard reasoning at scale, it establishes that sparse semantic routing can meaningfully mitigate interference in continual adaptation settings (Kirkpatrick et al., 2017a; Flynn, 2026; Liang & Li, 2024).

Our main contributions are:

- **A new control variable for parameter-efficient adaptation:** Steering-Conditioned LoRA replaces dense-state routing with frozen SAE features to steer LoRA specialists and allocate token-dependent effective rank, enabling sparse, semantically grounded adaptation.
- **Adaptation–retention results across scale and domain:** evaluated on Qwen3 models (0.6B–32B) across five domains, Steering-Conditioned LoRA improves instruction-following and general-knowledge averages while explicitly characterizing its limits on complex reasoning at larger scales.
- **Sequential adaptation under domain order:** structured domain-transition experiments demonstrate reduced catastrophic forgetting, with maximal gains when broad instructional data precedes narrow specialization, highlighting the role of routing sparsity in mitigating interference.
- **Causal evidence for SAE feature geometry:** targeted ablations (dense, random, shuffled, and inverted routing signals) show that disrupting SAE latents degrades performance far more than removing dynamic rank, isolating the sparse representation space as the core source of adaptation gains.

2. Related Work

Parameter-efficient adaptation. Parameter-efficient fine-tuning (PEFT) adapts large language models by updating a small set of parameters while keeping the backbone frozen. LoRA replaces full weight updates with trainable low-rank matrices inserted into selected linear layers (Hu et al., 2022); QLoRA reduces memory further by combining quantized backbones with low-rank updates (Dettmers et al., 2023). These methods are simple and effective, but standard LoRA uses the same rank for every token and domain, even when the required adaptation budget varies across inputs. Exist methods make this

budget more flexible, AdaLoRA reallocates rank using parameter-importance scores (Zhang et al., 2023), while Routed adapters and mixture-of-experts models assign inputs to different parameter subsets (Jiang et al., 2024). However, these methods usually route from dense hidden states. In multi-domain adaptation, dense gates must separate superposed semantic factors from coordinates not designed for routing. Steering-Conditioned LoRA changes the control space: low-rank updates are routed from sparse autoencoder features rather than dense hidden-state coordinates.

Sparse Autoencoders and Representation Orientation. Mechanistic interpretability seeks internal variables that explain the model's behavior. Sparse autoencoders (SAEs) decompose dense activations into sparse latent features, which are typically more interpretable and more monosemantic than individual neurons (Bricken et al., 2023; Cunningham et al., 2024). Most work with SAEs uses these features after training: activations are encoded, inspected, labeled, or subjected to interventions to understand a fixed model. Representation Engineering similarly demonstrates that latent directions can causally guide model outputs (Zou et al., 2023). Orientation-Conditional LoRA extends this idea to fine-tuning: the features of the SAE determine which low-dimensional updates are applied during adaptation, as these features provide a candidate coordinate system for identifying which concepts are strongest.

Adapting models with SAEs. Recent work has begun to combine SAEs with model optimization. One line improves SAE architectures or objectives to obtain better sparsity–fidelity trade-offs (Rajamanoharan et al., 2024). Another adapts a language model around a fixed SAE, using LoRA to reduce the loss gap caused by inserting SAE reconstructions into the forward pass (Chen et al., 2025). This shows that SAEs and PEFT can improve the interpretability–performance frontier, not only support post-hoc analysis. Steering-Conditioned LoRA takes the complementary direction. It does not adapt the model to tolerate an inserted SAE reconstruction. Instead, it keeps the SAE frozen and uses its features as the routing substrate for LoRA. Thus, the SAE is not a reconstruction bottleneck; it is a sparse control space for conditional adaptation.

3. Steering-Conditioned Low-Rank Adaptation

Steering-Conditioned LoRA introduces a control mechanism based on sparse autoencoder (SAE) features. By extracting highly interpretable and disentangled representations from a frozen base model, these SAE features serve as the sole conditioning variables to dynamically dictate two key aspects of the low-rank adaptation process: which specialized LoRA modules are activated, and the exact representational capacity (effective rank) allocated to each update.

3.1. Frozen SAE Feature Extraction

The pipeline begins by mapping dense, entangled representations into a dedicated control space. For each run, we train SAEs offline on the activations of the frozen base model. The SAE encoder E projects a hidden state $h \in \mathbb{R}^{d_{\text{model}}}$ into a higher-dimensional latent space d_{sae}. To enforce sparsity and extract the most salient concepts, we apply a TopK operation:

$$z = \mathrm{TopK}(E(h)), \qquad z \in \mathbb{R}^{d_{\text{sae}}} \tag{1}$$

where TopK keeps the k largest activations and sets the rest to zero. This sparse vector z acts as our conceptual fingerprint for the token and is the sole conditioning signal for routing and rank allocation.

Instead of routing tokens based on dense activations—which can be noisy—we use z to navigate a bank of N_e LoRA specialists, $\mathcal{E} = \{e_1, e_2, \ldots, e_{N_e}\}$. A routing network predicts the optimal distribution over these experts:

$$\mathbf{g} = f_{\text{gate}}(z) = \mathrm{SparseSoftmax}(W_2\,\mathrm{GELU}(W_1 z)) \tag{2}$$

where $\mathbf{g}$ is the resulting gate distribution, W_1 and W_2 are learnable projection matrices, and $\mathrm{SparseSoftmax}$ keeps and renormalizes the top-K logits. Consequently, each token engages only a highly targeted subset of experts. Furthermore, if broad domain characteristics are known, we can optionally bias this token-level routing toward domain-aligned specialists using cosine alignment with domain prototypes (see Appendix A.1).

3.2. Token-Dependent Effective Rank Allocation

Identifying the right experts is only half the equation; allocating the appropriate computational budget is equally critical. While standard LoRA assigns a static rank across all tokens, our method dynamically scales the representational capacity based on token complexity. Using the same sparse feature vector z, a rank controller generates a continuous mask $\mathbf{m} \in [0, 1]^r$

via:

$$\mathbf{m} = \mathrm{cumprod}\left(\sigma(W_r z + \mathbf{b}_r)\right) \tag{3}$$

where W_r and $\mathbf{b}_r$ are learnable parameters, σ is the sigmoid activation, and $\mathrm{cumprod}$ is the cumulative product operator. This formulation enforces a monotonically non-increasing property ($m_1 \geq m_2 \geq \cdots \geq m_r$). This structural prior forces the model to prioritize earlier dimensions, effectively nesting lower-rank subspaces within higher ones.

During the forward pass for an input $\mathbf{x}$, the chosen experts and their allocated capacities are seamlessly fused:

$$\Delta W\mathbf{x} = \sum_{i=1}^{N_e} g_i \, B_i \left((A_i\mathbf{x}) \odot \mathbf{m}\right) \tag{4}$$

where $A_i \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B_i \in \mathbb{R}^{d_{\text{out}} \times r}$ denote the projection matrices for expert i, g_i is its routing weight, and $\odot$ represents element-wise multiplication. The continuous effective rank allocated to a token t is quantified as $r_{\text{eff}}(t) = \sum_{j=1}^{r} m_{t,j}$. This creates a highly efficient adaptive computation mechanism: functionally simple tokens expend minimal capacity ($r_{\text{eff}} \to 0$), whereas complex ones fully leverage the subspace ($r_{\text{eff}} \to r$).

3.3. Training Objective

To ensure this dynamic routing and scaling behaves as intended, the system requires carefully balanced regularization. We train Steering-Conditioned LoRA using a joint objective $\mathcal{L}$ that augments the standard next-token cross-entropy loss ($\mathcal{L}_{\text{task}}$) with six structural regularizers, scaled by hyperparameters $\alpha_1, \ldots, \alpha_6$:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \alpha_1\mathcal{L}_{\text{pres}} + \alpha_2\mathcal{L}_{\text{steer}} + \alpha_3\mathcal{L}_{\text{sparsity}} + \alpha_4\mathcal{L}_{\text{budget}} + \alpha_5\mathcal{L}_{\text{load}} + \alpha_6\mathcal{L}_{\text{div}} \tag{5}$$

Each term serves a specific architectural purpose. The preservation ($\mathcal{L}_{\text{pres}}$) and steering ($\mathcal{L}_{\text{steer}}$) losses act as counterweights: the former anchors general capabilities to prevent catastrophic forgetting, while the latter pulls domain-specific representations toward target prototypes to encourage adaptation. Simultaneously, the sparsity ($\mathcal{L}_{\text{sparsity}}$) and budget ($\mathcal{L}_{\text{budget}}$) terms optimize adaptive computation by penalizing routing entropy and excessive rank usage. Finally, to maximize the efficiency of our expert bank, the load-balancing ($\mathcal{L}_{\text{load}}$) and diversity ($\mathcal{L}_{\text{div}}$) losses prevent token clustering and guarantee orthogonal parameter subspaces. Complete mathematical formulations for all regularization terms are detailed in Appendix A.2.

3.4. Specialist Lifecycle

Finally, to maintain robustness under evolving or heterogeneous data streams, the specialist pool is not static but undergoes continuous lifecycle management. Each expert tracks an exponential moving average (EMA) of its usage, alongside a prototype vector representing the centroid of its assigned sparse features z. This tracking enables a self-regulating system: dormant specialists are spawned when the model encounters novel concepts (i.e., routing confidence is low, $\max(\mathbf{g}) < \tau_{\text{spawn}}$), while experts with sustained low usage are pruned. If two specialists develop highly similar prototypes, they are merged via usage-weighted averaging to consolidate redundant capacity (see Appendix A.3).

4. Results

In this section, we study how Steering-Conditioned LoRA improves the adaptation–retention frontier: increasing target-domain performance, preserving general capabilities, and reducing interference under sequential adaptation. Full details regarding our mathematical formulation of our training objective and merging operations is detailed in Appendix A, while the complete experimental setup and hyperparameters can be found in Appendix B.

4.1. Adaptation–Retention Frontier

We evaluate whether Steering-Conditioned LoRA can enhance target-domain specialization without causing catastrophic forgetting of general capabilities. We categorize our suite into: (i) **Instruction-following** (IFEval, AlpacaEval 2); (ii) **General-knowledge** (HellaSwag, WinoGrande, ARC-C, TQA); and (iii) **Hard reasoning** (MMLU-Pro, GSM8K, GPQA). Figure 1 illustrates this adaptation–retention frontier, with detailed benchmark scores reported in Table 1. Across all evaluated scales, our full method pushes the Pareto frontier beyond standard PEFT baselines. The improvements are most pronounced in instruction-following (averaging +0.016 to +0.052 gains) and general-knowledge retention (+0.017 to

Figure 1. (Left) Performance on the specialized target domain across model scales. (Right) General score aggregated over instruction-following, general-knowledge, and hard-reasoning benchmarks.

+0.023) over base models. This demonstrates that sparse feature conditioning successfully guides specialization without indiscriminately degrading pre-trained knowledge.

Ablation studies on constrained variants isolate the contributions of our core mechanisms. The *routing-only* variant tests feature-conditioned specialist selection without token-dependent rank, while the *dynamic-rank* variant evaluates adaptive capacity allocation without routing. The superior performance of the full method confirms these components are complementary: routing localizes *where* adaptation occurs, while dynamic rank calibrates *how much* capacity is applied. However, the results also reveal a distinct boundary condition. While highly effective for knowledge and alignment, Steering-Conditioned LoRA yields marginal or negative gains on complex reasoning tasks (MMLU-Pro, GSM8K) at larger scales (14B and 32B). Consequently, we interpret the method as a robust mechanism for knowledge adaptation and capability retention, rather than a universal reasoning enhancer.

4.2. Sequential Adaptation and Catastrophic Forgetting

To evaluate robustness against catastrophic forgetting, we simulate sequential domain adaptation. Since temporal interference is path-dependent, we train models on ordered sequences $A \to B \to C$ and measure performance degradation on earlier domains following later updates.

Let $S_d^{(t)}$ be the evaluation score on domain d after training stage t. We quantify strictly positive forgetting (effectively ignoring backward transfer) as:

$$F_{A\to B} = \max\left(0, S_A^{(A)} - S_A^{(B)}\right), \tag{6}$$

$$F_{A\to C} = \max\left(0, \max(S_A^{(A)}, S_A^{(B)}) - S_A^{(C)}\right), \tag{7}$$

$$F_{B\to C} = \max\left(0, S_B^{(B)} - S_B^{(C)}\right). \tag{8}$$

We test two distinct three-stage trajectories:

$$S_1 : \texttt{medicine} \to \texttt{code} \to \texttt{math}, \qquad S_2 : \texttt{general\_instruction} \to \texttt{medicine} \to \texttt{math}.$$

S_1 shifts across disjoint, specialized domains, whereas S_2 transitions from broad instruction-following to narrow tasks.

As illustrated in Figure 2, Steering-Conditioned LoRA reduces mean positive forgetting across 30 transition measurements from 0.343 (standard LoRA) to 0.298 (-13.1%). This mitigation is substantially more pronounced in the broad-to-narrow sequence (S_2), where mean forgetting drops from 0.244 to 0.151 (-38.3%).

Size	Variant	IFEval	MMLU	MMLU-Pro	GSM8K	AlpacaEval 2	HellaSwag	WinoGrande	ARC-C	TQA MC1	TQA MC2	GPQA
0.6B	Base	0.556	0.285	0.142	0.126	0.185	0.000	0.512	0.253	0.301	0.473	0.232
	Routing-only	0.553	0.280	0.139	0.122	0.181	0.000	0.508	0.251	0.299	0.470	0.230
	Dynamic-rank	0.562	0.290	0.140	0.128	0.192	0.018	0.525	0.255	0.305	0.478	0.235
	Full method	**0.571**	**0.295**	**0.148**	**0.130**	**0.201**	**0.022**	**0.540**	**0.268**	**0.312**	**0.495**	**0.242**
4B	Base	0.723	0.485	0.268	0.124	0.295	0.000	0.546	0.354	0.348	0.512	0.232
	Routing-only	0.726	0.482	0.265	0.121	0.301	0.313	0.548	0.360	0.351	0.515	0.235
	Dynamic-rank	0.735	0.494	0.270	0.128	0.318	0.329	0.559	0.372	0.358	0.524	0.239
	Full method	**0.748**	**0.502**	**0.276**	**0.133**	**0.334**	**0.341**	**0.566**	**0.381**	**0.364**	**0.531**	**0.246**
8B	Base	0.801	0.605	0.343	0.572	0.372	0.556	0.657	0.492	0.421	0.603	0.318
	Routing-only	0.806	0.602	0.339	0.561	0.379	0.558	0.660	0.497	0.423	0.606	0.319
	Dynamic-rank	0.814	0.612	0.345	0.578	0.395	0.565	0.671	0.506	0.429	0.612	0.325
	Full method	**0.827**	**0.620**	**0.349**	**0.584**	**0.411**	**0.573**	**0.682**	**0.517**	**0.437**	**0.621**	**0.332**
14B	Base	0.842	0.657	**0.392**	**0.709**	0.425	0.582	0.689	0.521	0.446	0.629	0.341
	Routing-only	0.846	0.653	0.386	0.694	0.431	0.584	0.690	0.525	0.448	0.631	0.340
	Dynamic-rank	0.855	0.662	0.391	0.710	0.447	0.591	0.701	0.534	0.455	0.638	0.346
	Full method	**0.868**	**0.669**	0.390	0.704	**0.461**	**0.598**	**0.711**	**0.543**	**0.462**	**0.646**	**0.352**
32B	Base	0.881	0.704	**0.430**	**0.770**	0.478	0.598	0.703	0.553	0.466	0.652	0.366
	Routing-only	0.884	0.701	0.421	0.752	0.482	0.600	0.704	0.557	0.468	0.654	0.364
	Dynamic-rank	0.892	0.710	0.428	0.766	0.497	0.606	0.715	0.565	0.474	0.661	0.371
	Full method	**0.903**	**0.716**	0.426	0.761	**0.511**	**0.612**	**0.724**	**0.573**	**0.481**	**0.669**	**0.376**

Table 1. Downstream language modeling evaluations on Qwen3 models from 0.6B to 32B with Base, Routing-only, Dynamic-rank and Full method variants. Best results for each size are **bolded**. All models are trained with the same procedure.

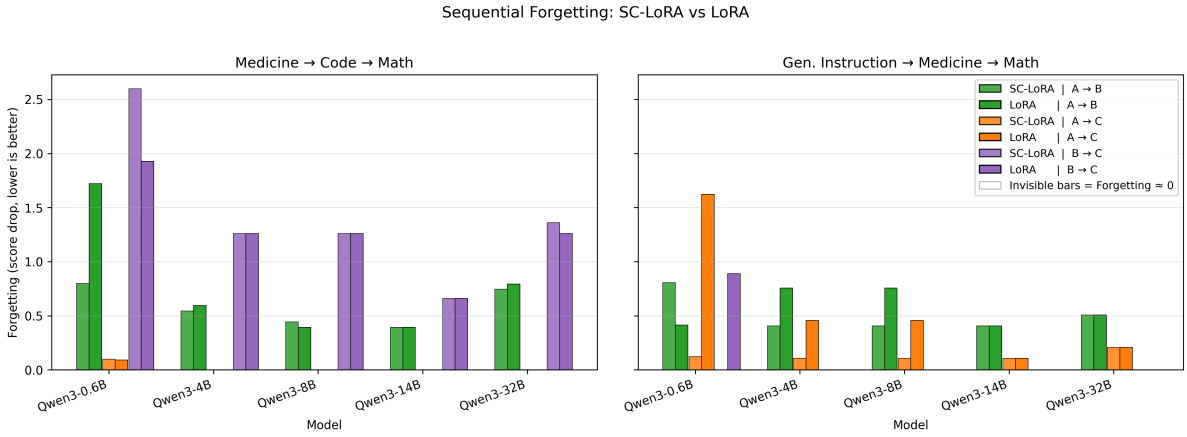

Figure 2. Sequential forgetting evaluations on Qwen3 models from 0.6B to 32B. The plots compare the score drop of Steering-Conditioned LoRA versus standard LoRA across consecutive training phases ($A \to B$, $A \to C$, $B \to C$). Lower is better.

This indicates that sparse feature conditioning effectively preserves broad, reusable capabilities. Domain prototypes provide a stable coordinate system, directing later updates into specialized subspaces without perturbing earlier shared structures. Conversely, the fully disjoint sequence (S_1) presents a harder interference profile, with both methods showing comparable forgetting. Thus, our method best mitigates temporal interference when initial broad adaptations provide a structural foundation for subsequent targeted updates.

5. Causal and Mechanistic Analysis

Given the empirical improvements on the adaptation–retention frontier (Section 4), we now investigate the underlying mechanisms. Our central hypothesis is that sparse feature geometry provides a superior control space for low-rank adaptation compared to dense hidden states. If the gains stem merely from added capacity, baseline gating methods should match our performance. Conversely, if the structured geometry of sparse features is responsible, disrupting this geometry should induce severe degradation.

5.1. Sparse Feature Geometry Drives Adaptation

To isolate the causal role of sparse feature conditioning, we design five interventions that preserve the adapter parameter budget and training data while systematically corrupting the routing signal. We compare our full method against the

following controls: (1) **Dense gate**, which replaces sparse features with dense hidden states; (2) **Random gate**, which assigns routes via uniform sampling; (3) **Shuffled features**, which preserves global sparsity statistics but destroys feature identity across examples; (4) **Inverted signal**, which routes updates away from the originally activated features; and (5) **Fixed-rank routing**, which preserves the sparse routing mechanism but disables token-dependent rank allocation.

Variant	Conditioning Signal	Dynamic Rank	Target Score
Steering-Conditioned LoRA	Sparse features	Yes	**0.363**
Fixed-rank routing	Sparse features	No	0.320
Shuffled features	Shuffled sparse features	Yes	0.241
Dense hidden-state gate	Dense hidden state	Yes	0.168
Inverted signal	Inverted sparse features	Yes	0.151
Random gate	Random assignment	Yes	0.125

Table 2. Causal control evaluations for sparse-conditioned adaptation. All variants preserve the same adapter parameter budget and training data. Best result is **bolded**.

These controlled interventions reveal a clear dependency structure (Table 2). Disabling dynamic rank allocation yields a modest drop of 11.8% (from 0.363 to 0.320), indicating that it serves as a useful refinement rather than a core driver. In contrast, corrupting the routing signal causes substantial degradation: replacing sparse features with dense states or random assignments reduces performance by 53.7% and 65.6%, respectively. This demonstrates that generic conditional capacity or stochastic routing cannot substitute for structured sparse signals, intermediate scores under shuffled (0.241) and inverted (0.151) controls further isolate the necessity of both feature identity and directional alignment. Together, these results establish that adaptation efficacy is primarily driven by the geometric structure of the sparse features, rather than by auxiliary capacity or unstructured conditioning.

5.2. Preventing Routing Collapse

To understand why dense gating fails, we analyze expert specialization. Figure 3 visualizes the top-1 routing frequency across domains for both gating mechanisms.

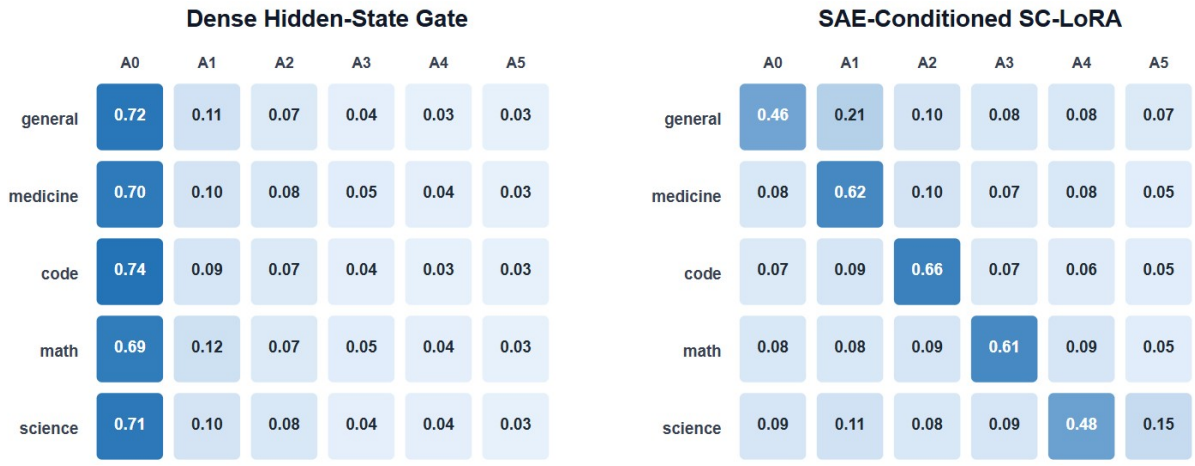

Figure 3. Left: Dense gating collapses to a single expert (A0) across all domains. *Right:* SAE-conditioned routing produces a diagonal, domain-aligned pattern, where each domain consistently activates a distinct expert (A0–A4). Rows denote input domains; columns denote experts.

The dense hidden-state gate exhibits severe routing collapse, concentrating over 69% of the routing mass on a single adapter across all domains. The gate fails to partition the representational space. By contrast, sparse feature conditioning yields a distinctly diagonal, domain-aligned specialization pattern. Because sparse projections map hidden states into an orthogonalized space, domain-relevant activations become linearly separable. The router can subsequently map these recurring sparse patterns to stable, independent experts, successfully isolating conflicting domain updates.

5.3. Feature Drift Localization

Finally, we assess how targeted adaptation perturbs the underlying representation. For a given domain d, let $\mathbf{z}_{\text{base}}$ and $\mathbf{z}_{\text{ft}}$ denote the sparse feature vectors from the base and adapted models, respectively. We quantify representational drift as:

$$\Delta \mathbf{z}_d = \|\mathbf{z}_{\text{ft}} - \mathbf{z}_{\text{base}}\|_2 . \tag{9}$$

While standard LoRA applies a uniform low-rank update that inadvertently perturbs general-capability features, Steering-Conditioned LoRA localizes these modifications. By conditioning on sparse features, domain-specific inputs engage targeted specialists, whereas general-domain inputs expend minimal effective rank ($r_{\text{eff}} \to 0$) or bypass modified experts entirely. This targeted drift corroborates our continual learning results (Section 4.2): preserving a stable sparse coordinate system minimizes interference, allowing broad capabilities to remain anchored while narrow domains specialize.

6. Conclusion

We presented Steering-Conditioned LoRA, a parameter-efficient adaptation framework that uses frozen Sparse Autoencoder features as the control space for low-rank updates. Our results show that routed adaptation depends not only on adapter capacity, but also on the representation used to control it. By routing specialists and allocating rank from sparse semantic features, Steering-Conditioned LoRA makes adaptation more selective, improves retention, and reduces interference across sequential domains. The method is most useful when Sparse Autoencoder features separate the target domains, which explains its gains on instruction following and general knowledge and its weaker effect on hard reasoning and highly overlapping domains. These findings suggest a simple lesson for conditional adaptation: better adapters may not be enough; better control coordinates are also needed.

Limitations. Steering-Conditioned LoRA is evaluated on a single model family (Qwen3, 0.6B–32B), five domain mixtures, and two ordered three-stage sequences; the gains depend on the quality of the per-run SAEs and concentrate when broad capabilities precede narrower updates rather than between adjacent specialized domains. Activation harvesting and SAE training add a fixed preprocessing cost that is amortized over multi-domain workloads. Appendix D expands on these scope conditions.

References

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Denison, A., Askell, A., Lasenby, R., et al. Towards monosemanticity: Extracting interpretable features from large language models. *Transformer Circuits Thread*, 2023.

Chen, M., Engels, J., and Tegmark, M. Low-rank adapting models for sparse autoencoders, 2025. URL https://arxiv.org/abs/2501.19406.

Cunningham, H., Ong, E., Ewart, A., Nanda, P., and Turner, A. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=35psQ8eqE0.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Ding, N., Lv, X., Wang, Q., Chen, Y., Zhou, B., Liu, Z., and Sun, M. Sparse low-rank adaptation of pre-trained language models, 2023. URL https://arxiv.org/abs/2311.11696.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. arXiv preprint arXiv:2101.03961.

Flynn, N. Compass: Continual multilingual peft with adaptive semantic sampling, 2026. URL https://arxiv.org/abs/2604.20720.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, F., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017a. ISSN 1091-6490. doi: 10.1073/pnas.1611835114. URL http://dx.doi.org/10.1073/pnas.1611835114.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 114(13):3521–3526, 2017b.

Lialin, V., Deshmukh, V., Gallego, K., and Rumshisky, A. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*, 2023.

Liang, Y.-S. and Li, W.-J. Inflora: Interference-free low-rank adaptation for continual learning, 2024. URL https://arxiv.org/abs/2404.00228.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

Mirzadeh, I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., and Farajtabar, M. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2025. URL https://arxiv.org/abs/2410.05229.

Rajamanoharan, S., Lieberum, T., Sonnerat, N., Conmy, A., Varma, V., Kramár, J., and Nanda, N. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders, 2024. URL https://arxiv.org/abs/2407.14435.

Sun, M., Liu, W., Luan, J., Gao, P., and Wang, B. Mixture of diverse size experts. In Dernoncourt, F., Preoţiuc-Pietro, D., and Shimorina, A. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 1608–1621, Miami, Florida, US, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-industry.118. URL https://aclanthology.org/2024.emnlp-industry.118/.

Templeton, A., Conerly, T., Marcus, J., Lindsey, J., Bricken, T., Chen, B., Pearce, A., Citro, C., Ameisen, E., Jones, A., Cunningham, H., Turner, N. L., McDougall, C., MacDiarmid, M., Freeman, C. D., Sumers, T. R., Rees, E., Batson, J., Jermyn, A., Carter, S., Olah, C., and Henighan, T. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Turner, A. M., Thiergart, L., Leech, G., Udell, D., Vazquez, J. J., Mini, U., and MacDiarmid, M. Steering language models with activation engineering, 2024. URL https://arxiv.org/abs/2308.10248.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., Li, T., Ku, M., Wang, K., Zhuang, A., Fan, R., Yue, X., and Chen, W. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL https://arxiv.org/abs/2406.01574.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=lq62uWRJjiY.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A., Chen, Z., Le, Q., and Laudon, J. Mixture-of-experts with expert choice routing, 2022. URL https://arxiv.org/abs/2202.09368.

Zou, A., Phan, L., Chen, S., Campbell, J., Braun, P., Kosmaczewski, V., Oughton, A., Zhao, R., Basu, E., Franks, R., et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023.

A. Methodology Details

A.1. Domain Prototype Routing

When domain prototypes are available, the SAE-conditioned gate score g_i for specialist i can be refined by cosine alignment. Let μ_d be the mean SAE feature vector for domain d, and z be the sparse feature vector for the current token. The refined gate score $\tilde{g}_i$ is computed as:

$$\tilde{g}_i = g_i \cdot \left(\frac{z \cdot \mu_d}{\|z\| \|\mu_d\|} \right)^{\gamma} \tag{10}$$

where γ is a hyperparameter controlling the strength of the alignment bias, and $\| \cdot \|$ denotes the ℓ_2 norm. This encourages tokens with similar sparse features to reuse related domain-aligned specialists.

A.2. Detailed Training Objective

The complete training objective incorporates six regularization terms. The preservation loss anchors general capabilities to the frozen base model:

$$\mathcal{L}_{\text{pres}} = \frac{1}{N} \sum_{t=1}^{N} \left\| (h_{\text{ft}}^{(t)} - h_{\text{base}}^{(t)}) \odot \text{mask}_{\text{general}}^{(t)} \right\|_2^2 \tag{11}$$

where N is the batch token count, $h_{\text{ft}}^{(t)}$ and $h_{\text{base}}^{(t)}$ are hidden states from the adapted and base models for token t, and $\text{mask}_{\text{general}}^{(t)}$ selects dimensions responsible for general capabilities.

The steering loss aligns domain-specific representations with the target domain prototype:

$$\mathcal{L}_{\text{steer}} = \frac{1}{N} \sum_{t=1}^{N} \left\| (h_{\text{ft}}^{(t)} - \mu_d) \odot \text{mask}_{\text{domain}}^{(t)} \right\|_2^2 \tag{12}$$

where μ_d is the sparse feature centroid for domain d, and $\text{mask}_{\text{domain}}^{(t)}$ isolates the domain-relevant dimensions.

To prevent routing collapse, the sparsity loss bounds the entropy of the routing distribution $\mathbf{g}$:

$$\mathcal{L}_{\text{sparsity}} = \text{ReLU}(H_{\min} - H(\mathbf{g})) + 0.5 \cdot \text{ReLU}(\max(\mathbf{g}) - 0.9) \tag{13}$$

where $H(\cdot)$ calculates the Shannon entropy, and $H_{\min}$ is the target minimum entropy threshold.

The budget loss encourages low effective rank usage by directly penalizing the rank mask elements $m_{t,j}$:

$$\mathcal{L}_{\text{budget}} = \frac{1}{N \cdot r} \sum_{t=1}^{N} \sum_{j=1}^{r} m_{t,j} \tag{14}$$

where r is the maximum rank capacity.

The load-balancing loss prevents token accumulation in a small subset of experts:

$$\mathcal{L}_{\text{load}} = N_e \sum_{i=1}^{N_e} f_i p_i \tag{15}$$

where N_e is the active specialist count, f_i is the fraction of tokens routed to expert i, and p_i is its mean routing probability across the batch.

Finally, the diversity loss ensures specialists capture orthogonal features:

$$\mathcal{L}_{\text{div}} = \|WW^{\top} - I\|_F^2 \tag{16}$$

where $W \in \mathbb{R}^{N_e \times D}$ is a matrix where each row i is the flattened, ℓ_2-normalized parameter vector of specialist i (D being the total parameters per specialist), I is the identity matrix, and $\| \cdot \|_F$ is the Frobenius norm.

A.3. Specialist Merging Operation

If the prototype similarity between two specialists i and j exceeds a threshold τ_{merge}, they are consolidated. Let u_i and u_j denote the exponential moving average (EMA) of their respective token usages. The new parameters A_{new} and B_{new} are computed via usage-weighted averaging:

$$A_{\text{new}} = \frac{u_i A_i + u_j A_j}{u_i + u_j}, \quad B_{\text{new}} = \frac{u_i B_i + u_j B_j}{u_i + u_j} \tag{17}$$

The combined expert occupies the slot of the higher-utilization specialist, while the secondary slot is returned to the dormant pool.

B. Hyperparameters and Reproducibility Details

This appendix consolidates the configuration values, dataset sizes, training schedules, compute environment, and determinism procedures required to reproduce the Steering-Conditioned LoRA experiments reported in the main text. All numerical values are the canonical defaults used in our primary benchmarks; deviations introduced by individual ablation suites are listed explicitly when relevant.

B.1. Experimental Setup

Models and domains. We evaluate Qwen3 models from 0.6B to 32B parameters. The 0.6B model validates the pipeline; the 4B and 8B models support the main comparisons, causal controls, and diagnostics; larger models test whether the adaptation–retention pattern persists across scale. The adaptation corpus consists of five domains: medicine, programming, mathematics, science, and general instruction. We use MedMCQA, CodeAlpaca-20K, NuminaMath-CoT, SciQ, and SlimOrca for these domains, respectively. All examples are converted to a unified chat format, and multiple-choice tasks follow a standardized template.

Baselines, metrics and training protocol. We compare against Full Fine-Tuning (Full-SFT), LoRA (Hu et al., 2022), and QLoRA (Dettmers et al., 2023). We also evaluate two constrained variants: *Routing-only*, which uses SAE-conditioned routing with fixed-rank adapters, and *Dynamic-rank*, which uses input-dependent effective rank without a specialist bank. The full method combines SAE-conditioned routing, token-dependent rank allocation, and preservation regularization. We report three outcomes: target-domain benchmarks for specialization, general-domain benchmarks for retained capabilities after adaptation, and sequential forgetting, defined as positive score drops on earlier domains after later updates. Unless stated otherwise, all methods use matched data and optimization budgets on the medium-scale mixture, with fixed regularization weights across experiments. For Steering-Conditioned LoRA, SAEs are trained offline on base-model activations, frozen, and used only to condition routing and rank allocation.

B.2. Steering-Conditioned LoRA Architectural Configuration

Table 3 reports the architectural hyperparameters of the Steering-Conditioned LoRA adapter bank, the gate, and the dynamic-rank controller. Targets for low-rank injection are the standard attention and MLP projections of each Qwen3 transformer block, namely `q_proj`, `k_proj`, `v_proj`, `o_proj`, `up_proj`, `down_proj`, and `gate_proj`.

B.3. Sparse Autoencoder Training

A separate Sparse Autoencoder is trained per instrumented layer on activations harvested from the frozen base model, following the protocol of Section 3.1. After training, every SAE is frozen (`eval()` mode and `requires_grad = False` on all parameters) and reused unchanged during Steering-Conditioned LoRA optimization. Hyperparameters are summarized in Table 4.

B.4. Optimization Protocol

The base model is kept frozen throughout Steering-Conditioned LoRA training; only the adapter bank, the gate, and the rank controller receive gradients. Optimization uses AdamW with the settings in Table 5, with bfloat16 mixed precision enabled by default and gradient checkpointing applied to every transformer block. Multi-GPU runs are sharded with Fully

Table 3. Defaults used across all evaluated models scales unless overridden by the ablation suites described in Section 5.1.

Parameter	Value	Description
num_specialists	4	Total adapter slots per wrapped layer
initial_active_specialists	2	Active adapters at the start of training
min_active_specialists	2	Lower bound enforced by the lifecycle manager
max_rank	16	Maximum LoRA rank per specialist (upper bound for $\mathbf{m}$)
lora_alpha	16	LoRA scaling factor
lora_dropout	0.0	Dropout applied to LoRA branch
gate_hidden_dim	256	Hidden dimension of the gate MLP f_{gate}
rank_hidden_dim	256	Hidden dimension of the rank controller f_{rank}
controller_topk	2	Top-K used by SparseSoftmax in Eq. 2
retrieval_topk	2	Top-K for prototype retrieval in routing
layer_stride	4	Distance between SAE-instrumented transformer blocks
max_hook_layers	8	Maximum number of layers wrapped per model

Table 4. **SAE training hyperparameters.**

Parameter	Value	Description
expansion_factor	4	Latent dim $d_{\text{sae}} = 4 \cdot d_{\text{model}}$
epochs	3	Passes over harvested activations
batch_size	4096	Activation rows per SGD step
lr	1×10^{-3}	AdamW learning rate
lambda_sparse	1×10^{-3}	Weight on sparsity penalty
lambda_aux	1×10^{-3}	Weight on dead-feature + decoder-norm penalty
sparse_mode	l1	L_1 penalty on latent activations
dead_feature_threshold	1×10^{-4}	Mean-activation cutoff for the dead-feature penalty
decoder_norm_penalty	1×10^{-3}	Penalty enforcing unit-norm decoder atoms
normalize_inputs	true	Standardize activations before encoding
probe_samples_per_domain	1024	Activation rows harvested per domain per layer
TopK at inference	32	Sparsity level applied at adaptation time

Sharded Data Parallel (FSDP) using sharded state dicts and a transformer-based auto-wrap policy. Non-finite losses trigger an automatic optimizer-state sanitization with patience 8, so transient numerical instabilities do not require manual restarts.

Table 5. **Optimization and training protocol.** Defaults used in the primary benchmarks; multi-GPU runs use FSDP with sharded state dicts.

Parameter	Value	Description
Optimizer	AdamW	On trainable Steering-Conditioned LoRA parameters only
Learning rate	5×10^{-5}	Constant schedule
Weight decay	0.0	Disabled to limit drift on small adapter set
Adam ϵ	1×10^{-6}	Numerical stability term
Max grad norm	0.5	Global gradient clipping
Train batch size	2	Per device; effective batch via accumulation/FSDP
Eval batch size	2	Per device
Max sequence length	1024	Truncated from the right tail of each example
Mixed precision	bfloat16	Enabled across forward and backward passes
Gradient checkpointing	enabled	Reduces activation memory
Replay ratio	0.2	Fraction of replay tokens per phase batch
Replay buffer size	128	Examples retained per prior domain

B.5. Loss Weights

The combined objective of Section 3.3 is instantiated with the weights in Table 6. The task and representation-shaping terms ($\mathcal{L}_{\text{task}}$, $\mathcal{L}_{\text{pres}}$, $\mathcal{L}_{\text{steer}}$) carry unit weight, while the structural regularizers act as gentle priors. We do not tune these weights per task; an explicit ablation suite sweeps each weight independently to confirm robustness.

Table 6. **Loss-weight schedule.** The numbered subscripts (α_1–α_6) follow the convention introduced in Section 3.3.

Symbol	Value	Loss term
λ_{task}	1.0	Cross-entropy task loss $\mathcal{L}_{\text{task}}$
α_1	1.0	Preservation loss $\mathcal{L}_{\text{pres}}$
α_2	1.0	Steering loss $\mathcal{L}_{\text{steer}}$
α_3	0.01	Gate sparsity $\mathcal{L}_{\text{sparsity}}$
α_4	0.001	Average-rank budget $\mathcal{L}_{\text{budget}}$
α_5	0.05	Load balance $\mathcal{L}_{\text{load}}$
α_6	0.001	Specialist diversity $\mathcal{L}_{\text{div}}$

B.6. Lifecycle and Memory-Policy Thresholds

Spawning, merging, and pruning are governed by exponential moving averages of expert usage and prototype similarities, as described in Section 3.4. Table 7 reports the canonical thresholds; an additional ablation suite sweeps each value independently to confirm robustness.

Table 7. **Specialist lifecycle and memory-policy thresholds.**

Parameter	Value	Description
`memory_ema_decay`	0.95	EMA decay for usage_ema$_i$
`spawn_confidence_threshold` (τ_{spawn})	0.45	Activate dormant slot when $\max(\mathbf{g}) < \tau_{\text{spawn}}$
`merge_similarity_threshold` (τ_{merge})	0.92	Fuse experts with prototype cosine $\geq \tau_{\text{merge}}$
`prune_usage_threshold` (τ_{prune})	0.01	Deactivate experts with usage_ema $< \tau_{\text{prune}}$
`memory_policy_interval`	50	Steps between lifecycle scans
`memory_compaction_warmup_steps`	400	Warm-up steps before lifecycle is engaged
`enable_memory_policy`	enabled	Toggles the entire lifecycle system

B.7. Multi-Domain Adaptation Corpus

The five-domain adaptation corpus is built by a deterministic data-preparation pipeline. Each domain provides train and eval splits, and three mixture sizes are produced—small, medium, and full—so that pilot, primary, and stress-test runs draw from comparable but progressively larger pools. All examples are normalized to the chat templates of Appendix C, so prompt formatting is identical across baselines and Steering-Conditioned LoRA variants. Multiple-choice answers are deterministically permuted using SHA-256 hashes of the example payload concatenated with a fixed global seed.

Table 8. **Per-domain sources and per-mixture sizes.** Each domain contributes the same number of training examples per mixture; evaluation splits are fixed at 1,000 examples per domain across all mixtures. The full mixture exhausts the upstream pool of each source after deduplication and template normalization.

Domain	Source dataset	Small (train)	Medium (train)	Full (train)
medicine	MedMCQA	3,000	10,000	all available
code	CodeAlpaca-20K	3,000	10,000	all available
mathematics	NuminaMath-CoT	3,000	10,000	all available
science	SciQ	3,000	10,000	all available
general_instruction	SlimOrca (ChatML)	3,000	10,000	all available
Total per mixture	–	15,000	50,000	$\geq$ 250,000
Eval per domain	–	1,000	1,000	1,000

The *small* mixture is reserved for pipeline validation (e.g. Qwen3-0.6B), the *medium* mixture is the primary mixture used in Section 5.1, and the *full* mixture is reserved for the scaling.

B.8. Sequential Training Schedules

The retention-and-forgetting analysis (Section 4.2) uses two ordered three-stage schedules:

- S_1: `medicine` → `code` → `math`
- S_2: `general_instruction` → `medicine` → `math`

Each phase is trained on the medium mixture restricted to the active domain, with the same optimization settings as Table 5

and a replay ratio of 0.2 drawing from a 128-example buffer per previously seen domain. The sequential interference metric defined in Section 4.2 is evaluated on the held-out eval splits of all five domains after every phase boundary.

B.9. Compute Resources and Software Environment

All experiments were executed on a single host equipped with two NVIDIA B200 GPUs providing 180 GB of VRAM each, with CUDA-enabled PyTorch. The same hardware is used for every stage of the pipeline, per-run Sparse Autoencoder training, full supervised fine-tuning (Full-SFT), LoRA, QLoRA, and Steering-Conditioned LoRA optimization, so that all comparisons are made under matched device conditions. The Qwen3 variants from 0.6B through 8B parameters fit on a single B200 device at bfloat16 precision with gradient checkpointing, while the 14B and 32B variants are sharded across both B200 GPUs using Fully Sharded Data Parallel with sharded state dicts. Per-layer SAE training fits comfortably on a single B200 for every evaluated scale, given the chosen expansion factor and batch size. Wall-clock training time scales approximately linearly with model size, with each medium-mixture run completing in single-digit GPU-hours for the smaller scales and tens of GPU-hours for the larger scales. The full sweep of ablation suites (loss, architecture, memory policy, and SAE-conditioning components) summed across seeds 42–46 dominates the project compute budget. Software dependencies (PyTorch, Transformers, Accelerate, Datasets) are pinned in the released code repository.

B.10. Determinism, Seeds, and Artifact Manifests

Each experimental run is fully seeded: the global seed is propagated through Python, NumPy, and PyTorch's CPU and CUDA random number generators. Data preparation uses a fixed global seed, while the primary training and evaluation runs use seeds $\{42, 43, 44\}$ (extended to $\{42, 43, 44, 45, 46\}$ for the robustness suite). Multiple-choice option order is anchored to a SHA-256 hash of the example payload concatenated with the global seed, so all baselines and Steering-Conditioned LoRA variants observe the exact same permutation. For every run, we additionally serialize (i) the parsed run arguments and the resolved configuration, (ii) the per-phase training logs, lifecycle events, and final metrics, and (iii) an environment manifest describing the host architecture, dependency tree, CUDA footprint, and deterministic seeds. Together, these artifacts allow another researcher to reproduce any reported number from the canonical configuration and the recorded seed.

C. Prompt Templates for Multi-Domain Datasets

To ensure full reproducibility and to standardize the generation across distinct structural domains, all inputs are mapped into a strict conversational structure before being applied to the chat template of the respective model. Table 9 outlines the exact text structures used for standardizing the datasets.

Table 9. **Normalized Chat Templates across Domains.** The variables within braces denote fields dynamically populated from the source dataset per example. The deterministic shuffling of the `choices` array mitigates positional bias.

Task Type	User Prompt	Assistant Target
Multiple-Choice	{question} Context: {support} *(Optional)* Choices: A. {choice_0} B. {choice_1} C. {choice_2} D. {choice_3} Answer with the correct option and explanation.	Answer: {answer_label}. {gold_choice} Explanation: {explanation} *(Optional)*
Math / General	{instruction_or_problem}	{solution_or_output}

D. Extended Limitations

While Steering-Conditioned LoRA improves the adaptation–retention trade-off across the evaluated settings, several limitations frame the scope of the present claims. First, the empirical analysis is conducted on a single model family (Qwen3,

0.6B–32B) and on five domain mixtures with two ordered three-stage sequences; results may shift on architectures with different attention patterns, on more domains, or on substantially longer adaptation curricula. Second, the routing signal depends on the quality and stability of the per-run Sparse Autoencoders: degenerate or poorly trained dictionaries (e.g. a dead-feature rate that is too high, or insufficient probe coverage of the target domain) collapse the structured coordinate system that Steering-Conditioned LoRA exploits, as exhibited by the random-projection and shuffled-SAE controls. Third, the gain on highly overlapping specialized domains (e.g. the medicine–code–math sequence) is small, indicating that Steering-Conditioned LoRA primarily helps when broad reusable capabilities precede narrower domain updates rather than between adjacent specialized domains. Fourth, the method introduces non-trivial preprocessing overhead: per-run activation harvesting and SAE training add a fixed cost ahead of fine-tuning, which is amortized over multi-domain or sequential workloads but may not pay off for one-shot adaptation. Finally, the reported results use seeds $\{42, 43, 44\}$ (extended to $\{42, \ldots, 46\}$ in the robustness suite) on the canonical configuration; we do not provide exhaustive sensitivity analyses for every threshold in Table 7, although individual sweeps confirm the configuration is robust within the tested ranges.