EAD-LoRA: Expert-Aware Differential Initialization for MoE-LoRA Frameworks

Anonymous authors
Paper under double-blind review

Abstract

Parameter-efficient fine-tuning (PEFT), particularly Low-Rank Adaptation (LoRA), has become essential for adapting large foundation models to downstream tasks. To enhance representation capabilities, Mixture-of-Experts (MoE) architectures are often combined with LoRA (MoE-LoRA) by employing multiple low-rank adapters. However, existing MoE-LoRA methods uniformly initialize all experts using random distributions, ignoring the intrinsic structure of pretrained models and the anticipated functional divergence among experts. We propose EAD-LoRA, Expert-Aware Differential Initialization for MoE-LoRA frameworks. Unlike uniform initialization, our method uses a brief warm-up phase to collect routing data, then applies Principal Component Analysis (PCA) to initialize each expert's parameters based on its specific input distribution. Extensive experiments demonstrate that EAD-LoRA converges faster and achieves an average accuracy improvement of 1.6% and 5.8% over standard MoE-LoRA on different backbones, while its hybrid variant EAD-LoRA+ brings up to 2.7% and 7.7% improvements.

1 Introduction

The advent of large foundation models has necessitated the development of parameter-efficient fine-tuning (PEFT) techniques (Zhang et al., 2025a; Han et al., 2024) to adapt these models to downstream tasks without the prohibitive cost of full fine-tuning. Low-Rank Adaptation (LoRA) (Hu et al., 2022) has emerged as a predominant PEFT method, which freezes pre-trained weights and injects trainable low-rank matrices into linear layers. While effective, standard LoRA suffers from limited representational capacity, especially for complex and diverse tasks. To address this, Mixture of Experts (MoE) architectures have been integrated with LoRA (MoE-LoRA) (Liu et al., 2023), employing multiple low-rank adapters to enhance model expressiveness. However, a critical yet overlooked challenge in MoE-LoRA is the initialization strategy of these experts. Conventional approaches initialize all LoRA experts identically using random distributions (e.g., Kaiming initialization) (Hayou et al., 2024), ignoring the intrinsic structure of the pre-trained model and the anticipated functional divergence among experts. This uniform initialization often results in slow convergence, suboptimal performance, and insufficient expert specialization, as the optimization process lacks guidance toward distinct feature regions from the outset.

Recent advancements have highlighted the importance of tailored initialization for LoRA. LoRA-GA (Wang et al., 2024) aligns the initial gradient direction of low-rank matrices with that of full fine-tuning, accelerating convergence. PiSSA (Meng et al., 2024) initializes adapters using the principal singular components of the pre-trained weights. However, these methods are designed for single adapters and do not account for the multi-expert dynamics in MoE. In MoE-LoRA, the gate network learns to route different inputs to different experts, implying that each expert should be optimized for a specific subspace of the input distribution. Initializing all experts identically fails to leverage this routing prior, potentially leading to homogeneous experts and inefficient training.

In this paper, we propose a novel **E**xpert-**A**ware **D**ifferential Initialization for MoE-LoRA frameworks (EAD-LoRA). Our core innovation is a data-driven, expert-specific initialization that leverages the routing behavior of the gate network to tailor each expert's low-rank matrices. EAD-LoRA operates in two phases:

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.

Phase 1: Hidden State Collecting We begin with a short warm-up phase consisting only of forward passes, during which the gate network is active but all LoRA parameters remain frozen. During this stage, we gather the hidden states of tokens that are routed to each expert. This allows us to capture an initial data distribution handled by each expert.

Phase 2: Expert-Aware Differential Initialization For every expert, we apply Principal Component Analysis (PCA) to the hidden states accumulated during Phase 1. The resulting principal components are then used to initialize that expert's LoRA down-projection matrix A_i, while the corresponding up-projection matrix B_i retains the standard zero initialization. This strategy ensures that each expert's adaptive capacity is aligned from the outset with the dominant variation patterns in its assigned token subset.

To balance between exploiting the warm-up data prior and maintaining exploratory capacity, we introduce a **hybrid initialization strategy** that combines PCA components with random weights, allowing the adaptation to be primed toward salient feature directions while preserving sufficient flexibility for downstream refinement. We further employ a memory-efficient hidden state collecting and adaptive sampling strategy to handle varying expert activation patterns. These steps ensure robust handling of imbalanced token-to-expert distributions without incurring substantial overhead.

The contributions of our work are summarized as follows:

- We identify the limitation of uniform initialization in MoE-LoRA and propose EAD-LoRA, an expert-aware initialization scheme that uses routing-derived data statistics to tailor each expert's initialization.
- We implement a scalable and memory-efficient warm-up phase with optimized hidden state collection and PCA-based initialization, supporting large-scale models and diverse tasks.
- We introduce a hybrid initialization strategy enhancing robustness to data variability and maintaining exploration capabilities.
- We validate EAD-LoRA through extensive experiments on question answering from different domains, demonstrating improved convergence speed and performance over baselines.

2 Related Works

Parameter-Efficient Fine-Tuning Parameter-Efficient Fine-Tuning (PEFT) adapts large pre-trained models to downstream tasks with reduced costs. Unlike full fine-tuning, PEFT freezes the pre-trained backbone and trains a small set of parameters. Methods include adapter-based (Houlsby et al., 2019; He et al., 2022), prompt-based (Zhang et al., 2023b; Lester et al., 2021), and low-rank adaptation approaches. Low-Rank Adaptation (LoRA) (Hu et al., 2022) is popular for its simplicity, decomposing weight updates into low-rank matrices. Variants like AdaLoRA (Zhang et al., 2023a) and DoRA (Liu et al., 2024) further enhance its expressiveness.

LoRA initialization Initialization plays a crucial role in training stability. Xavier initialization (Glorot & Bengio, 2010) ensures stability under linear activations, while Kaiming initialization (He et al., 2015) extends this to ReLU networks. Orthogonal initialization (Choromanski et al., 2018) improves training stability. While standard LoRA employs a simple initialization scheme (Hayou et al., 2024), recent advances propose more sophisticated methods. PiSSA (Meng et al., 2024) initializes LoRA matrices by decomposing pre-trained weights via SVD, using principal singular components for a strong low-rank approximation. However, its static SVD subset limits adaptability and complicates MoE integration. LoRA-GA (Wang et al., 2024) and LoRA-One (Zhang et al., 2025b) initialize A and B with singular vectors of the full fine-tuning gradient $\nabla_W \mathcal{L}$, yet they are not optimized for MoE-LoRA. GOAT (Fan et al., 2025) introduces an SVD-structured MoE framework where each expert is initialized with distinct singular value segments and adaptively selected via a router, with a theoretical scaling factor aligning optimization to full fine-tuning, achieving state-of-the-art across 25 datasets.

LoRA with Mixture-of-Experts LoRA-MoE integration combines low-rank adaptation with expert specialization. Early implementations like LoRAMoE (Dou et al., 2024) and MixLoRA (Li et al., 2024) used multiple LoRA experts, while subsequent work expanded architectural variety: MixDA

(Diao et al., 2023), MoLA (Gao et al., 2024), with training enhancements like contrastive routing (Luo et al., 2024) and lightweight gating (Zadouri et al., 2023). A key limitation is uniform parameter initialization, where all experts start identically, contradicting the principle of expert specialization. While GOAT addresses this via SVD-based segmentation, it still relies on fixed decomposition. In contrast, our work introduces a data-driven, expert-aware differential initialization that collects routing statistics during a warm-up phase and applies PCA to tailor each expert's parameters to its empirical input distribution.

3 METHODS

3.1 REVIEW OF LORA

Low-Rank Adaptation (LoRA) is a parameter-efficient fine-tuning method that operates on the hypothesis that weight updates during adaptation reside in a low intrinsic dimension. Given a pre-trained weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ in a linear layer, instead of fine-tuning W directly, LoRA constrains its update ΔW to be a low-rank matrix. This is achieved by decomposing ΔW into the product of two smaller matrices:

$$\Delta W = BA, \tag{1}$$

where $B \in \mathbb{R}^{d_{\text{out}} \times r}$, $A \in \mathbb{R}^{r \times d_{\text{in}}}$, and the rank $r \ll \min(d_{\text{out}}, d_{\text{in}})$. During fine-tuning, the original weight W remains frozen, and only the matrices A and B are trained. The forward pass of the adapted layer thus becomes:

$$h = Wx + \Delta Wx = Wx + \alpha/r \cdot BAx, \tag{2}$$

where x is the input to the layer. A scaling factor α/r is often applied to BA to stabilize training, and α is a hyperparameter.

The standard initialization scheme for LoRA sets A using Gaussian or Kaiming initialization and initializes B to zero (Hayou et al., 2024), ensuring that $\Delta W = 0$ at the start of training and the model's initial behavior remains unchanged. This approach significantly reduces the number of trainable parameters from $d_{\text{out}} \times d_{\text{in}}$ to $r \times (d_{\text{out}} + d_{\text{in}})$, making it highly efficient for large-scale models.

3.2 MIXTURE-OF-EXPERTS LORA (MOE-LORA)

Mixture-of-Experts LoRA (MoE-LoRA) extends the standard LoRA framework by incorporating multiple LoRA adapters within a mixture-of-experts (MoE) architecture. Instead of employing a single LoRA adapter per layer, MoE-LoRA introduces N independent LoRA experts $\{E_i\}_{i=1}^{N}$, where each expert E_i consists of its own low-rank matrices B_i and A_i.

The forward pass of a MoE-LoRA layer combines the outputs of these experts through a gating mechanism. Let $g(x) = [g_1(x), g_2(x), \ldots, g_N(x)]$ denote the gate values computed by a gating network $G(\cdot)$, typically implemented as a linear layer followed by a softmax activation:

$$g_i(x) = \frac{\exp(G_i(x))}{\sum_{j=1}^{N} \exp(G_j(x))} \tag{3}$$

The final output of the MoE-LoRA layer is then given by:

$$h = Wx + \sum_{i=1}^{N} g_i(x) \cdot (B_i A_i x). \tag{4}$$

To maintain computational efficiency during training and inference, only the top-k experts with the highest gate values are typically activated for each input token. This sparse activation strategy ensures that the computational cost scales with k rather than N, making the approach feasible even with a large number of experts.

The standard initialization practice in MoE-LoRA initializes all experts identically, typically using the same Kaiming initialization for all A_i matrices and zero initialization for all B_i matrices. However, this uniform initialization strategy fails to account for the intended specialization of different experts, which forms the motivation for our proposed differential initialization approach.

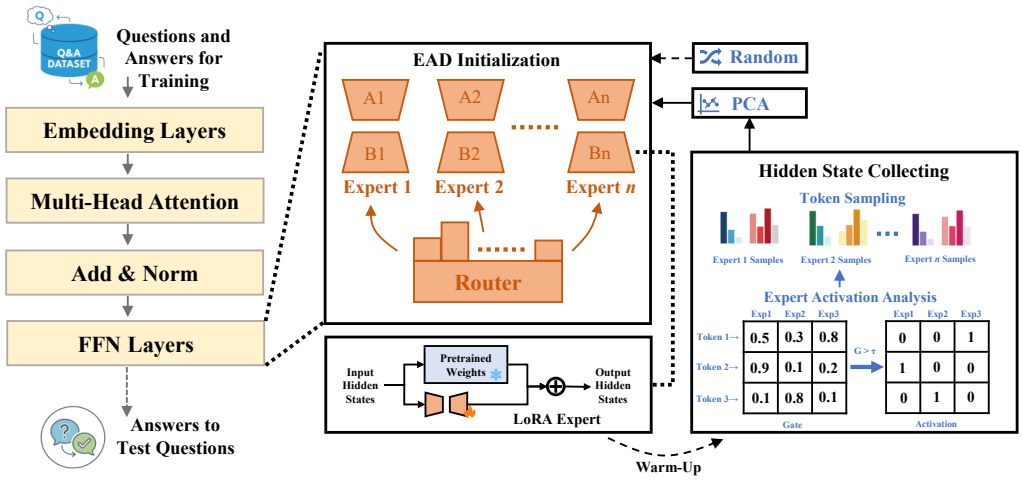


Figure 1: The overall structure of EAD-LoRA, integrating input and output, hidden state collecting, and expert-aware differential (EAD) LoRA initialization. The hidden state collecting phase contains expert activation analysis and token sampling to collect highly activate tokens for each expert.

3.3 Overall Structure of EAD-LoRA

The Expert-Aware Differential LoRA Initialization (EAD-LoRA) framework integrates the previously described components into a cohesive two-phase pipeline. The overall process begins with data collection during a brief warm-up period, proceeds through expert-specific analysis, and culminates in the differentiated initialization of each expert's parameters. The structure of the overall framework is shown in **Figure 1**.

Phase 1: Hidden State Collection A short preliminary warm-up training is conducted with the gate network active but LoRA parameters frozen. The system collects the hidden states of tokens routed to each expert, establishing representative input distributions for full training through the strategic sampling described in Section 3.4.

Phase 2: Expert-Aware Differential Initialization For each expert, PCA is performed on collected hidden states to initialize the LoRA down-projection matrix A_i with principal components. This aligns adaptation with key data features. A hybrid initialization in Section 3.5 combines PCA components with random elements, dynamically weighted by distribution diversity. The process concludes with normalization for training stability.

3.4 Hidden State Collecting

In transformer-based architectures, hidden states represent the intermediate feature representations at each layer. For a given input sequence X, the hidden state h_l at layer l captures the contextualized information after processing through the self-attention and feed-forward networks. In the context of LoRA fine-tuning, these hidden states serve as the input to the low-rank adapters, where the adaptation occurs through the transformation $h_{l+1} = W_l h_l + B_l A_l h_l$. Specifically, the covariance structure of hidden states reveals the principal directions of variation in the input data.

We design an efficient **Expert Hidden State Collector** to gather hidden states during the warm-up phase. The collector maintains separate buffers for each expert and implements a two-stage sampling strategy that aligns with the quantile-based activation selection described in the text. As shown in Algorithm 1 in Appendix B, the collector ensures balanced and memory-bounded sampling through several key techniques: CPU offloading (line 2) moves tensors to CPU immediately after collection to reduce GPU memory pressure; quantization (line 3) uses float32 precision to minimize memory

footprint; and strict buffer management (lines 7-9) prevents unbounded memory growth by discarding incoming batches when the buffer is full.

Building upon this collector, our method operates during the forward pass of MoE-LoRA layers when the warm-up mode is active. We first flatten both the input tensor and the corresponding gate values to create unified representations for processing. For each expert, we calculate a quantile-based activation threshold (line 11) to identify tokens that demonstrate significant activation patterns. We then perform a comprehensive count of tokens that exceed this threshold for each expert, establishing the relative activation frequency across all experts.

Based on these activation counts, we implement a proportional allocation strategy where experts with higher activation frequencies receive correspondingly larger sample budgets. This ensures that the collection process accurately reflects the actual routing distribution learned by the gate network during the warm-up period. The sample selection process employs a dual approach (lines 16-25): we allocate approximately 70% of the budget to the most highly activated tokens based on their gate values, while the remaining 30% is filled with randomly selected samples from the pool of above-threshold tokens. This balanced methodology maintains representativeness by prioritizing strongly activated patterns while simultaneously preserving diversity through random sampling. Crucially, the selection is constrained by the remaining buffer capacity (line 7), ensuring the total number of stored samples never exceeds the predefined `max_samples` limit.

Finally, the system collects and stores the hidden states corresponding to the selected tokens for each expert, creating specialized datasets that capture the characteristic input distributions for each expert's designated specialization area. The entire collection process operates during the forward passes of MoE-LoRA layers when the model is in warm-up mode, ensuring efficient data gathering without gradient updates.

The warm-up phase runs for a small number of iterations (20 of 500 total training steps), during which the model performs forward passes without gradient updates. Crucially, these 20 steps are strictly included within the total training budget rather than added on top of it, so that every baseline is trained for the exact same total number of optimization steps and processes the same number of tokens as EAD-LoRA; only the initialization differs. This brief period provides sufficient data to characterize each expert's expected input distribution while minimizing computational overhead (detailed in Appendix E).

3.5 Expert-Aware Differential Initialization

We now describe the initialization strategies for LoRA matrices. The standard **Kaiming initialization** (Hayou et al., 2024) method, designed for layers with ReLU activations, samples weights from a scaled uniform distribution:

$$W_{ij} \sim \mathcal{U}\left(-\sqrt{\frac{6}{d_{in}}}, \sqrt{\frac{6}{d_{in}}}\right), \tag{5}$$

where d_{in} is the input dimension. Alternatively, **Gaussian initialization** (Hu et al., 2022) employs a normal distribution with variance scaling:

$$W_{ij} \sim \mathcal{N}\left(0, \frac{1}{d_{in}}\right). \tag{6}$$

While these methods provide reasonable random starting points, they lack specialization for individual experts in MoE-LoRA systems.

Different from these standard baselines for initialization, EAD-LoRA uses Principal Component Analysis (PCA), motivated by its unique advantages for expert-aware initialization. PCA provides a mathematically principled foundation by directly extracting dominant patterns from actual expert data rather than relying on heuristic distributions, ensuring initialization is grounded in empirical evidence. Its computational efficiency and scalability make it particularly suitable for handling high-dimensional hidden states without becoming a bottleneck, unlike more complex nonlinear alternatives. The orthogonality of PCA components offers inherent benefits by ensuring independent feature learning across experts and preventing redundancy, while its adaptive nature allows each expert's initialization to naturally match the complexity of its specialized domain. In the context of MoE-LoRA, this enables precise alignment between LoRA matrix initialization and the natural

structure of data each expert encounters, creating optimal synergy between architectural design and functional specialization.

The theoretical foundation for this approach lies in the connection between PCA and the optimal low-rank approximation of a matrix. According to the Eckart-Young-Mirsky theorem (Eckart & Young, 1936), the best rank-r approximation of a matrix in terms of Frobenius norm is given by its first r principal components. Since LoRA essentially performs a low-rank adaptation of the pre-trained weights, initializing with PCA components provides a principled starting point that captures the most important directions of variation in the expert's input distribution.

The principal directions of the input distribution are also closely related to the effective gradient directions near initialization. For a linear layer with input h, the expected outer product of the loss gradient with respect to the weight matrix satisfies $\mathbb{E}[\nabla_W \mathcal{L} \, (\nabla_W \mathcal{L})^T] \propto \mathbb{E}[hh^T]$, so the top eigenvectors of the input covariance approximate the top eigenvectors of the empirical Fisher Information Matrix, which correlates with the Hessian of the loss. Initializing along these directions targets the sharpest directions of the loss landscape, the ones gradient descent follows most aggressively, and concentrates the update budget on high-variance, task-relevant directions instead of spreading it over noisy ones. We use PCA here as a principled low-rank heuristic rather than a claim of optimality; its practical value is validated empirically in Section 4.3.

Given the collected hidden states $H_i \in \mathbb{R}^{n \times d}$ for expert i, where n represents the number of tokens routed to this expert during warm-up and d is the hidden dimension, we first preprocess the data to ensure numerical stability and remove scaling artifacts. The preprocessing involves centering and normalization:

$$H_i^{\text{norm}} = \frac{H_i - \mu_i}{\sigma_i + \epsilon}, \tag{7}$$

where μ_i is the sample mean, σ_i is the sample standard deviation, and $\epsilon = 10^{-8}$ ensures numerical stability.

The PCA initialization then uses the principal components:

$$A_i^{\text{pca}} = V_i[:, : r]^T, \tag{8}$$

where V_i contains the principal components from PCA decomposition and r is the LoRA rank.

However, while PCA initialization provides data-driven specialization, it may constrain the adaptation space to patterns observable in limited warm-up data with a weak backbone model. To better balance parameter efficiency with exploration capability, we propose **EAD-LoRA+**, a hybrid approach that strategically combines the advantages of data-driven initialization with structured exploration. The method preserves meaningful PCA patterns while maintaining sufficient flexibility to discover novel feature combinations during fine-tuning. Our hybrid initialization synthesizes PCA components with randomized elements as follows:

$$A_i^{\text{hybrid}} = \alpha \cdot A_i^{\text{pca}} + (1 - \alpha) \cdot A_i^{\text{random}}, \tag{9}$$

where $\alpha \in [0, 1]$ controls the PCA strength, A_i^{pca} contains the top-r principal components from PCA decomposition, and A_i^{random} is sampled from a Kaiming uniform distribution as defined in Equation 5. This formulation enables progressive interpolation between completely random initialization ($\alpha = 0$) and pure PCA initialization ($\alpha = 1$). The impact of the PCA strength α on model performance will be systematically evaluated in Appendix K.

The final weights are normalized to ensure training stability and proper gradient flow. This normalization process involves two steps: first, the weight matrix is unitized using the Frobenius norm to control magnitude; second, it is scaled by $\sqrt{2/d_{in}}$ to maintain variance propagation consistent with Kaiming initialization principles:

$$A_i^{\text{final}} = \frac{A_i^{\text{hybrid}}}{\|A_i^{\text{hybrid}}\|_F} \cdot \sqrt{\frac{2}{d_{in}}}, \tag{10}$$

This normalization strategy prevents gradient explosion through magnitude control while preserving output variance across layers.

Finally, we follow the *zero-init* convention of LoRA: the up-projection B_i is initialized to zero so that $\Delta W_i = B_i A_i = 0$ at step 0, preserving the pre-trained model's behavior exactly at the start of

fine-tuning and preventing catastrophic forgetting or initial performance degradation. Initializing A_i with PCA components injects a semantic prior into the learnable subspace without violating this stability guarantee, since it defines the search directions of each expert while $B_i = 0$ keeps the initial output unchanged. Indeed, in our early experiments, initializing B_i (or both A_i and B_i) with PCA components broke the zero-init property and caused training divergence or significantly slower convergence.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

In most of our experiments, we utilize Llama-3.2-3B and Llama-2-7B (Touvron et al., 2023) as the foundation model and construct a mixture of LoRAs modules. This mixture comprises a total of 6 experts, with each expert having a rank of 4, and only the top-2 experts are activated at any given time. During the training phase, we adhere to a linear decay learning-rate scheduler. To ensure stable training behavior, we allocate a relatively smaller learning rate to the gate module compared to other trainable components. This reduced learning rate for the gate aids in preventing the model from undergoing sudden and erratic routing changes.

Benchmarks We consider the following benchmarks in our evaluation: ScienceQA (Lu et al., 2022) focuses on scientific knowledge Q&A, covering multiple disciplines such as physics and chemistry. CommonsenseQA (Talmor et al., 2019) focuses on commonsense reasoning and question answering, testing the model's understanding of everyday knowledge. OpenBookQA (Mihaylov et al., 2018) designs questions and answers centered on the knowledge of popular science books. SIQA (Davis & Marcus, 2015) focuses on reasoning and question answering in social scenarios. BoolQ (Clark et al., 2019) is a Boolean question answering based on web text.

Baselines We compare our method with several state-of-the-art baselines. **MoE-LoRA(Gaussian)** and **MoE-LoRA(Kaiming)** (Dou et al., 2024; Hayou et al., 2024) employ Gaussian or Kaiming initialization for MoE-LoRA. **MoE-LoRA-GA** (Wang et al., 2024) aligns initialization with full fine-tuning gradients. **PiSSA** (Meng et al., 2024) uses SVD to initialize adapters with principal singular components of pre-trained weights. **GOAT** (Fan et al., 2025) introduces an SVD-structured MoE framework with distinct singular value segments per expert and theoretical scaling.

To ensure fair comparison, all baselines are adapted to the MoE-LoRA architecture under identical experimental conditions: 6 experts, top-k=2 routing, LoRA rank $r = 4$, and the same learning rate (3×10^{-5}) and linear-decay scheduler, so that the number of trainable parameters is identical across all methods. For PiSSA and GOAT, we preserve their original design by performing Singular Value Decomposition (SVD) on the pre-trained weights and assigning distinct singular-value segments to each expert, maintaining the initialization logic within the MoE structure. For MoE-LoRA-GA, the full fine-tuning gradient is approximated on a small data batch before training. As described in Section 3.4, the warm-up steps of EAD-LoRA are counted within the total training budget, so no baseline is disadvantaged or advantaged in terms of optimization steps or tokens seen.

4.2 PERFORMANCE EVALUATION

Table 1 presents the results across all QA benchmarks. On the weaker Llama-3.2-3B backbone, EAD-LoRA achieves a 62.6% average accuracy, outperforming PiSSA (61.6%) and GOAT (61.5%); the hybrid EAD-LoRA+ further improves this to 63.7%. The gains are largest on reasoning-intensive tasks, e.g., CommonsenseQA (61.0%, +5.1 pp over MoE-LoRA/Kaiming) and ScienceQA (67.3%, +3.5 pp). On the stronger Llama-2-7B, EAD-LoRA and EAD-LoRA+ reach 65.9% and 67.8%, respectively, clearly above PiSSA (63.1%) and GOAT (64.2%). EAD-LoRA+ is again the best on CommonsenseQA (61.7% vs. 51.9% for the Gaussian baseline, +9.8 pp), ScienceQA (68.5%) and BoolQ (81.9%); the sole exception is SIQA, where pure EAD-LoRA slightly edges out EAD-LoRA+ (69.1% vs. 68.3%). Overall, relative to the strongest baseline, EAD-LoRA+ gains +2.1 pp on Llama-3.2-3B (over PiSSA) and +3.6 pp on Llama-2-7B (over GOAT), indicating that the balanced exploration-exploitation trade-off remains well calibrated.

Table 1: Experimental Results of Accuracy(%) on Question Answering Benchmarks. K: Kaiming, G: Gaussian, SCQA: ScienceQA, CSQA: CommonsenseQA, OBQA: OpenBookQA. EAD-LoRA+ achieves the best performance on almost all benchmarks.

Backbone	Method	SCQA	CSQA	OBQA	SIQA	BoolQ	AVG
Llama-3.2-3B	MoE-LoRA(K)	63.8	55.9	60.0	58.7	66.5	61.0
	MoE-LoRA(G)	62.1	55.4	60.8	58.8	67.9	61.0
	MoE-LoRA-GA	61.9	54.8	60.0	58.7	65.6	60.2
	PiSSA	64.5	56.8	60.5	59.2	67.0	61.6
	GOAT	64.9	56.5	60.2	59.2	66.7	61.5
	EAD-LoRA	66.7	58.3	60.8	61.1	65.9	62.6
	EAD-LoRA+	**67.3**	**61.0**	**61.2**	**61.1**	**68.1**	**63.7**
Llama-2-7B	MoE-LoRA(K)	54.3	44.8	54.4	68.3	78.8	60.1
	MoE-LoRA(G)	59.4	51.9	54.4	66.7	80.2	62.5
	MoE-LoRA-GA	58.3	58.1	50.4	68.8	81.9	63.5
	PiSSA	60.2	53.5	54.0	67.5	80.5	63.1
	GOAT	63.3	56.0	55.1	67.6	79.0	64.2
	EAD-LoRA	67.3	55.2	57.6	**69.1**	80.3	65.9
	EAD-LoRA+	**68.5**	**61.7**	**58.4**	68.3	**81.9**	**67.8**

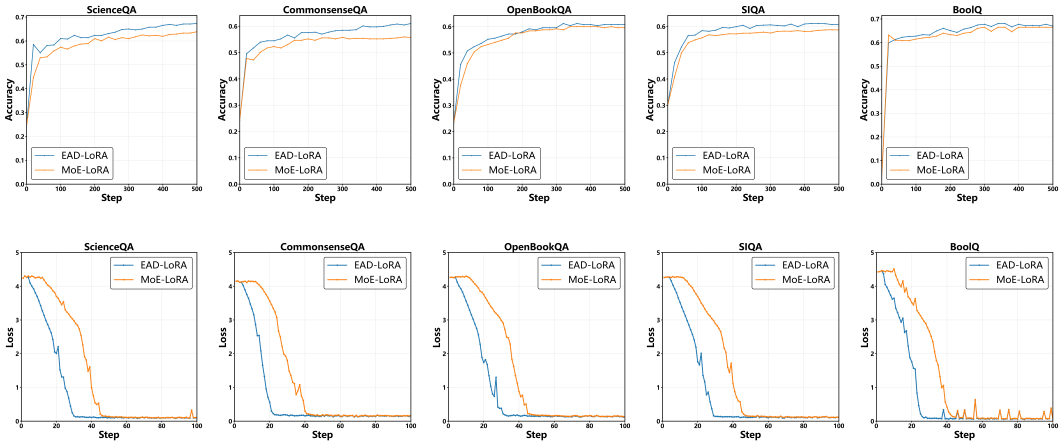


Figure 2: Accuracy (top) and Loss (bottom) on Llama-3.2-3B backbone. Blue: EAD-LoRA+. Orange: MoE-LoRA. The blue curve (EAD-LoRA+) exhibits a more rapid increase in accuracy and a faster decline in loss.

Unlike PiSSA and GOAT, which rely on a static SVD decomposition of the pre-trained weights, our EAD-LoRA collects empirical routing statistics during a warm-up phase and applies PCA to tailor each expert to its actual input distribution, which is why it consistently outperforms all baselines on both backbones.

Performance Visualization Besides final accuracy, EAD-LoRA+ demonstrates superior training dynamics. As shown by the Step-Accuracy and Step-Loss curves in Figure 2 on the Llama-3.2-3B backbone, EAD-LoRA converges faster: the Step-Accuracy curves (500 steps) show it reaches comparable accuracy in far fewer steps, and the Step-Loss curves (100 steps) show lower, more stable loss early in training, especially on complex reasoning tasks such as ScienceQA and CommonsenseQA.

Routing Entropy Analysis EAD-LoRA also keeps lower routing entropy than randomly or PiSSA-initialized MoE-LoRA during early training, confirming that the router dispatches tokens to appropriate experts with higher confidence from the outset and exploits the initialized expert structure immediately.

4.3 Ablation Study

To systematically isolate the contribution of each component of EAD-LoRA, we conduct ablation studies on the Llama-3.2-3B backbone. We ablate (i) the PCA-based expert initialization itself and (ii) the hybrid exploration component of EAD-LoRA+, while the choice of the data-driven initialization strategy is ablated in Appendix L.

Effect of the PCA Initialization Component We first quantify the contribution of the warm-up and PCA-based initialization. As shown in Figure 3, replacing the PCA-initialized experts with purely random (Kaiming) initialization—i.e., the standard MoE-LoRA baseline without any expert-aware guidance—drops the average accuracy from 62.6% to 61.0%. This confirms that the data-driven PCA initialization is the primary source of improvement (+1.6 pp), consistent with the main comparison in Table 1.

Effect of the Hybrid Exploration Component To isolate the contribution of the hybrid exploration strategy in EAD-LoRA+, we compare the hybrid variant ($\alpha = 0.7$) against pure PCA initialization ($\alpha = 1.0$). Figure 3 shows that removing the random exploration component drops the average accuracy from 63.7% to 62.6% (-1.1 pp). Pure PCA already provides the majority of the gain, while the hybrid strategy further improves performance by balancing data-driven specialization with exploratory capacity.

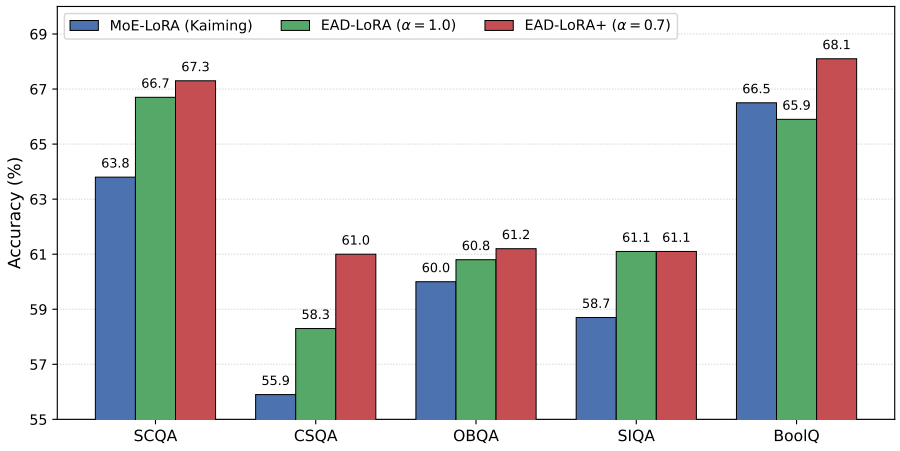


Figure 3: Component ablation of EAD-LoRA on Llama-3.2-3B. Removing the PCA initialization (random Kaiming) drops the average accuracy from 62.6% to 61.0%, and removing the hybrid exploration component (pure PCA, $\alpha = 1.0$) drops it from 63.7% to 62.6%. Best results in **bold**.

The sensitivity of the two key hyperparameters—the PCA strength α and the warm-up duration $T_{\text{warm-up}}$—is analyzed in Appendix K. Collectively, these ablations demonstrate that both the PCA initialization and the hybrid exploration components contribute to the final performance.

5 Conclusion

In this paper, we proposed EAD-LoRA, a novel initialization scheme for MoE-LoRA frameworks that addresses the critical limitation of uniform expert initialization. Our method provides expert-specific initialization through a two-phase process of hidden state collection and PCA-based initialization. The hybrid variant EAD-LoRA+ further enhances robustness by balancing PCA-guided specialization with random exploration. Extensive experiments across multiple benchmarks demonstrate that our approach achieves superior performance compared to existing initialization methods.

AI USE STATEMENT

In this work, generative AI tools were used only for recommended-disclosure tasks: drafting/editing text, suggesting organization, formatting references, and modifying figures. No required-disclosure tasks were performed with AI (e.g., no synthetic data generation, theoretical development, hypothesis formulation, experiment design, method implementation, or result interpretation). All figures derive from authors' experimental data and were manually verified. All AI-assisted content was reviewed for factual accuracy, plagiarism, and scientific integrity through manual literature survey and author verification. We take full responsibility for the final content of this work.

ETHICS STATEMENT

This work focuses on the efficient adaptation of large language models through improved initialization of MoE-LoRA frameworks. The research does not involve human subjects, sensitive personal data, or experiments with potential for direct harm. All models used in our experiments are publicly available pre-trained models, and all datasets employed are publicly available benchmark datasets used under their respective licenses for academic research purposes. We do not anticipate any negative societal impacts from this work; on the contrary, by reducing the computational resources required for model adaptation, our method may help lower the environmental footprint and resource barriers associated with fine-tuning large foundation models.

REPRODUCIBILITY STATEMENT

We describe the full architecture, the two-phase initialization procedure, and the complete hyperparameter settings in the paper and its appendices. All experiments are run with fixed random seeds, and every main experiment is repeated five times with different seeds to report stable statistics. The code used to produce all reported results is publicly available at `https://anonymous.4open.science/r/EAD-LoRA-plus-AC73`, and the data preparation, training, and evaluation procedures are documented in detail in the appendices to ensure that the experimental results can be reproduced.

REFERENCES

Krzysztof Choromanski, Carlton Downey, and Byron Boots. Initialization matters: Orthogonal predictive state recurrent neural networks. In *International Conference on Learning Representations*, 2018.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

Ernest Davis and Gary Marcus. Commonsense reasoning and commonsense knowledge in artificial intelligence. *Communications of the ACM*, 58(9):92–103, 2015.

Shizhe Diao, Tianyang Xu, Ruijia Xu, Jiawei Wang, and Tong Zhang. Mixture-of-domain-adapters: Decoupling and injecting domain knowledge to pre-trained language models memories. *CoRR*, 2023.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Wei Shen, Limao Xiong, Yuhao Zhou, Xiao Wang, Zhiheng Xi, Xiaoran Fan, et al. Loramoe: Alleviating world knowledge forgetting in large language models via moe-style plugin. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1932–1945, 2024.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3): 211–218, 1936.

Chenghao Fan, Zhenyi Lu, Sichen Liu, Chengfeng Gu, Xiaoye Qu, Wei Wei, and Yu Cheng. Make lora great again: Boosting lora with adaptive singular values and mixture-of-experts optimization alignment. In *International Conference on Machine Learning*, pp. 15804–15832. PMLR, 2025.

Chongyang Gao, Kezhen Chen, Jinmeng Rao, Baochen Sun, Ruibo Liu, Daiyi Peng, Yawen Zhang, Xiaoyuan Guo, Jie Yang, and VS Subrahmanian. Higher layers need more lora experts. *arXiv preprint arXiv:2402.08562*, 2024.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and S Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. arxiv 2024. *arXiv preprint arXiv:2403.14608*, 10, 2024.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. The impact of initialization on lora finetuning dynamics. *Advances in Neural Information Processing Systems*, 37:117015–117040, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Shwai He, Liang Ding, Daize Dong, Miao Zhang, and Dacheng Tao. Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. *arXiv preprint arXiv:2210.04284*, 2022.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Dengchun Li, Yingzi Ma, Naizheng Wang, Zhiyuan Cheng, Lei Duan, Jie Zuo, Cal Yang, and Mingjie Tang. Mixlora: Enhancing large language models fine-tuning with lora based mixture of experts. *CoRR*, 2024.

Qidong Liu, Xian Wu, Xiangyu Zhao, Yuanshao Zhu, Derong Xu, Feng Tian, and Yefeng Zheng. Moelora: An moe-based parameter efficient fine-tuning method for multi-task medical applications. *CoRR*, 2023.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, 2022.

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. *CoRR*, 2024.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072, 2024.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158. Association for Computational Linguistics, 2019.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931, 2024.

Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermis, Acyr Locatelli, and Sara Hooker. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning. *CoRR*, 2023.

Dan Zhang, Tao Feng, Lilong Xue, Yuandong Wang, Yuxiao Dong, and Jie Tang. Parameter-efficient fine-tuning for foundation models. *arXiv preprint arXiv:2501.13787*, 2025a.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023a.

Yuanhe Zhang, Fanghui Liu, and Yudong Chen. Lora-one: One-step full gradient could suffice for fine-tuning large language models, provably and efficiently. In *Forty-second International Conference on Machine Learning*, 2025b.

Zhen-Ru Zhang, Chuanqi Tan, Haiyang Xu, Chengyu Wang, Songfang Huang, et al. Towards adaptive prefix tuning for parameter-efficient language model fine-tuning. In *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023b.

A Notation

For clarity, Table 2 summarizes the principal notation used throughout the paper.

Table 2: Principal notation used in this paper.

Symbol	Description
W_0, W	Frozen pre-trained weights and fine-tuned weights
ΔW	Low-rank update such that $W = W_0 + \Delta W$
$B \in \mathbb{R}^{d_{\text{out}} \times r}$, $A \in \mathbb{R}^{r \times d_{\text{in}}}$	LoRA down- and up-projection matrices, $r \ll \min(d_{\text{out}}, d_{\text{in}})$
h, x	Hidden state (input to a LoRA layer) and token/input embedding
$G(\cdot)$, $g_i(x)$	Gating network and gate value (routing probability) of expert i
E_i, N	i-th expert network and number of experts
$T_{\text{warm-up}}$	Warm-up duration (number of forward-only steps)
$\mathbf{H}_i$	Collected hidden states routed to expert i
Σ_i, $\lambda_j^{(i)}$	Covariance matrix and j-th eigenvalue for expert i
$v_j^{(i)}$	j-th principal component (eigenvector) for expert i
r_{pca}	Number of principal components used for initialization
α	Hybrid strength controlling the PCA/random mixing ratio
P_A, P_B	Assignment matrices mapping principal components to A/B
k	Top-k routing sparsity (number of activated experts)
$\mathcal{L}$	Training loss

B Expert Hidden State Collector

The complete pseudo-code of the Expert Hidden State Collector used during the warm-up phase is provided in Algorithm 1.

C Experimental Setup

C.1 Dataset Setup

Data Preparation Process A standardized data preparation pipeline was employed across all datasets to ensure consistency in instruction formatting and evaluation. The common process involves: (i) loading and splitting the raw data into training, validation, and test sets; (ii) structuring each sample into a unified instruction-following format composed of an instruction (the task description and input context) and an output (the expected model response); and (iii) implementing a standardized evaluation function that extracts the model's answer based on a dataset-specific regular expression pattern for accurate accuracy calculation. This uniform approach allows for fair comparison and seamless training across diverse task types.

We use the ScienceQA dataset as a concrete example to illustrate this pipeline, as detailed in Algorithm 2. The raw ScienceQA data, which contains multimodal (text and image) questions, is

Algorithm 1 Expert Hidden State Collector

Input: Expert i, hidden states $H \in \mathbb{R}^{n \times d}$, gate values $G \in \mathbb{R}^{n}$, quantile threshold τ
$H_{\text{cpu}} \leftarrow H$.cpu()
$H_{\text{np}} \leftarrow H_{\text{cpu}}$.numpy().astype(np.float32)
$G_{\text{np}} \leftarrow G$.cpu().numpy().astype(np.float32)
$n_{\text{new}} \leftarrow \text{len}(H_{\text{np}})$
$n_{\text{current}} \leftarrow$ sample_counts$[i]$
$n_{\text{budget}} \leftarrow \max(0,\ \text{max_samples} - n_{\text{current}})$
if $n_{\text{budget}} = 0$ **then**
  skip {Buffer full, discard incoming batch}
else
  threshold_val $\leftarrow$ quantile$(G_{\text{np}},\ \tau)$
  active_mask $\leftarrow G_{\text{np}} \geq$ threshold_val
  $H_{\text{active}} \leftarrow H_{\text{np}}$[active_mask]
  $G_{\text{active}} \leftarrow G_{\text{np}}$[active_mask]
  $n_{\text{active}} \leftarrow \text{len}(H_{\text{active}})$
  if $n_{\text{active}} \leq n_{\text{budget}}$ **then**
    $H_{\text{selected}} \leftarrow H_{\text{active}}$
  else
    $n_{\text{top}} \leftarrow \lfloor 0.7 \times n_{\text{budget}} \rfloor$
    $n_{\text{rand}} \leftarrow n_{\text{budget}} - n_{\text{top}}$
    top_indices $\leftarrow$ argsort$(G_{\text{active}})[-n_{\text{top}}:]$
    rand_indices $\leftarrow$ random_choice$(n_{\text{active}},\ n_{\text{rand}})$
    selected_indices $\leftarrow$ unique(concat(top_indices, rand_indices))
    $H_{\text{selected}} \leftarrow H_{\text{active}}$[selected_indices]
  end if
  hidden_states$[i] \leftarrow$ concatenate(hidden_states$[i]$, H_{selected})
  sample_counts$[i] \leftarrow$ sample_counts$[i] + \text{len}(H_{\text{selected}})$
end if
Output: hidden_states$[i]$, sample_counts$[i]$

first processed by filtering for text-only samples. Key components like the question context, multiple-choice options, and the correct answer are extracted. These components are then formatted into a final instruction-output pair using a predefined template (e.g., the CQM-A format: Context, Question, Options, Answer). The resulting structured data is saved in a standardized format ready for model training and evaluation.

The algorithm begins by loading and restructuring the dataset (lines 3-5). For each text-only example (line 8), it constructs the input by concatenating the context, question, and formatted options (line 9-11). The output is formatted as a fixed string "Answer: The answer is X." (where "X" is the correct option). This structured format, consistent with the template described in the Instruction Template Details section, enables the model to learn the desired question-answering pattern and allows the evaluation function to reliably extract the predicted answer using the regex pattern "The answer is ([A-Z])".

Dataset Statistics The datasets vary in size and domain, covering commonsense reasoning, scientific question answering, and textual inference tasks. Table 3 summarizes the number of samples in the training, validation, and test splits for each dataset. The dataset sizes range from smaller datasets like OpenBookQA (with 5,957 total samples) to larger ones like SIQA (with 35,364 total samples), providing a diverse testbed for evaluating our method across different data scales and task complexities. All datasets were preprocessed using the standardized instruction-tuning format described in Section C.2 to ensure consistent evaluation and fair comparison.

The distribution of samples across splits follows standard practices in the literature, with approximately 80-90% of data allocated for training and the remainder divided between validation and testing. The validation sets are used for hyperparameter tuning and early stopping, while the test sets are held out for final evaluation. This partitioning ensures robust model development and reliable performance estimation on unseen data across all benchmark tasks.

Algorithm 2 ScienceQA Data Preprocessing

Input: Raw ScienceQA dataset, prompt format (e.g., CQM-A)
Output: Processed dataset in instruction-following format
Load dataset from HuggingFace Hub
Merge and re-split data into
Train/Validation/Test
for each $split$ in [train, validation, test] **do**
  for each $example$ in $split$ **do**
    if $example.image$ is $None$ **then**
      $context \leftarrow$ get_context_text($example$) *// Get hint & caption*
      $choice \leftarrow$ get_choice_text($example, options$) *// Format as (A) ... (B) ...*
      $answer \leftarrow$ get_answer($example, options$) *// Map to A, B, etc.*
      $instruction, output \leftarrow$
        create_one_example($format$,
        $example.question, context$,
        $choice, answer$)
      Store processed example with
      $instruction$ and $output$ fields
    end if
  end for
  Save split as JSON and HuggingFace Dataset
end for
return Processed DatasetDict

Table 3: Dataset size statistics across different benchmarks (in thousands)

Dataset	Train	Validation	Test
ScienceQA	8.8k	1.1k	1.0k
CommonsenseQA	8.8k	1.1k	1.1k
OpenBookQA	5.0k	0.5k	0.5k
SIQA	28.3k	3.5k	3.5k
BoolQ	10.2k	1.3k	1.3k

C.2 Instruction Templates

We provide a comprehensive overview of the instruction templates used for each dataset in our experiments in Table 4. All datasets follow a consistent multiple-choice QA format built upon the `MultipleChoiceQADataset` base class, which implements standardized prompt formatting, response extraction, and evaluation logic. The templates employ a structured format with clear section demarcations (Instruction, Input, Response) and include specific response templates that guide the model to produce answers in a consistent pattern (e.g.,"The answer is .."). During evaluation, responses are extracted using regular expression patterns that match the expected answer format, ensuring accurate performance measurement across different datasets.

Table 4: Instruction template configurations for different datasets.

Dataset	Input Length	Output Length	Answer Pattern
ScienceQA	256	10	`The answer is ... ([A-Z])`
CommonsenseQA	256	10	`The answer is ... ([A-Z])`
OpenBookQA	256	10	`The answer is ... ([A-Z])`
SIQA	256	10	`The answer is ... ([A-Z])`
BoolQ	512	10	`The answer is ... (True|False)`

All templates share common structural elements including prompt input/no-input variants, response split markers (`### Response:`), and failure handling mechanisms. The evaluation process automatically extracts answers using the specified regex patterns and performs case-insensitive matching after

cleaning non-alphanumeric characters, ensuring robust performance measurement across different response variations.

C.3 Experimental Metrics

We employ accuracy and cross-entropy loss as our primary evaluation metrics. Accuracy measures the proportion of correctly predicted answers, while loss quantifies the divergence between predicted and actual probability distributions. The accuracy computation process involves extracting model responses using regex patterns and comparing them with ground truth labels, as detailed in Algorithm 3. For multiple-choice datasets, answers are extracted using dataset-specific regular expressions (e.g., `([A-Z])` for letter choices or `(True|False)` for boolean questions), with case-insensitive matching after removing non-alphanumeric characters.

Algorithm 3 Accuracy Computation

1: **Input:** inputs, labels, preds, tokenizer
2: **Output:** Accuracy
3: $pairs \leftarrow [\,]$
4: **for** $i \leftarrow 1$ to len$(preds)$ **do**
5: $\quad label_str \leftarrow$ tokenizer.decode$(labels[i])$
6: $\quad pred_str \leftarrow$ tokenizer.decode$(preds[i])$
7: $\quad pred_str \leftarrow$ get_response$(pred_str)$ {Extract response part}
8: $\quad pairs$.append$((pred_str, label_str))$
9: **end for**
10: $count \leftarrow 0$
11: **for** $i, (pred, label)$ in $pairs$ **do**
12: $\quad pattern \leftarrow$ dataset.response_template
13: $\quad pred_clean \leftarrow$ clean_text$(pred)$ {Remove non-alphanumeric}
14: $\quad answer \leftarrow$ regex_match$(pattern, pred_clean)$
15: $\quad label_clean \leftarrow$ clean_text$(label)$
16: $\quad ground_truth \leftarrow$ regex_match$(pattern, label_clean)$
17: $\quad$ **if** $answer = ground_truth$ **then**
18: $\quad\quad count \leftarrow count + 1$
19: $\quad$ **end if**
20: **end for**
21: **return** $count/$len$(pairs)$

The algorithm first decodes tokenized predictions and labels into text strings (lines 3-7). It then extracts the relevant response portion using dataset-specific split markers. For each prediction-label pair, both texts are cleaned and processed using the dataset's response template regex pattern to extract the core answer (lines 10-13). Answers are compared case-insensitively after cleaning, with matches contributing to the final accuracy score. This approach ensures robust evaluation across varying response formats while maintaining consistency with each dataset's answer schema.

D Implementation Details

D.1 Hardware and Software Environment

All experiments were conducted on a server equipped with two NVIDIA Tesla V100 GPUs, each with 32 GB of memory, and Intel Xeon Gold 6240 CPUs running at 2.60 GHz. The system has 256 GB of RAM and runs Ubuntu 20.04 LTS. CUDA version 12.0 and PyTorch 2.0 were used for model training and evaluation.

D.2 Training Hyperparameters

We provide a comprehensive list of hyperparameters used in our experiments with the SGD optimizer in Table 5 on dataset ScienceQA. These configurations were carefully selected to optimize the performance of our MoE-LoRA model across different datasets. The hyperparameters include the

number of experts, LoRA rank (r), LoRA alpha (α), top-k experts selected, learning rates for experts and the gating network, batch sizes, gradient accumulation steps, and other training-specific settings. The sequence lengths (e.g., 64+10) represent the sum of input tokens and output tokens, where the first number indicates the maximum input sequence length and the second number indicates the maximum generated output tokens. All values are adjusted according to these sequence lengths to ensure efficient memory usage and stable training.

Table 5: Hyperparameter configuration for Experiments on ScienceQA / Llama-3.2-3B.

Hyperparameter	**Value**
Number of Experts	6
LoRA Rank (r)	4
LoRA Alpha (α)	16
Top-k Experts	2
Expert Learning Rate	3×10^{-5}
Gate Learning Rate	3×10^{-8}
Warm-up Steps	20
Total Gradient Steps	500
Evaluation Steps	20
Max Gate Grad Norm	1.0
Per-GPU Batch Size	
- Sequence Length 64+10	120
- Sequence Length 128+10	60
- Sequence Length 256+10	30
- Sequence Length 512+10	15
Accumulation Steps	
- Sequence Length 64+10	1
- Sequence Length 128+10	1
- Sequence Length 256+10	2
- Sequence Length 512+10	4

E Overhead Analysis

In this section, we provide a rigorous quantitative analysis of the computational burden and runtime overhead introduced by the two-phase process (warm-up and PCA computation) of EAD-LoRA. We quantify these additional steps relative to standard MoE-LoRA training and demonstrate their cost-effectiveness given the performance improvements.

To accurately measure overhead, we conduct experiments under the following configuration: model configuration (Llama-3.2-3B, 6 experts, LoRA rank $r = 4$, top-$k = 2$), dataset (ScienceQA training set with 8,800 samples), and training configuration (total training steps 500, batch size 30, sequence length 256+10).

E.1 Warm-up Phase Overhead Analysis

The warm-up phase involves forward propagation with frozen LoRA parameters to collect hidden states. Table 6 provides a detailed comparison between the warm-up phase and standard training iterations. The warm-up phase requires no backward propagation or parameter updates, resulting in 12.2% less GPU time per step compared to standard training. However, offloading hidden states from GPU to CPU for storage increases CPU memory usage by 104.8%, while GPU-CPU data transfer increases significantly (500%), representing the primary overhead source for hidden state collection.

E.2 PCA Computation Phase Overhead Analysis

The PCA computation phase performs principal component analysis on the collected hidden states for each expert. Table 7 shows PCA computation overhead under different expert configurations.

Table 6: Warm-up Phase vs. Standard Training Iteration Overhead

Metric	Warm-up Phase	Standard Training	Relative Overhead
GPU Time/Step (ms)	125.3 ± 2.1	142.7 ± 3.5	-12.2%
GPU Memory Peak (GB)	18.2	22.5	-19.1%
CPU Memory Usage (GB)	4.3	2.1	+104.8%
Data Transfer (GB/step)	1.8	0.3	+500.0%

Table 7: PCA Computation Phase Overhead Analysis

# Experts	Total Samples	PCA Time (s)	Memory Peak (GB)	% of Training Time
4	15,360	2.34 ± 0.12	3.2	0.47%
6	23,040	4.87 ± 0.21	4.8	0.97%
8	30,720	8.92 ± 0.35	6.5	1.78%
Average per Expert	3,840	0.81 ± 0.04	0.8	0.16%

PCA time scales approximately linearly with the number of experts and samples, averaging 0.81 seconds per expert. The computation requires loading hidden states into memory for covariance matrix calculation, with memory usage proportional to sample dimensions. Despite this, even with 8 experts, PCA accounts for only 1.78% of total training time, representing negligible overhead.

E.3 End-to-End Training Overhead Analysis

To illustrate the practical cost of our method, we compare the total wall-clock training time of EAD-LoRA against standard (MoE-LoRA) training across the main benchmark tasks in Table 8. EAD-LoRA consistently completes training in approximately 104% to 107% of the baseline time, i.e., it incurs only a 4% to 7% additional overhead. This marginal increase is primarily attributed to the warm-up phase, during which we collect data representations for PCA computation (forward passes with frozen LoRA parameters for approximately 20 steps, followed by a one-time PCA decomposition on the collected representations). Once the initialization is completed, the subsequent training proceeds at the same speed as the baseline, since our method adds no per-step computational cost during optimization. Given the performance improvements observed across both QA and text classification settings, we believe this 4% to 7% overhead is well justified and practically negligible, especially in scenarios where model accuracy is critical.

Table 8: End-to-end total training clock time of EAD-LoRA relative to standard MoE-LoRA (100%). Across all main benchmark tasks, EAD-LoRA completes training in approximately 104%–107% of the baseline time.

Method	Total Training Time (% of MoE-LoRA)
Standard MoE-LoRA	100
EAD-LoRA (range across benchmarks)	104–107

F Sensitive Experiment

To investigate the robustness of our EAD-LoRA+ method under different mixture-of-experts (MoE) configurations, we conduct sensitivity experiments varying the number of experts and the top-k routing parameter on Llama-3.2-3B. The baseline configuration uses 6 experts with top-k=2, which represents a balanced setting for computational efficiency and model capacity. As shown in Table 9, our analysis reveals a clear and consistent pattern: **EAD-LoRA+ maintains superior performance over MoE-LoRA across all examined configurations**, demonstrating the fundamental advantage of our hybrid initialization approach regardless of architectural parameters.

Notably, EAD-LoRA+ achieves higher accuracy than MoE-LoRA in every tested configuration, with performance advantages ranging from 2.2 percentage points (6 experts, top-k=3) to 5.5 percentage points (6 experts, top-k=1). This consistent superiority confirms that our initialization method provides intrinsic benefits independent of the specific MoE hyperparameter choices. Moreover, while both methods exhibit sensitivity to routing parameters, EAD-LoRA+ consistently achieves better absolute performance across the entire parameter space.

The robustness of EAD-LoRA+ is particularly evident when examining performance stability. Across different expert counts with fixed top-k=2, EAD-LoRA+ shows remarkably stable accuracy (67.1%-67.3%), whereas MoE-LoRA exhibits greater variability (62.9%-64.6%). Similarly, when varying top-k with fixed expert count, EAD-LoRA+ maintains higher performance levels across all settings. This demonstrates that our hybrid initialization not only improves peak performance but also reduces sensitivity to architectural choices, making it more reliable for practical applications where optimal hyperparameters may not be known in advance.

The consistent performance advantage stems from EAD-LoRA+'s ability to better initialize expert networks with task-relevant directions while maintaining exploration capacity. Unlike MoE-LoRA's conventional initialization, which relies on random or simple heuristic methods, our approach leverages PCA-derived patterns from warm-up data combined with controlled randomization. This enables each expert to start from a more informative initialization point, accelerating convergence and improving final performance regardless of how many experts are used or how tokens are routed among them. The results conclusively demonstrate that EAD-LoRA+'s benefits are fundamental to its initialization strategy rather than dependent on specific MoE configurations.

Table 9: Sensitive experiment results of MoE parameters with EAD-LoRA+ and standard MoE-LoRA. All reported values are accuracy (%).

Experts	**Top_K**	**MoE-LoRA**	**EAD-LoRA+**
6	2	63.8	67.3
4	2	64.6	67.1
8	2	62.9	67.3
6	1	65.6	71.1
6	3	61.6	63.8

G Generalization to Text Classification (SST-2)

To provide preliminary evidence of generalization beyond QA, we additionally evaluate our method on a text classification task (SST-2, sentiment analysis) using the same Llama-3.2-3B backbone and MoE-LoRA configuration (6 experts, rank 4, top-2, 500 total training steps with a 20-step warm-up for EAD-LoRA). EAD-LoRA initialization consistently leads to faster convergence and better final performance stability compared to standard MoE-LoRA. This early positive signal, while preliminary, suggests that the benefit of routed PCA initialization extends beyond reasoning-heavy QA tasks to broader NLU scenarios, reinforcing our claim that the method captures general-purpose feature structures rather than task-specific artifacts.

H Compatibility with Other Methods

It is important to emphasize that our proposed EAD-LoRA framework specifically targets the initialization phase of LoRA parameters in MoE architectures, making it inherently orthogonal and complementary to other optimization techniques. Since our method operates exclusively during the parameter initialization stage and does not modify the core training dynamics or architectural components, it can be seamlessly integrated with various existing optimization approaches.

For gradient-based optimization methods, EAD-LoRA provides a superior starting point that may enhance their effectiveness by beginning from a more favorable region in the optimization landscape. Similarly, for expert coordination mechanisms, our initialization scheme offers better-initialized experts that are already aligned with their respective input distributions, potentially accelerating the learning of effective routing patterns.

The modular nature of our approach ensures that it can be combined with other LoRA variants and MoE enhancements without architectural conflicts. This compatibility positions EAD-LoRA as a fundamental building block that can be incorporated into broader optimization pipelines, potentially yielding cumulative benefits when paired with other advanced techniques in the rapidly evolving field of parameter-efficient fine-tuning.

I Expert Specialization Analysis

To provide direct evidence that our initialization induces meaningful expert differentiation rather than merely random diversity, we quantify the geometric relationship between the expert parameters on Llama-3.2-3B/ScienceQA using two complementary metrics: pairwise cosine similarity (directional alignment) and Frobenius distance (parameter-space separation). Table 10 reports both metrics.

Randomly initialized (Kaiming) experts are essentially orthogonal (average pairwise cosine similarity ≈ 0) and far apart (Frobenius distance 1.5677): they are effectively "strangers" in parameter space with no shared context, an indiscriminate form of diversity. In contrast, PCA-initialized experts exhibit a positive cosine similarity of 0.2276 and a significantly compact Frobenius distance of 0.0310. Because all experts are anchored in the principal components of the shared data distribution, they start from a cohesive, high-value region of parameter space while still diverging along distinct, meaningful directions within that subspace. Our initialization therefore creates a team of *purposefully specialized* experts rather than the indiscriminate diversity of random initialization, providing a direct empirical basis for the convergence and routing improvements reported above.

Table 10: Geometric properties of initialized experts on Llama-3.2-3B/ScienceQA. Lower Frobenius distance and positive cosine similarity indicate that PCA-initialized experts are anchored in a shared salient subspace while diverging along meaningful directions.

Initialization	**Pairwise Cosine Sim.**	**Frobenius Distance**
Kaiming (random)	≈ 0	1.5677
EAD-LoRA (PCA)	0.2276	0.0310

J Statistical Significance

To confirm that the reported gains are not an artifact of a single favorable seed, we run every main experiment five times with different random seeds. On Llama-3.2-3B, EAD-LoRA+ attains an average accuracy between 63.2% and 65.1% across the five runs, consistently outperforming MoE-LoRA(Kaiming), which ranges from 60.5% to 61.4%. Similarly, the alternative-initialization comparisons in Appendix L are stable across runs. The multi-run analysis therefore confirms that the performance gains reported in Table 1 are statistically robust rather than seed-dependent.

K Hyperparameter Sensitivity Analysis

To complement the component ablations in Section 4.3, we analyze the sensitivity of two key hyperparameters of EAD-LoRA: the PCA strength α and the warm-up duration $T_{\text{warm-up}}$, using the Llama-3.2-3B backbone. Table 11 summarizes both studies.

Effect of α Varying α from 0.5 (balanced) to 1.0 (pure PCA), we find that $\alpha = 0.7$ yields the best average accuracy (63.7%), outperforming $\alpha = 0.5$ (62.2%) and $\alpha = 1.0$ (62.6%). The gain is largest on complex reasoning (CommonsenseQA: +3.8 pp over $\alpha = 0.5$), indicating that PCA-dominated yet non-deterministic initialization optimally balances data-driven specialization and adaptability.

Effect of $T_{\text{warm-up}}$ Increasing warm-up from 5 to 20 steps improves average accuracy from 62.1% to 63.7%, as more stable routing patterns are captured. Extending to 50 steps slightly reduces performance (63.5%) due to less fine-tuning time under a fixed total budget. Thus, 20 steps offer the best trade-off.

Table 11: Sensitivity of the PCA strength α (top, with 20-step warm-up) and warm-up duration $T_{\text{warm-up}}$ (bottom, with $\alpha = 0.7$). Best per block in **bold**.

Variant	SCQA	CSQA	OBQA	SIQA	BoolQ	AVG
			α **variation**			
$\alpha = 0.5$	66.5	57.2	60.4	60.5	66.3	62.2
$\alpha = 0.9$	67.1	57.1	60.4	61.1	69.0	62.9
$\alpha = 1.0$ (EAD-LoRA)	66.7	58.3	60.8	61.1	65.9	62.6
$\alpha = 0.7$ (EAD-LoRA+)	**67.3**	**61.0**	**61.2**	**61.1**	**68.1**	**63.7**
			$T_{\text{warm-up}}$ **variation**			
5 steps	65.8	58.1	59.8	60.2	66.5	62.1
10 steps	66.5	59.4	60.8	60.8	67.2	62.9
20 steps	**67.3**	**61.0**	**61.2**	61.1	**68.1**	**63.7**
50 steps	67.0	60.2	60.8	**61.4**	67.8	63.5

L Alternative Initialization Strategies

To isolate the contribution of PCA and further validate the design choice, we compare against alternative structured and data-driven initialization strategies on Llama-3.2-3B/ScienceQA in Table 12: random orthogonal initialization (62.3%), K-means centroid initialization (61.8%), and SVD applied to the collected activations (62.5%). Our PCA-only EAD-LoRA ($\alpha = 1.0$) reaches 66.7%, and the hybrid EAD-LoRA+ ($\alpha = 0.7$) further improves to 67.3%. The large gap over K-means (67.3% vs. 61.8%) confirms that the benefit arises from capturing the principal directions of variation within each cluster rather than from mere centroid-based partitioning, while PCA's advantage over SVD applied to the same activations (66.7% vs. 62.5%) reflects its variance-maximizing and orthogonality properties.

Table 12: Comparison with alternative initialization strategies on Llama-3.2-3B/ScienceQA (%).

Initialization	Accuracy
Random Orthogonal	62.3
K-means Centroid	61.8
SVD (collected activations)	62.5
EAD-LoRA ($\alpha = 1.0$, PCA)	66.7
EAD-LoRA+ ($\alpha = 0.7$, hybrid)	**67.3**