InLoRA: INSTANCE-CUSTOMIZED LORA GENERATION VIA DISCRETE PRIORS FOR EFFICIENT LLM ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

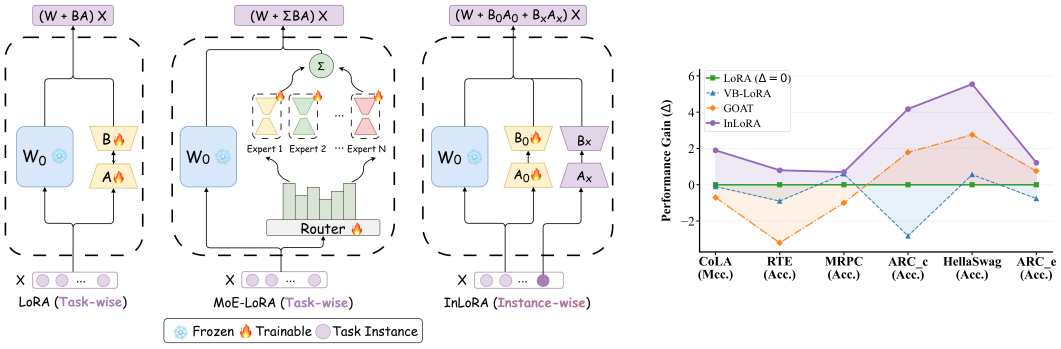
As a mainstream Parameter-Efficient Fine-Tuning (PEFT) method, Low-Rank Adaptation (LoRA) significantly reduces the optimization and deployment costs of large language models by introducing trainable low-rank matrices. Despite the progress, existing LoRA and its variants typically learn a *universal* adapter at the entire task or dataset level, failing to achieve fine-grained personalized adaptation for individual instances, and thus limiting the performance in complex scenarios. In this paper, we propose **InLoRA**, a novel dynamic, instance-customized low-rank adapter considering individual idiosyncrasies. In particular, our **InLoRA** first maintains a discrete codebook with vector quantization to store the prior patterns of adapters. More importantly, we retrieve from the codebook for individual instances and then utilize lightweight projections to generate instance-customized yet well-structured adapters, which would be combined with task-specific adapters for efficient fine-tuning. Extensive experiments demonstrate that our **InLoRA** not only outperforms state-of-the-art PEFT methods, but also matches or even surpasses the performance of full fine-tuning on multiple benchmarks. The code and pre-trained models will be made publicly available.

1 INTRODUCTION

Parameter-Efficient Fine-Tuning (PEFT) is a prevailing solution to considerably reduce the number of trainable parameters on fine-tuning Large Language Models (LLMs) (Vucetic et al., 2022; Tastan et al., 2026). Among various PEFT paradigms, Low-Rank Adaptation (LoRA) (Hu et al., 2022) updates low-rank adaptation matrices while fixing the pre-trained weights of LLMs. To enhance adaptation capacity, it is beneficial to increase adaptation diversity of LLMs by employing multiple parallel task-wise LoRA experts (Liu et al., 2023; Li et al., 2024a; 2026; Huang et al., 2024) or attention modules (Pfeiffer et al., 2021), which are dynamically aggregated via input-dependent routing. Alternatively, hyper-networks like HyperLoRA (Lv et al., 2024; Li et al., 2025) utilize a generative meta-network to synthesize adaptive LoRA weights conditioned on task-specific instructions. These advancements successfully elevate LoRA from *static, task-specific* matrices into *dynamic, task-aware* adaptations, thereby mitigating interference across distinct downstream tasks. However, task-aware LoRAs suffer from weak expert integration (Li et al., 2026) and suppressed task specialization (Li et al., 2024a).

A closer examination of task-aware LoRAs (Pfeiffer et al., 2021; Liu et al., 2023; Li et al., 2024a; Huang et al., 2024; Li et al., 2025; 2026; Lv et al., 2024) reveals a lingering limitation: within any given task or instruction, the constituent base adapters remain *fixed for individual instances*. This prevailing paradigm inadvertently enforces an "instance-agnostic" bottleneck—it relies on a coarse, task routing scheme that fails to capture the fine-grained heterogeneity and conditional data shifts (Zhu et al., 2025) inherent in individual instances. Consequently, when LLMs encounter boundary instances that demand divergent adaptations, this forced module sharing induces severe instance interference (Ivison & Peters, 2022; Ma et al., 2018; Gong et al., 2020). An analogy to human cognition further illustrates this inherent limitation: within the broad domain of mathematical reasoning, algebra and geometry already demand fundamentally distinct logical frameworks. More critically, even within geometry alone, each individual problem presents a unique context that resists uniform treatment. Therefore, instead of relying on a fixed "universal expert", models should dy-

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.



(a) Architecture Comparison (b) Performance Gain

Figure 1: (a) **Architecture comparison**. Unlike LoRA (Hu et al., 2022) and MoE-based LoRAs (Li et al., 2024a; Liu et al., 2023; Huang et al., 2026; Fan et al., 2025), `InLoRA` introduces an instance-customized mechanism to dynamically generate specific adapter matrices (A_x, B_x). (b) **Performance gain** (Δ) relative to LoRA (Hu et al., 2022) across diverse GLUE (Wang et al., 2018) and commonsense reasoning benchmarks.

namically allocate adaptation capacity at an instance-customized granularity, tailoring themselves to the salient features of each incoming query—just as humans dynamically recall relevant prior knowledge for specific problems. A natural question arises: ***can we equip LLMs with instance-customized problem-solving capability by dynamic adaptation?***

To answer this question, three technical challenges must be addressed: ① **Stability–Specificity Dilemma**. When LoRA adapters are dynamically conditioned on individual instances, the effective LLM weights vary across disparate instances, potentially leading to representation collapse. Ensuring stable optimization while preserving instance specificity is non-trivial. ② **Personalization–Efficiency Trade-off**. Naively assigning exclusive LoRA adapters to different instances would cause a prohibitive explosion in storage and GPU memory, as instance numbers can reach millions. Achieving fine-grained personalization without sacrificing scalability remains an open problem. ③ **Instance-Discriminative Bottleneck**. Instance-customized adaptation requires extracting instance-specific cues. In the absence of meaningful priors, the adaptation process suffers from directionless optimization and dynamic instability, easily collapsing to a trivial solution.

In this paper, we propose `InLoRA`, a novel instance-customized LoRA that tackles the above challenges through a decoupled dual-branch design, as illustrated in Fig. 1 (a). A task-specific branch is implemented as vanilla LoRA to capture common structural patterns across instances, ensuring stable optimization and mitigating gradient conflicts (**Challenge** ①). Concurrently, an instance-customized branch dynamically generates per-sample LoRAs via two lightweight projectors, achieving fine-grained personalization without storing individual matrices and thereby sidestepping the personalization–efficiency trade-off (**Challenge** ②). To guide these projectors and avoid directionless optimization, we build a learnable discrete codebook of latent adaptation patterns. Specifically, we cluster features extracted by the pretrained LLM encoder across task instances to initialize the codebook, and then for each instance we aggregate relevant priors using a vector quantization mechanism (van den Oord et al., 2017; Esser et al., 2021) with a Top-K admixture module (Li et al., 2024b). This discrete prior provides stable, instance-discriminative conditioning, effectively preventing LoRA generation collapse (**Challenge** ③). `InLoRA` achieves state-of-the-art results across four primary LLMs, *i.e.*, RoBERTa, Llama-2-7B, Mistral-7B and Gemma-7B, on diverse benchmarks spanning natural language understanding, mathematical reasoning, instruction tuning, and commonsense reasoning. As summarized in Fig. 1 (b), it outperforms vanilla LoRA by 0.7–5.6 points on six benchmarks, while requiring 12–43% fewer trainable parameters across all four LLMs.

2 Related Work

Parameter-Efficient Fine-Tuning (PEFT). Early PEFT methods introduce additive tuning modules into LLMs (Houlsby et al., 2019; Pfeiffer et al., 2021), fine-tune a small subset of LLM parameters (Guo et al., 2021; Vucetic et al., 2022), or learn additional low-rank matrices for LLM fine-tuning (Hu et al., 2022; Liu et al., 2024; Hao et al., 2024). Later, many variants of LoRA (Hu et al.,

2022) have been developed to improve the parameter efficiency, optimization dynamics, and adapter initialization, *etc*. Parameter-sharing VeRA (Kopiczko et al., 2024) and Tied-LoRA (Renduchintala et al., 2024) share the core low-rank matrices across all transformer layers, while learning layer-specific and diagonal vectors. But these strategies suffer from suboptimal tuning performance due to extreme compression on representational capacity. Recent SVD-based PEFT methods decompose pre-trained weight matrices and selectively update different singular subspaces for efficient adaptation: PiSSA (Meng et al., 2024) tunes the *principal* components to accelerate convergence, whereas MiLoRA (Wang et al., 2025b) and KaSA (Wang et al., 2025a) instead tune the *minor* components to preserve core pre-trained knowledge. These methods engage in a task-level debate over superior singular subspace, overlooking a more fundamental reality: in practice, different instances may inherently require adaptation in different subspaces. Our work tackles these limitations by learning instance-customized generation of LoRA parameters, thereby sidestepping the representational constraints of static compression and the task-level singular subspace debate.

Task-aware LoRAs overcome the limitations of standard static adaptation in LoRA. The first category of methods focuses on structural and parameter-level flexibility during training. For example, AdaLoRA (Zhang et al., 2023) adaptively allocates rank budgets across modules. NOLA (Koohpayegani et al., 2023) and VB-LoRA (Li et al., 2024b) construct low-rank matrices from shared bases or a global vector bank purely for parameter compression. The second category shifts toward inference-time task-aware dynamic adaptation. GOAT (Fan et al., 2025) utilizes an SVD-based MoE routing for task priors. Meanwhile, optimization-centric approaches like LoFT (Tastan et al., 2026) attempt to align the dynamics of static adapters with full fine-tuning. These methods inherently overlook the fine-grained heterogeneity of diverse instances: once a task is specified, the adapted parameters are identically applied across all inputs within any given task (Yang et al., 2023).

Recent work has explored directly generating LoRA parameters. One prominent category employs conditional latent diffusion priors (Rombach et al., 2022) over the parameter latent space to generate task-specific LoRA parameters guided by reference images or task identifiers (Wu et al., 2024; Jin et al., 2024; Soro et al., 2024). Additionally, Text-to-LoRA (Charakorn et al., 2025) and Hyperlora (Lv et al., 2024) employ hypernetworks to directly synthesize weights conditioned on natural language task instructions. To better handle multi-task scenarios, ICM-LoRA (Shao et al., 2025) introduces an in-context meta-learning framework using a Conditional Variational Autoencoder (Sohn et al., 2015). More recently, LoRAGen (Huang et al., 2026) proposes a structure-aware weight space learning approach, addressing the non-uniqueness of low-rank decomposition. HyperLoRA (Li et al., 2025) generates LoRA parameters in a low-dimensional linear space to relieve the optimization pressure in the visual domain. However, these methods suffer from critical limitations in generalizability and structural specificity Huang et al. (2026). In this work, our `InLoRA` achieves instance-customized LoRA generation by leveraging discrete codebook as semantic priors and learning lightweight projectors.

3 METHODOLOGY

3.1 PRELIMINARIES

LoRA and MoE-based LoRAs. LoRA (Hu et al., 2022) adapts a frozen pre-trained weight $\mathbf{W}_0 \in \mathbb{R}^{d_{out} \times d_{in}}$ via a low-rank update $\Delta\mathbf{W} = \frac{\alpha}{r}\mathbf{BA}$, where d_{in} and d_{out} denote the input and output dimensions of the linear layer, with trainable matrices $\mathbf{B} \in \mathbb{R}^{d_{out} \times r}$, $\mathbf{A} \in \mathbb{R}^{r \times d_{in}}$ and rank $r \ll \min(d_{out}, d_{in})$. While LoRA drastically cuts tunable parameters compared to full fine-tuning, its learned adapter remains fixed regardless of input variation. To enhance adaptation diversity, MoE-based LoRAs (Liu et al., 2023; Li et al., 2024a; 2026; Huang et al., 2024) employ E parallel expert pairs $\{(\mathbf{A}_i, \mathbf{B}_i)\}_{i=1}^{E}$ and a router, constructing $\Delta\mathbf{W}_{\text{MoE}} = \sum_{i=1}^{E} w_i(\mathbf{B}_i\mathbf{A}_i)$. However, these experts typically specialize along task boundaries after training; consequently, instances within the same task receive near-identical adapter compositions, still collapsing to an *instance-agnostic* regime.

Discrete Codebook Priors via Vector Quantization. To inject a discrete structural prior into continuous latent spaces, Vector Quantization (VQ) maps each continuous representation vector $\mathbf{x}$ to its nearest entry in a learnable codebook $\mathbf{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_J]^\top \in \mathbb{R}^{J \times d_{in}}$ containing J code vectors in $\mathbb{R}^{d_{in}}$. The standard hard assignment is defined as $\mathbf{c}^* = \arg\min_{\mathbf{c}_j \in \mathbf{C}} \|\mathbf{x} - \mathbf{c}_j\|_2$. To circumvent the non-differentiability of $\arg\min$ and enable richer code utilization, a common relaxation replaces

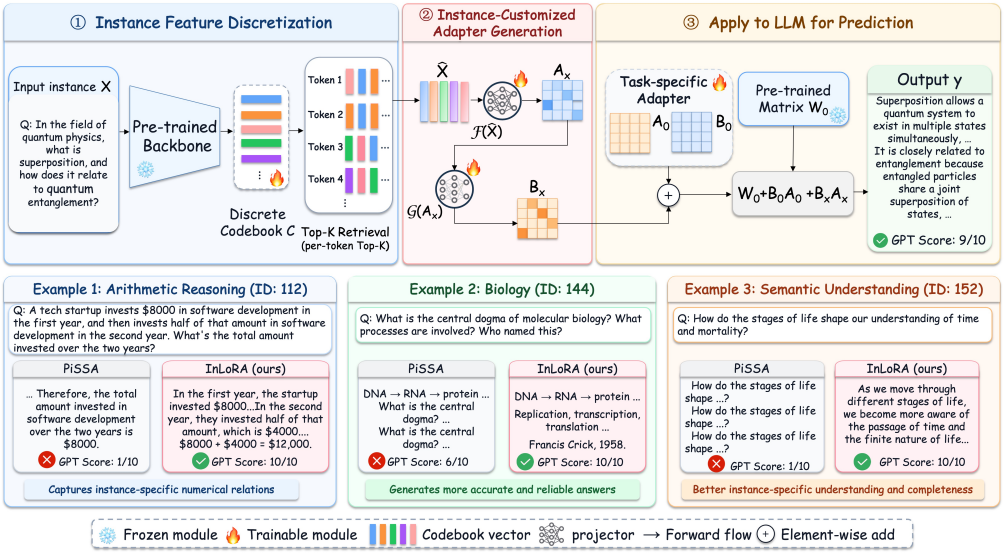


Figure 2: Overall architecture of **InLoRA** and example comparisons with PiSSA (Meng et al., 2024).

the hard assignment with a soft, Top-K aggregation. This soft VQ offers a compact, structured prior that can dynamically re-weight latent codes per instance. In this work, we exploit this property to retrieve instance-specific code combinations and generate input-dependent LoRA adapters.

3.2 The Proposed InLoRA

Architecture of InLoRA. As illustrated in Fig. 1, **InLoRA** employs a decoupled dual-branch architecture for each layer: a *task-specific branch* $(\mathbf{A}_0, \mathbf{B}_0)$ that learns shared knowledge across instances, and an *instance-customized branch* $(\mathbf{A}_x, \mathbf{B}_x)$ that captures per-sample variations for a given instance x with T tokens. For each instance, we generate module-specific $(\mathbf{A}_x, \mathbf{B}_x)$ pairs from local activations, omitting layer and module indices for clarity. The final LoRA update in **InLoRA** is the sum of both branches:

$$\Delta\mathbf{W} = \mathbf{B}_0\mathbf{A}_0 + \mathbf{B}_x\mathbf{A}_x, \tag{1}$$

where $\mathbf{A}_0 \in \mathbb{R}^{r_0 \times d_{in}}$, $\mathbf{A}_x \in \mathbb{R}^{r_x \times d_{in}}$, $\mathbf{B}_0 \in \mathbb{R}^{d_{out} \times r_0}$ and $\mathbf{B}_x \in \mathbb{R}^{d_{out} \times r_x}$. The *task-specific* branch follows the vanilla LoRA formulation, maintaining a pair of low-rank matrices $(\mathbf{A}_0, \mathbf{B}_0)$ that are updated globally across all instances of a task. It provides a stable, task-level foundation and prevents **InLoRA** from overfitting to individual idiosyncrasies. For parameter efficiency, the rank r_0 of $(\mathbf{A}_0, \mathbf{B}_0)$ is set smaller than the rank r of vanilla LoRA. The *instance-customized* branch dynamically synthesizes specific adapters $(\mathbf{A}_x, \mathbf{B}_x)$ for individual inputs. To this end, **InLoRA** first learns a codebook $\mathbf{C}$ as a reservoir of latent adaptation patterns. Given an input feature $\mathbf{X}$, a VQ mechanism with a Top-K Admixture Module (TKAM, Li et al., 2024b) selects and aggregates K most relevant codes, yielding a discretized feature $\hat{\mathbf{X}}$. Then the discretized feature $\hat{\mathbf{X}}$ is fed into two sequential projectors, $\mathcal{F}$ and $\mathcal{G}$, to obtain instance-customized low-rank matrices $\mathbf{A}_x$ and $\mathbf{B}_x$, respectively. The detailed algorithm is provided in Appendix A. We next describe **InLoRA** in detail.

Decoupled Dual-Branch Architecture. Individual instances within a task inherently share common structural and semantic patterns. Ignoring these common patterns can lead to instance-specific overfitting and unstable fine-tuning. To address this, we employ a static, *task-specific* LoRA branch $(\mathbf{A}_0, \mathbf{B}_0)$ as a foundational adaptation to anchor global task knowledge. Meanwhile, we introduce a dynamic, *instance-customized* branch $(\mathbf{A}_x, \mathbf{B}_x)$ to absorb the fine-grained, instance-specific variance without disrupting the global convergence. The final weight update combines both branches as in Eq. (1). During training, the *task-specific* branch $(\mathbf{A}_0, \mathbf{B}_0)$ is updated exactly like vanilla LoRA (Hu et al., 2022), accumulating gradients across all mini-batches to learn the global task adaptation. Concurrently, gradients are backpropagated through the *instance-customized* branch $(\mathbf{A}_x, \mathbf{B}_x)$ to update the codebook and lightweight projectors. The conditioned generation for instance x via continuous projectors ensures that semantically similar instances produce similar $(\mathbf{A}_x, \mathbf{B}_x)$ parameters, which is verified empirically in Appendix B.1. Additionally, combining the dual LoRA branches yields an implicit decoupling effect, tailoring suitable adaptation for each instance. Supporting subspace-similarity results are provided in Appendix B.2.

Offline Codebook Initialization. To avoid generating adapters from an information-sparse single instance, we construct a discrete codebook (Esser et al., 2021) that captures prototypical adaptation patterns for each task as a compact representation of the adaptation space. For a given task dataset with N instances, we feed each instance x with T tokens through a pre-trained LLM and collect the output hidden features $\mathbf{X} \in \mathbb{R}^{T\times d_{in}}$ from the last hidden layer of the LLM, yielding a pool of $N \times T$ token-level vectors of dimension d_{in}. We then apply K-means clustering to these vectors, and the J cluster centers serve as the initial items of the learnable codebook $\mathbf{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_J]^\top \in \mathbb{R}^{J\times d_{in}}$,where $\mathbf{c}_j \in \mathbb{R}^{d_{in}}$ for $j = 1, ..., J$. The codebook initialization typically takes 5–10 minutes per task on a server with an Intel Xeon 8336C CPU and an RTX 4090 GPU. After initialization, the codebook is jointly optimized with the rest of our `InLoRA` during LLM fine-tuning.

Instance-Customized LoRA Generation. To obtain instance-customized adapters, we perform instance feature discretization and instance-customized adapter generation, as illustrated in the first two stages of Fig. 2. For an input instance x with T tokens, we employ the same frozen LLM encoder used in **Offline Codebook Initialization** to extract its *raw* feature $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]^\top \in \mathbb{R}^{T\times d_{in}}$. For each token feature $\mathbf{x}_t$ ($t \in \{1, \ldots, T\}$) within the instance, we compute the Euclidean distances $s_{tj} = \|\mathbf{x}_t - \mathbf{c}_j\|_2$ between it and all latent codes $\mathbf{c}_j$ ($j = 1, ..., J$) in the codebook $\mathbf{C}$.

To capture instance-customized nuances, we utilize a Top-K Admixture Module (TKAM, Li et al., 2024b) to select the K nearest code vectors from the codebook $\mathbf{C}$, and compute their corresponding routing weights $\mathbf{w}_t = \{w_{t,1}, \ldots, w_{t,K}\}$ based on the inverse distances $\{1/s_{tj}\}_{j=1}^{J}$. The weights are then normalized via a softmax function to ensure optimization stability:

$$w_{t,k} = \frac{\exp(\hat{s}_{tk})}{\sum_{m=1}^{K}\exp(\hat{s}_{tm})}, \quad \hat{s}_{tj} \in \text{Top-}K(\{\frac{1}{s_{tj}}\}_{j=1}^{J}), \quad k = 1, \ldots, K, \tag{2}$$

where Top-$K(\{\frac{1}{s_{tj}}\}_{j=1}^{J})$ selects the K largest values in $\{\frac{1}{s_{tj}}\}_{j=1}^{J}$. The discretized token feature for $\mathbf{x}_t$, denoted as $\hat{\mathbf{x}}_t$, is produced by weighted combination of the K selected codes $\{\hat{\mathbf{c}}_k\}_{k=1}^{K}$ from $\mathbf{C}$:

$$\hat{\mathbf{x}}_t = \sum_{k=1}^{K} w_{t,k}\hat{\mathbf{c}}_k, \quad t = 1, \ldots, T. \tag{3}$$

Aggregating these T discretized token feature vectors yields the discretized instance feature $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_T]^\top \in \mathbb{R}^{T\times d_{in}}$, which provides a discrete-informed semantic foundation to generate customized adapters. To prevent codebook collapse, we align the discrete codebook and continuous token feature space, by introducing an auxiliary quantization loss function $\mathcal{L}_{\text{quant}}$ between the input token features and their discretized counterparts: $\mathcal{L}_{\text{quant}} = \frac{1}{T\cdot d_{in}}\sum_{t=1}^{T}\|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2^2$. This ensures joint optimization of discrete codebook and instance-customized adapter generation.

Finally, to bridge the domain gap between the aggregated code and the specific parameter spaces of LoRA, we employ two lightweight MLP projectors, $\mathcal{F}$ and $\mathcal{G}$. Each projector consists of two linear layers with explicit bias terms, interspersed with a ReLU activation function. Specifically, given the discretized instance feature $\hat{\mathbf{X}}$, the input-side adapter $\mathbf{A}_x \in \mathbb{R}^{r_x\times d_{in}}$ and the output-side adapter $\mathbf{B}_x \in \mathbb{R}^{d_{out}\times r_x}$ are generated via the lightweight MLP projectors $\mathcal{F}$ and $\mathcal{G}$, as follows:

$$\mathbf{A}_x = \mathcal{F}(\hat{\mathbf{X}}) = \mathbf{W}_f^2\,\sigma\left(\mathbf{W}_f^1\hat{\mathbf{X}}\right), \quad \mathbf{B}_x = \mathcal{G}(\mathbf{A}_x) = \mathbf{W}_g^2\,\sigma\left(\mathbf{W}_g^1\mathbf{A}_x^\top\right), \tag{4}$$

where $\sigma(\cdot)$ denotes the ReLU activation, all $\mathbf{W}$ matrices are learnable weights with compatible dimensions as specified above, and biases are omitted for brevity. This decoupled projection mechanism ensures that $\mathbf{A}_x$ and $\mathbf{B}_x$ are precisely tailored to their respective roles of input projection and output expansion while maintaining high parameter efficiency.

In practice, to deploy $\Delta\mathbf{W}$ for specific instances, we only need to retain the globally shared parameters, *i.e.*, the *task-specific* adapter matrices $(\mathbf{A}_0, \mathbf{B}_0)$, the shared codebook $\mathbf{C}$, and the projectors $(\mathcal{F}, \mathcal{G})$, completely eliminating the need to explicitly store $\mathbf{A}_x$ and $\mathbf{B}_x$ for each instance. The overall training objective of `InLoRA` is formulated as: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \gamma\mathcal{L}_{\text{quant}}$, where $\mathcal{L}_{\text{task}}$ is the task loss and γ is a hyperparameter. More details are provided in Appendix C.2.

3.3 Theoretical Analysis of `InLoRA`

Let $E^*_{\text{LoRA}}(r)$ and $E^*_{\texttt{InLoRA}}$ denote the optimal squared Frobenius errors of rank-r LoRA and `InLoRA`, respectively. Define $\Delta_{r_0,r} := E^*_{\text{LoRA}}(r_0) - E^*_{\text{LoRA}}(r) \geq 0$ as the penalty from reducing the task-

Table 1: **Results of RoBERTa-large fine-tuned with different PEFT methods on six datasets of the GLUE benchmark (Wang et al., 2018)**. "*" indicates the values reported in (Li et al., 2024b).

Method	# Trainable Parameters	SST-2 (Acc.)	MRPC (Acc.)	CoLA (Mcc.)	QNLI (Acc.)	RTE (Acc.)	STS-B (Pcc.)	All Avg.
FFT*	355.0M	96.4	90.9	68.0	94.7	86.6	92.4	88.2
Tied-LoRA*	0.066M	94.8±0.6	89.7±1.0	64.7±1.2	94.1±0.1	81.2±0.1	90.8±0.3	85.9
FFT	355.0M	96.0±0.4	90.4±0.2	67.4±1.0	94.4±0.1	86.0±1.8	92.3±0.1	87.8
LoRA	0.786M	95.7±0.2	89.2±0.8	65.1±2.4	94.6±0.1	84.6±1.8	91.7±0.2	86.8
VB-LoRA	0.161M	95.7±0.2	89.8±0.7	65.0±1.7	94.4±0.2	83.7±1.0	91.5±0.1	86.7
GOAT	2.231M	94.8±0.5	88.2±0.4	64.4±1.1	93.4±0.3	81.4±2.4	90.5±0.4	85.4
`InLoRA` (*Ours*)	0.688M	96.1±0.4	89.9±0.9	67.0±0.9	94.6±0.1	85.4±0.7	91.9±0.1	87.5

specific rank, and let $G_{\text{gen}} \geq 0$ denote the supremal reduction of the resulting residual attainable by the constrained instance-customized generator.

Theorem 3.1. *Assume that $r_0 \leq r$, that $\operatorname{rank}(\hat{\mathbf{S}}_l) \leq r_0, \operatorname{rank}\left(\mathbf{D}^{(n)}_{\boldsymbol{\theta},l}\right) \leq r_x$ for every n, l, and feasible $\boldsymbol{\theta}$, and that there exists $\boldsymbol{\theta}_0$ such that $\mathbf{D}^{(n)}_{\boldsymbol{\theta}_0,l} = \mathbf{0}$ for all n, l. Then, with $d = r_0 + r_x$,*

$$E_{\text{oracle}}(d) \leq E^*_{\texttt{InLoRA}} \leq E^*_{\text{LoRA}}(r) + \Delta_{r_0,r} - G_{\text{gen}}. \tag{5}$$

Consequently, `InLoRA` achieves a strictly smaller approximation error than rank-r LoRA whenever $G_{\text{gen}} > \Delta_{r_0,r}$. When $r_0 = r$, the penalty vanishes and LoRA containment follows. The complete proof is provided in Appendix D.

4 Experiments

We conduct extensive experiments to evaluate `InLoRA` across four representative downstream settings: (1) natural language understanding on RoBERTa-base and RoBERTa-large (Liu et al., 2019), (2) mathematical reasoning on Mistral-7B (Jiang et al., 2023) and Gemma-7B (Team et al., 2024), (3) instruction tuning on Llama-2-7B (Touvron et al., 2023) and Mistral-7B, and (4) commonsense reasoning on Llama-2-7B (Touvron et al., 2023) and Llama-3.1-8B-Instruct (Grattafiori et al., 2024). We compare `InLoRA` with 15 state-of-the-art PEFT approaches, including LoRA (Hu et al., 2022), VB-LoRA (Li et al., 2024b), GOAT (Fan et al., 2025), MixLoRA (Li et al., 2024a), *etc*. For our experiments, we perform five independent runs with distinct random seeds and report the median unless otherwise specified; results shown as mean ± standard deviation report the mean and standard deviation. All experiments are conducted on a server equipped with two NVIDIA RTX 4090 and two RTX A6000 GPUs. To ensure reproducibility while keeping the main text concise, dataset descriptions are provided in Appendix C.1. Detailed implementation configurations, including hyperparameter settings (*e.g.*, rank, codebook size, and learning rate) and specific training setups, are given in Appendix C.2. Additional experimental results are presented in Appendix E.

4.1 Natural Language Understanding

Datasets and Methods. Experiments are conducted on RoBERTa-base and RoBERTa-large (Liu et al., 2019) with the GLUE benchmark (Wang et al., 2018). We evaluate different PEFT methods across diverse natural language understanding tasks, including similarity detection, paraphrase identification, and inference problems. Following (Li et al., 2024b), we focus on six representative tasks from GLUE: CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), MRPC (Dolan & Brockett, 2005), STS-B (Cer et al., 2017), QNLI (Rajpurkar et al., 2018) and RTE for inference. We compare `InLoRA` with LoRA, Tied-LoRA (Renduchintala et al., 2024), VB-LoRA and GOAT.

Results. Table 1 shows that our `InLoRA` outperforms all comparison methods except full fine-tuning (FFT) on RoBERTa-large. `InLoRA` achieves an average score of 87.5, surpassing LoRA (86.8%) and VB-LoRA (86.7%). Moreover, it achieves consistent gains across datasets, outperforming LoRA and VB-LoRA by 1.9 and 2.0 points on CoLA, while surpassing both VB-LoRA and GOAT by 1.7 and 4.0 points on RTE. Notably, `InLoRA` uses only 0.688M trainable parameters, which is 12.5% fewer than LoRA (0.786M). These results demonstrate that our instance-customized `InLoRA` effectively captures task-specific nuances while maintaining superior parameter efficiency.

Table 2: **Results of Mistral-7B and Gemma-7B fine-tuned with different methods on GSM8K and MATH**. "*" indicates results in (Li et al., 2024b).

Model	Method	# Parameters	GSM8K	MATH
MISTRAL-7B	Full-FT*	7242M	67.02	18.60
	LoRA	20.97M	62.09	19.14
	VB-LoRA	0.65M	53.30	21.04
	InLoRA (*Ours*)	14.91M	**63.68**	**21.38**
GEMMA-7B	Full-FT*	8538M	71.34	22.74
	LoRA	25.00M	66.64	24.20
	VB-LoRA	0.67M	59.21	25.18
	InLoRA (*Ours*)	15.96M	**68.54**	**27.58**

Table 3: **Results of Llama2-7B and Mistral-7B fine-tuned with different methods on MT-Bench**. "†" indicates results in (Li et al., 2024b; Meng et al., 2024).

	Method	# Parameters	Score
LLAMA2 7B	w/o FT†	-	4.79
	LoRA	40M	4.91
	VB-LoRA	1M	4.78
	InLoRA (*Ours*)	23M	**4.99**
MISTRAL-7B	Full-FT†	7242M	4.95
	LoRA†	168M	4.90
	PiSSA†	168M	5.34
	PiSSA	42M	6.34
	InLoRA (*Ours*)	26M	**7.18**

4.2 MATHEMATICAL REASONING

Datasets and Methods. We evaluate InLoRA on mathematical reasoning tasks, which heavily rely on long-form arithmetic reasoning traces and complex logical deductions. Experiments are performed on Mistral-7B and Gemma-7B, which are fine-tuned on MetaMathQA (Yu et al., 2024) and evaluated on GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). We compare InLoRA with LoRA (Hu et al., 2022) and VB-LoRA (Li et al., 2024b).

Results. Table 2 summarizes the mathematical reasoning results. InLoRA consistently outperforms LoRA and VB-LoRA across both models and datasets with fewer trainable parameters, approximately 71% of LoRA's parameter count on Mistral-7B and 64% of Gemma-7B. Specifically, on Mistral-7B, InLoRA achieves a 63.68% accuracy on GSM8K, outperforming VB-LoRA by 10.38 points. Similar trends are observed on Gemma-7B, where InLoRA surpasses VB-LoRA by 9.33 points on GSM8K. Furthermore, in the more challenging MATH dataset, InLoRA outperforms all comparison methods, achieving 27.58% on Gemma-7B, which represents an improvement of up to 3.38 points over LoRA. These results highlight the superior capability of InLoRA in handling complex reasoning tasks through its dynamic instance-customized adaptation.

4.3 INSTRUCTION TUNING

Datasets and Methods. Instruction tuning equips pre-trained LLMs with the capability to follow user prompts and align with human intentions (Ouyang et al., 2022). We fine-tune Llama-2-7B on the Cleaned Alpaca Dataset (Taori et al., 2023) and Mistral-7B on the WizardLM-Evol-Instruct dataset (Xu et al., 2024). After the fine-tuning phase, we evaluate both models using MT-Bench (Zheng et al., 2023), a challenging benchmark comprising 80 multi-turn questions designed to rigorously test conversational and reasoning capabilities. We compare InLoRA with LoRA (Hu et al., 2022), VB-LoRA (Li et al., 2024b) and PiSSA (Meng et al., 2024).

Evaluation Protocol. We use GPT-4 as an impartial judge to score the first-turn responses generated on MT-Bench, similar to (Li et al., 2024b). The scores are on a scale of $[1, 10]$. For a fair comparison, we re-evaluate all baselines, *e.g.*, LoRA, PiSSA and VB-LoRA, concurrently within the same temporal window to mitigate scoring inconsistencies caused by periodic GPT-4 updates.

Results. Table 3 presents the instruction-tuning results on MT-Bench. InLoRA demonstrates exceptional performance on both Mistral-7B and Llama-2-7B. On Llama-2-7B, InLoRA uses 23M parameters, which is only 58% of LoRA's parameter budget, while delivering superior performance. On Mistral-7B, InLoRA achieves a score of 7.18, significantly outperforming PiSSA (6.34) by 0.84 points, while utilizing only approximately 62% of PiSSA's trainable parameters. Selected first-turn MT-Bench comparisons are shown in Fig. 2. These results validate that the instance-customized adaptation mechanism of InLoRA is highly effective for open-ended instruction following, delivering state-of-the-art performance with high parameter efficiency.

4.4 COMMONSENSE REASONING

Datasets and methods. To evaluate the reasoning capabilities of InLoRA, we fine-tune Llama-2-7B on eight widely recognized commonsense reasoning tasks: BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi

Table 4: **Comparison results of Llama-2-7B across eight commonsense reasoning datasets**. Results for referenced baselines (*e.g.*, DoRA (Liu et al., 2024), PiSSA (Meng et al., 2024), MiLoRA (Wang et al., 2025b), LoRA-Dash (Si et al., 2025), NEAT (Zhong et al., 2024), HydraLoRA (Tian et al., 2024)) are directly taken from (Fan et al., 2025).

Method	# Params(%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
ChatGPT	/	73.10	85.40	68.50	78.50	66.10	89.80	79.90	74.80	77.01
Single LoRA Methods										
LoRA	1.17	**74.98**	86.29	81.32	85.40	84.85	86.11	67.41	84.08	81.30
DoRA	0.84	71.80	83.10	79.90	89.10	83.00	84.50	71.00	81.20	80.45
PiSSA	0.84	67.60	78.10	78.40	76.60	78.00	75.80	60.20	75.60	73.78
VB-LoRA	4.54	74.16	84.66	81.12	85.96	83.11	85.35	64.59	81.80	80.09
MiLoRA	0.84	67.60	83.80	80.10	88.20	82.00	82.80	68.80	80.60	79.24
LoRA-Dash	0.84	71.00	75.70	79.30	91.10	78.60	84.20	69.80	78.80	78.56
NEAT	0.84	71.70	83.90	80.20	88.90	84.30	86.30	71.40	83.00	81.21
MoE-based LoRA Methods										
HydraLoRA	0.84	72.78	84.06	79.68	80.34	**86.66**	87.12	**72.35**	**86.00**	81.12
GOAT	0.72	74.37	85.15	81.22	88.16	85.24	86.87	69.20	83.60	81.73
MixLoRA	3.00	73.00	82.59	77.53	**92.52**	76.72	77.35	54.61	77.60	76.49
`InLoRA` (*Ours*)	0.63	74.29	**86.45**	**82.24**	90.95	86.35	**87.33**	71.59	85.00	**83.03**

et al., 2021), ARC-e (Clark et al., 2018), ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). We follow the evaluation protocol established by MixLoRA (Li et al., 2024a).

Results. As shown in Table 4, `InLoRA` achieves a substantial lead of 6.54 points in the average score compared to MixLoRA. It attains notable performance leaps of 16.98 and 9.63 points on the challenging ARC-c and WinoGrande datasets, respectively. When compared to GOAT, the strongest competitor, `InLoRA` still surpasses it on 7 out of the 8 benchmarks, with robust improvements of 2.79 and 1.40 points on the HellaSwag and OBQA tasks, respectively. Finally, `InLoRA` uses only 0.63% trainable parameters of Llama-2-7B, substantially lower than LoRA (1.17%) and GOAT (0.72%). These results validate that `InLoRA` not only achieves the highest average score, but also demonstrates exceptional versatility and stability across diverse reasoning tasks.

4.5 Ablation Study

To understand `InLoRA`'s working mechanism, we conduct ablation studies on `InLoRA` using RoBERTa-large with the CoLA and MRPC datasets. The results are summarized in Table 5.

Task-specific Branch $(\mathbf{A}_0, \mathbf{B}_0)$. A core design of `InLoRA` is decoupling individual idiosyncrasies from task-specific common patterns. Removing this branch noticeably degrades performance, confirming that it preserves foundational common knowledge, complements the flexibility of the instance-customized branch $(\mathbf{A}_x, \mathbf{B}_x)$, and prevents instance overfitting.

Codebook Property. We further study the role and quality of codebook. *Without codebook* ("No"), we directly feed the raw instance features into the projectors and `InLoRA` obtains degraded performance, verifying the necessity of discrete codebook prior. *Fixing the codebook* after initialization ("Fixed") also hurts, indicating that end-to-end fine-tuning of the codebook is crucial for adapting to the target distribution. *Randomly initializing* the codebook without K-means ("Random") remains below the default, demonstrating the importance of a semantic codebook initialization.

Codebook Size J. Decreasing J from 270 to 90 degrades scores on both CoLA (66.83% $\rightarrow$ 66.76%) and MRPC (90.20% $\rightarrow$ 89.95%) due to capturing fewer semantic prototypes. However, increasing J further to 540 or 1024 yields diminishing returns and performance fluctuations, possibly due to routing ambiguity in expanded codebook space. More discussions are provided in Appendix E.3.

K in Top-K. Top-K controls retrieval sparsity. $K = 1$ yields hard routing and sparse updates that may hinder optimization, whereas $K = 4$ has task-dependent effects. We use $K = 2$ for consistently strong performance with sparse retrieval.

4.6 Analysis of `InLoRA`

Instance-Adapter Correlation. We analyze whether semantically similar instances generate similar adapters $(\mathbf{A}_x, \mathbf{B}_x)$. For each instance x, we compute its raw instance feature $\mathbf{X}$ and the correspondingly generated matrices $\mathbf{A}_x$, $\mathbf{B}_x$, and $\Delta\mathbf{W}_x = \mathbf{B}_x\mathbf{A}_x$. As shown in Fig. 3, a consistent

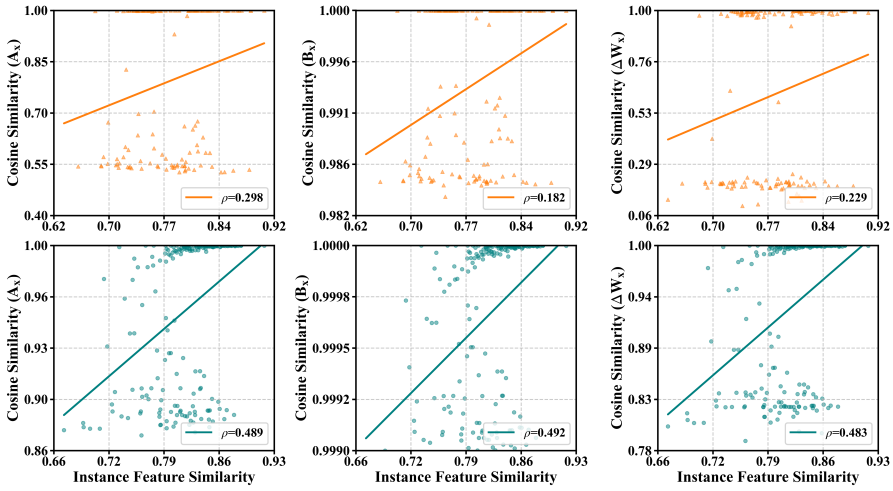


Figure 3: **Correlation between the similarity of instances and correspondingly generated adapters.** We plot pairwise cosine similarity of instance features (x-axis) *vs.* that of $\mathbf{A}_x$, $\mathbf{B}_x$, or $\Delta\mathbf{W}_x$ (y-axis) on RTE and QNLI (RoBERTa-large, layer 12). Spearman correlation ρ is reported.

Table 5: **Ablation results of `InLoRA` using RoBERTa-large on CoLA and MRPC**. `InLoRA` has a task-specific branch $(\mathbf{A}_0, \mathbf{B}_0)$ and an instance-customized branch $(\mathbf{A}_x, \mathbf{B}_x)$.

Variant	$(\mathbf{A}_0, \mathbf{B}_0)$	**Codebook C**	**Top-K**	**# Params**	**CoLA**	**MRPC**
`InLoRA`	✓	Learn, $J = 270$	2	0.69M	66.83	**90.20**
task-specific Branch	×	Learn, $J = 270$	2	0.30M	65.56	88.48
Codebook Property	✓	No, $J = 0$	–	0.41M	65.86	88.48
	✓	Fixed, $J = 270$	2	0.41M	65.79	89.46
	✓	Random, $J = 270$	2	0.69M	66.28	88.72
Codebook Size	✓	Learn, $J = 90$	2	0.50M	66.76	89.95
	✓	Learn, $J = 540$	2	0.96M	66.79	89.21
	✓	Learn, $J = 1024$	2	1.46M	66.27	89.95
K in Top-K	✓	Learn, $J = 270$	1	0.69M	66.02	88.48
	✓	Learn, $J = 270$	4	0.69M	**67.75**	89.95

positive Spearman correlation ($\rho > 0$) is observed, with QNLI achieving $\rho \approx 0.48 \sim 0.49$. Please refer to Appendix B.1 for more detailed explanation.

Subspace Orthogonality (Implicit Decoupling). We measure the normalized subspace similarity between the weight matrices of *task-specific* and *instance-customized* branches (see Fig. 4 in Appendix B.2). The similarity scores are consistently near zero, which empirically validates the *implicit decoupling* effect in `InLoRA`, where the task-specific branch anchors global task semantics while the instance-customized one captures individual idiosyncrasies for instance customization.

5 Conclusion

In this paper, we introduced `InLoRA`, a flexible parameter-efficient fine-tuning (PEFT) method that addresses the instance-agnostic limitations of static low-rank adaptation. To our knowledge, this is the first work to meaningfully exploit discrete codebook priors for the on-the-fly generation of per-instance LoRA adapters. By anchoring these instance-customized adaptations to a task-specific one, `InLoRA` explicitly captures fine-grained instance-specific variance while preserving stable learning dynamics across the full task distribution. Extensive experiments spanning natural language understanding, mathematical reasoning, instruction tuning, and commonsense reasoning demonstrate that `InLoRA` consistently outperforms vanilla LoRA and a range of state-of-the-art PEFT competitors, substantially closing the performance gap between PEFT and full fine-tuning.

AI USE STATEMENT

In this work, we used generative AI tools to assist with manuscript format checking, translation, and language editing, and to provide feedback on the presentation, notational consistency, and logical completeness of theoretical statements and proofs. All AI-assisted content, including mathematical derivations and theoretical claims, was independently reviewed and verified by the authors. The authors take full responsibility for the final content of this work, including any text, claims, or other artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

All authors have read and adhered to the ICLR Code of Ethics. Our empirical evaluation is conducted exclusively on publicly available and widely used academic benchmark datasets. These datasets do not contain personally identifiable or sensitive information. This work does not introduce any foreseeable negative applications or exacerbate biases that may already be present in general-purpose pretrained models.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide the complete hyperparameter settings, dataset preprocessing and task construction procedures, implementation details, evaluation protocols, random-seed settings, and computational environment in Section 4 and Appendix C. Pseudocode for the proposed algorithm is provided in Appendix A. The complete implementation and scripts required to reproduce our experiments are included in the supplementary material. The assumptions and complete proofs of our theoretical results are presented in Appendix D.

REFERENCES

Yonatan Bisk, Rowan Zellers, Ronan LeBras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Association for the Advancement of Artificial Intelligence*, pp. 7432–7439, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/6239.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th international workshop on semantic evaluation (SemEval-2017)*, pp. 1–14, 2017.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-loRA: Instant transformer adaption. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=zWskCdu3QA.

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 *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. 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the third international workshop on paraphrasing (IWP2005)*, 2005.

Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873–12883, 2021.

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 *Forty-second International Conference on Machine Learning*, pp. 15804–15832, 2025. URL https://openreview.net/forum?id=SUxq4HeIAd.

Chenggong Gong, Jianfei Yu, and Rui Xia. Unified feature and instance based domain adaptation for aspect-based sentiment analysis. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 7035–7045, 2020.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Demi Guo, Alexander M Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: Long papers)*, pp. 4884–4896, 2021.

Yongchang Hao, Yanshuai Cao, and Lili Mou. Flora: Low-rank adapters are secretly gradient compressors. In *International Conference on Machine Learning*, pp. 17554–17571, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

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, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. Lorahub: Efficient cross-task generalization via dynamic loRA composition. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=TrloAXEJ2B.

Hao Huang, Jingtao Ding, Mengqi Liao, Xin Wang, Jinyang Ban, Yuan Yuan, Huaiyu Wan, and Yong Li. LoRAGen: Structure-aware weight space learning for loRA generation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mrafO7aTYj.

Hamish Ivison and Matthew E Peters. Hyperdecoders: Instance-specific decoders for multi-task nlp. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 1715–1730, 2022.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, and Lucile Saulnier. Mistral-7b. *arXiv preprint arXiv:2310.06825*, 2023.

Xiaolong Jin, Kai Wang, Dongwen Tang, Wangbo Zhao, Yukun Zhou, Junshu Tang, and Yang You. Conditional lora parameter generation. *arXiv preprint arXiv:2408.01415*, 2024.

Soroush Abbasi Koohpayegani, KL Navaneet, Parsa Nooralinejad, Soheil Kolouri, and Hamed Pirsiavash. Nola: Compressing lora using linear combination of random basis. *arXiv preprint arXiv:2310.02556*, 2023.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

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. *arXiv preprint arXiv:2404.15159*, 2024a. URL https://doi.org/10.48550/arXiv.2404.15159.

Mengtian Li, Jinshu Chen, Wanquan Feng, Bingchuan Li, Fei Dai, Songtao Zhao, and Qian He. HyperLoRA: Parameter-efficient adaptive generation for portrait synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13114–13123, 2025.

Wenbing Li, Zikai Song, Hang Zhou, Junqing Yu, Yunyao Zhang, and Wei Yang. LoRA-mixer: Coordinate modular loRA experts through serial attention routing. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=GMP1S4R6Ke.

Yang Li, Shaobo Han, and Shihao Ji. Vb-lora: extreme parameter efficient fine-tuning with vector banks. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pp. 16724–16751, 2024b.

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. *arXiv preprint arXiv:2310.18339*, 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.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Chuancheng Lv, Lei Li, Shitou Zhang, Gang Chen, Fanchao Qi, Ningyu Zhang, and Hai-Tao Zheng. Hyperlora: Efficient cross-task generalization via constrained low-rank adapters generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 16376–16393, 2024.

Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1930–1939, 2018. URL https://doi.org/10.1145/3219819.3220007.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, pp. 121038–121072, 2024. URL https://openreview.net/forum?id=6ZBHIEtdP4.

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 *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2381–2391, 2018.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: main volume*, pp. 487–503, 2021.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you dont know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, 2018.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. Tied-lora: Enhancing parameter efficiency of lora with weight tying. *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8694–8705, 2024.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10674–10685, 2022. URL https://doi.org/10.1109/CVPR52688.2022.01042.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.

Yihua Shao, Minxi Yan, Yang Liu, Siyu Chen, Wenjie Chen, Xinwei Long, Ziyang Yan, Lei Li, Chenyu Zhang, Nicu Sebe, et al. In-context meta lora generation. *arXiv preprint arXiv:2501.17635*, 2025.

Chongjie Si, Zhiyi Shi, Shifan Zhang, Xiaokang Yang, Hanspeter Pfister, and Wei Shen. Unleashing the power of task-specific directions in parameter efficient fine-tuning. In *International Conference on Learning Representations*, volume 2025, pp. 78837–78869, 2025.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *The Twenty-ninth Annual Conference on Neural Information Processing Systems*, pp. 3483–3491, 2015.

Bedionita Soro, Bruno Andreis, Hayeon Lee, Wonyong Jeong, Song Chong, Frank Hutter, and Sung Ju Hwang. Diffusion-based neural network weights generation. *arXiv preprint arXiv:2402.18153*, 2024.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Stanford Alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Nurbek Tastan, Stefanos Laskaridis, Martin Takáč, Karthik Nandakumar, and Samuel Horváth. LoFT: Low-rank adaptation that behaves like full fine-tuning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=86P3sb1dpr.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Cheng-zhong Xu. Hydralora: An asymmetric lora architecture for efficient fine-tuning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, volume 37, pp. 9565–9584, 2024.

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.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6309–6318, 2017.

Danilo Vucetic, Mohammadreza Tayaranian, Maryam Ziaeefard, James J Clark, Brett H Meyer, and Warren J Gross. Efficient fine-tuning of bert models on the edge. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1838–1842. IEEE, 2022.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pp. 353–355, 2018.

Fan Wang, Juyong Jiang, Chansung Park, Sunghun Kim, and Jing Tang. KaSA: Knowledge-aware singular-value adaptation of large language models. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=OQqNieeivq.

Hanqing Wang, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. Milora: Harnessing minor singular components for parameter-efficient llm finetuning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4823–4836, 2025b.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *ACL (1)*, pp. 13484–13508, 2023. URL https://doi.org/10.18653/v1/2023.acl-long.754.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

Yujia Wu, Yiming Shi, Jiwei Wei, Chengwei Sun, Yang Yang, and Heng Tao Shen. Difflora: Generating personalized low-rank adaptation weights with diffusion. *arXiv preprint arXiv:2408.06740*, 2024.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=CfXh93NDgH.

Chunjin Yang, Fanman Meng, Shuai Chen, Mingyu Liu, and Runtong Zhang. Instance-wise adaptive tuning and caching for vision-language models. In *European Conference on Artificial Intelligence*, pp. 2834–2841, 2023.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=N8N0hgNDRt.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=lq62uWRJjiY.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 46595–46623, 2023.

Yibo Zhong, Haoxiang Jiang, Lincan Li, Ryumei Nakada, Tianci Liu, Linjun Zhang, Huaxiu Yao, and Haoyu Wang. Neat: Nonlinear parameter-efficient adaptation of pre-trained models. *arXiv preprint arXiv:2410.01870*, 2024. URL https://doi.org/10.48550/arXiv.2410.01870.

Yuxuan Zhu, Cong Fu, Yabo Ni, Anxiang Zeng, and Yuan Fang. A probabilistic framework for temporal distribution generalization in industry-scale recommender systems. *arXiv preprint arXiv:2511.21032*, 2025. URL https://doi.org/10.48550/arXiv.2511.21032.

A ALGORITHM OF InLoRA

Algorithm 1 Instance-Customized Adaptation Generation of InLoRA

Input: Input feature matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]^\top \in \mathbb{R}^{T \times d_{in}}$, Pre-trained codebook $\mathbf{C} \in \mathbb{R}^{J \times d_{in}}$, Base matrices $\mathbf{A}_0 \in \mathbb{R}^{r_0 \times d_{\text{in}}}$, $\mathbf{B}_0 \in \mathbb{R}^{d_{\text{out}} \times r_0}$, $\mathbf{W}_f^1 \in \mathbb{R}^{r_x \times T}$, $\mathbf{W}_f^2 \in \mathbb{R}^{r_x \times r_x}$, $\mathbf{W}_g^1 \in \mathbb{R}^{r_x \times d_{\text{in}}}$, $\mathbf{W}_g^2 \in \mathbb{R}^{d_{\text{out}} \times r_x}$, number of selected vectors used in the Top-K admixture module.
Output: Instance-specific weight update matrix $\Delta\mathbf{W}$.

Initialization: Initialize $\mathbf{A}_0 \sim \mathcal{N}(0, \sigma^2)$, $\mathbf{B}_0 = \mathbf{0}$.
Compute similarity scores $s_{tj} = \|\mathbf{x}_t - \mathbf{c}_j\|_2$, $t = 1, 2, ..., T$, $j = 1, 2, ..., J$.
Compute sparse routing weights via Top-K softmax: $\mathbf{w}_t = \{w_{t,1}, \ldots, w_{t,K}\}$,
    $w_{t,k} = \text{softmax}\,(\hat{s}_{tk})$, where $\hat{s}_{tj} \in \text{Top-}K(\{\frac{1}{s_{tj}}\}_{j=1}^{J}), k = 1, \ldots, K$.
Compute token-wise discretization: $\hat{\mathbf{x}}_t = \sum_{k=1}^{K} w_{t,k}\hat{\mathbf{c}}_k, \quad t = 1, \ldots, T$.
Aggregate to form the discretized instance feature: $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_T]^\top \in \mathbb{R}^{T \times d_{in}}$.
Generate instance-customized adapter matrix $\mathbf{A}_x$ via the first projector:
    $\mathbf{A}_x = \mathcal{F}(\hat{\mathbf{X}}) = \mathbf{W}_f^2\,\sigma(\mathbf{W}_f^1\hat{\mathbf{X}}) \in \mathbb{R}^{r_x \times d_{\text{in}}}$.
Generate instance-customized adapter matrix $\mathbf{B}_x$ by projecting matrix $\mathbf{A}_x$:
    $\mathbf{B}_x = \mathcal{G}(\mathbf{A}_x) = \mathbf{W}_g^2\,\sigma(\mathbf{W}_g^1\mathbf{A}_x^\top) \in \mathbb{R}^{d_{\text{out}} \times r_x}$.
Return $\Delta\mathbf{W} = \mathbf{B}_0\mathbf{A}_0 + \mathbf{B}_x\mathbf{A}_x$.

B CORRELATION ANALYSIS OF INSTANCE-CUSTOMIZED ADAPTERS IN InLoRA

Here, we provide detailed empirical evidence to characterize the properties of our dynamically generated instance-customized adapters. Specifically, we analyze their continuous correlation with the input instances (Sec. B.1) and their geometric relationship with the task-specific adapters (Sec. B.2).

B.1 INSTANCE-ADAPTER CORRELATION

To investigate the interpretability of our dynamic routing mechanism, we analyze the semantic continuity between the input instance space and the generated parameter space. We conducted a pairwise similarity analysis on the RTE and QNLI datasets by sampling 400 instances and calculating the cosine similarity between input representations $\mathbf{x}$ and their corresponding generated matrices ($\mathbf{A}_x$, $\mathbf{B}_x$, and $\Delta\mathbf{W}_x = \mathbf{B}_x\mathbf{A}_x$).

As illustrated in Fig. 3, we observe a consistent positive Spearman correlation ($\rho > 0$) across all matrices. Notably, the QNLI dataset exhibits a strong positive correlation ($\rho \approx 0.48 \sim 0.49$). These findings validate that our method effectively preserves instance-customized semantic relationships within the dynamic weight space.

This result provides a meaningful extension to the observations in LoRAGen (Huang et al., 2026). LoRAGen notes that for unconstrained generators, individual low-rank matrices ($\mathbf{A}$ and $\mathbf{B}$) often show no correlation with task description embeddings due to the non-uniqueness of matrix decomposition (*e.g.*, susceptibility to arbitrary rotations or rescaling). Our work extends this discussion to the instance level: by conditioning generation on instance feature $\mathbf{X}$ via a continuous mapping, we ensure a stable, smooth low-rank space. Thus, while each $\mathbf{X}$ produces a distinct pair $(\mathbf{A}_x, \mathbf{B}_x)$, semantically similar inputs naturally yield similar weights. Our approach successfully maps instance-customized nuances into a smooth and interpretable parameter manifold, offering a robust alternative to the generation instability typical of unconstrained baselines.

B.2 SUBSPACE ORTHOGONALITY (IMPLICIT DECOUPLING)

To empirically validate the **implicit decoupling** effect between our task-specific common branch and instance-customized branch, we measure the similarity between the subspaces spanned by their learned weight matrices. Following the normalized subspace similarity metric based on the Grassmann distance introduced in LoRA (Hu et al., 2022), we compute the overlap between the top singular vectors of the respective branches.

Table 6: **Overview of task descriptions and dataset statistics** on the GLUE benchmark.

Corpus	Task	# Train	# Val	# Test	# Labels	Metrics	Domain
			Single-Sentence Tasks				
CoLA	Acceptability	8.55k	1.04k	1.06k	2	Matthews Corr.	misc.
SST-2	Sentiment	67.3k	872	1.82k	2	Accuracy	Movie reviews
			Similarity and Paraphrase Tasks				
MRPC	Paraphrase	3.67k	408	1.73k	2	Accuracy/F1	News
STS-B	Sentence similarity	5.75k	1.5k	1.38k	1	Pearson/Spearman Corr.	misc.
			Inference Tasks				
QNLI	QA/NLI	105k	5.46k	5.46k	2	Accuracy	Wikipedia
RTE	NLI	2.49k	277	3k	2	Accuracy	News & Wikipedia

For this analysis, we set $r_0 = r_x = 8$ so that the two branches have matched ranks. We analyze three pairs of learned matrices: the down-projection factors $\mathbf{A}_0$ and $\mathbf{A}_x$, the up-projection factors $\mathbf{B}_0$ and $\mathbf{B}_x$, and the composed adaptation matrices $\Delta\mathbf{W}_0 = \mathbf{B}_0\mathbf{A}_0$ and $\Delta\mathbf{W}_x = \mathbf{B}_x\mathbf{A}_x$. We compare the right singular subspaces for the down-projection factors and composed updates, and the left singular subspaces for the up-projection factors. This choice measures the input-space directions captured by the down-projection factors and the composed adaptation matrices and the output-space directions captured by up-projection factors. Taking the up-projection factors as an example, let $\mathbf{U}_0^{(i)}$ and $\mathbf{U}_x^{(j)}$ contain the leading i and j left singular vectors of $\mathbf{B}_0$ and $\mathbf{B}_x$, respectively. Following Hu et al. (2022), their normalized subspace similarity is defined as

$$\phi(\mathbf{B}_0, \mathbf{B}_x, i, j) = \frac{\left\| \left(\mathbf{U}_0^{(i)}\right)^\top \mathbf{U}_x^{(j)} \right\|_F^2}{\min(i, j)} \in [0, 1], \tag{6}$$

where $\mathbf{U}^i$ represents the columns of $\mathbf{U}$ corresponding to the top-i singular vectors, and $\|\cdot\|_F^2$ denotes the squared Frobenius norm. The similarities between $\mathbf{A}_0$ and $\mathbf{A}_x$ and between $\Delta\mathbf{W}_0$ and $\Delta\mathbf{W}_x$ are computed analogously using their right singular vectors. We report the similarity averaged over all evaluated instances. We aim to quantify how much of the subspace spanned by the top-i singular vectors in $\mathbf{U}_{\text{task}}$ is contained within the subspace spanned by the top-j singular vectors of $\mathbf{U}_{\text{instance}}$.

This similarity measure is inherently tied to the standard Projection Metric (Grassmann distance). The value of $\phi(\cdot)$ lies in $[0, 1]$. Specifically, $\phi = 0$ indicates orthogonal subspaces, whereas $\phi = 1$ indicates that the smaller subspace is fully contained in the larger one; when $i = j$, the two subspaces coincide.

As illustrated in Fig. 4, we visualize the subspace similarity ϕ across the top singular vectors ($1 \leq i, j \leq 8$) for all three matrix pairs. Remarkably, the similarity scores remain consistently close to 0, with the maximum value not exceeding 0.058. This exceptionally low similarity provides compelling empirical evidence that the task and dynamic branches learn highly orthogonal feature representations. Consequently, it validates that our architecture successfully achieves **implicit decoupling**: the task branch stably anchors the macro-task semantics, while the dynamic branch independently models the idiosyncratic instance nuances without feature redundancy.

C Experimental Details

C.1 Descriptions of Datasets

GLUE Benchmark Datasets Table 6 provides an overview of the task descriptions and dataset statistics for the GLUE benchmark used in NLU evaluation.

Commonsense Reasoning Datasets Table 7 presents detailed information about the datasets used in our experiments, including their task names, respective domains, the number of training and test sets, and task types. All datasets are downloaded from HuggingFace using the `datasets` library.

Instruction Tuning Datasets Alpaca (Taori et al., 2023) is a well-known instruction dataset that contains 52k instruction-following demonstrations generated by **text-davinci-003**. These data are

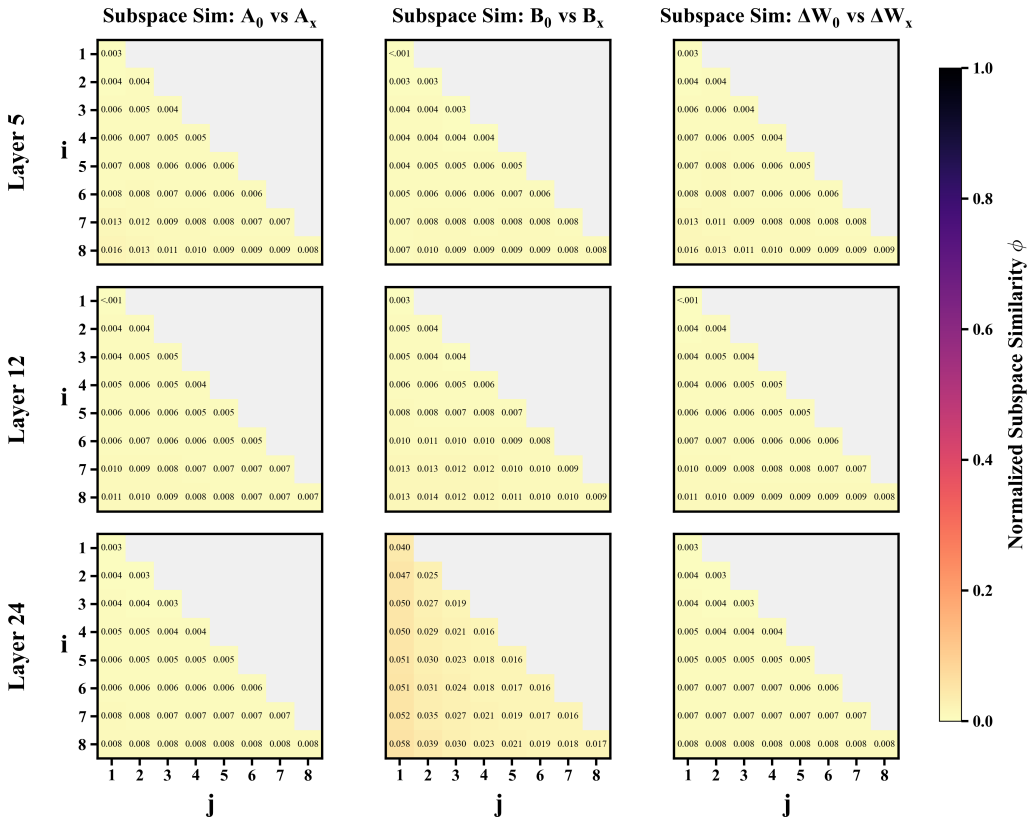


Figure 4: **Normalized subspace similarity between task-specific and instance-customized branches.** The visualization illustrates the similarities for the query ($\mathbf{W}_q$) projector in RoBERTa-large fine-tuned on the CoLA dataset. We report the subspace overlap of the down-projection weights ($\mathbf{A}_0$ *vs.* $\mathbf{A}_x$), up-projection weights ($\mathbf{B}_0$ *vs.* $\mathbf{B}_x$), and the composed adaptation matrices ($\Delta\mathbf{W}_0$ vs. $\Delta\mathbf{W}_x$) across the 5th, 12th, and 24th layers. The consistently near-zero similarity scores across all configurations empirically demonstrate the *implicit decoupling* effect, indicating that the instance-customized branch learns highly orthogonal feature representations relative to the task-specific branch.

Table 7: **Description of commonsense reasoning datasets** used in our experiments.

Task Name	Domain	# Train	# Test	Task Type
BoolQ	Wikipedia	9,427	3,270	Text Classification
ARC-E	Natural Science	2,250	2,380	Question Answering
ARC-C	Natural Science	1,120	1,170	Question Answering
OpenBookQA	Science Facts	4,957	500	Question Answering
PIQA	Physical Interaction	16,100	1,840	Question Answering
SIQA	Social Interaction	33,410	1,954	Question Answering
HellaSwag	Video Caption	39,905	10,042	Sentence Completion
WinoGrande	Winograd Schemas	9,248	1,267	Fill in the Blank

synthesized using an improved self-instruct method (Wang et al., 2023). The dataset is designed for instruction-tuning LLMs to improve their ability to follow instructions. Each sample includes an instruction, an input (if applicable), and an output. The instruction describes the targeted task to be performed by the model. Each of the 52k instructions is unique. The input can represent the optional input to the task or serve as the additional context to the corresponding instruction. The output is the response generated by *text-davinci-003* to the associated instruction. The text is the formatted combination of the instruction, input, and output, using the prompt template for fine-tuning models.

Table 8: **Hyperparameter configurations of `InLoRA` for natural language understanding experiments on the GLUE benchmark.** Fine-tuning targets the query and value projection modules and is conducted using either two NVIDIA RTX 4090 GPUs or two NVIDIA RTX A6000 GPUs.

Model	Hyperparameter	SST-2	MRPC	CoLA	QNLI	RTE	STS-B
Shared	Optimizer	AdamW					
	Warmup Ratio	0.06					
	LR Schedule	Linear					
	LR (Projector)	1E-2					
	LR (Codebook)	5E-3					
	LR (Base A_0, B_0)	1E-3					
	Quantization Loss Wt. (γ)	1E-4					
BASE	Epochs	60	30	80	25	160	80
	Base Model LR (Head)	4E-3	4E-3	2E-2	1E-2	2E-2	2E-2
	Codebook Size	270					
	Ranks	$r = r_x = 8, \quad r_0 = 4$					
	Max Seq. Len.	512					
	Batch Size Per GPU	16					
LARGE	Epochs	20	40	40	20	40	40
	Base Model LR (Head)	3E-3	3E-3	3E-3	2E-3	2E-3	6E-3
	Codebook Size	270					
	Ranks	$r = r_x = 8, \quad r_0 = 4$					
	Max Seq. Len.	128					
	Batch Size Per GPU	32					

MT-bench (Zheng et al., 2023) contains 80 predefined open-ended questions across diverse domains such as writing, roleplay, reasoning, math, coding, extraction, STEM, and humanities. These challenging questions are designed to automatically assess an LLM's instruction-following capabilities, with advanced service LLMs like GPT-4 acting as judges.

C.2 Implementation Details

All experiments were conducted on a server equipped with 2× NVIDIA RTX 4090 and 2× NVIDIA A6000 GPUs. Our implementation is built upon the efficient tuning frameworks of QLoRA and VB-LoRA to ensure memory-efficient training. However, due to the unique dual-branch architecture of `InLoRA`, updating all parameters with a single, uniform learning rate is sub-optimal. The discrete nature of the codebook and the continuous nature of the base matrices operate on entirely different gradient scales. Therefore, we employ a fine-grained, module-specific optimization strategy. Regarding the insertion points of our dynamic adapters, by default we apply them to 7 distinct linear modules within each transformer layer: the attention projections ($\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o$), as well as the feed-forward network projections ($\mathbf{W}_{up}, \mathbf{W}_{down}, \mathbf{W}_{gate}$). For the GLUE benchmark, we restrict the adapters to only the query and value projections ($\mathbf{W}_q, \mathbf{W}_v$) to achieve a better trade-off between performance and parameter efficiency.

NLU on RoBERTa. For baseline evaluations, the hyperparameters for full fine-tuning, LoRA, and VBLoRA are configured identically to those reported in the work (Li et al., 2024b), while the settings for GOAT follow the work (Fan et al., 2025). For a fair comparison, the adaptation rank is set to $r = 8$ for all baseline methods. In our method, we set the rank of the task-specific branch ($\mathbf{A}_0, \mathbf{B}_0$) to $r_0 = 4$ and the instance-customized dynamic branch ($\mathbf{A}_x, \mathbf{B}_x$) to $r_x = 8$. Furthermore, we assign a relatively higher learning rate of 5×10^{-3} to the codebook to encourage active exploration of the discrete priors, and 1×10^{-2} to the dynamic MLP projectors. Conversely, to maintain the stability of the global foundational knowledge, the learning rate for the shared base matrices ($\mathbf{A}_0, \mathbf{B}_0$) is restricted to a lower scale of 1×10^{-3}. In addition, a quantization loss weight of 1×10^{-4} is consistently applied to all tasks to prevent routing collapse. The complete set of hyperparameters used for the GLUE benchmark experiments is detailed in Table 8.

Table 9: **Hyperparameters for mathematical reasoning on Mistral-7B and Gemma-7B**.

Hyperparameter	LoRA	VB-LoRA	InLoRA (Ours)
# GPUs		1	
Optimizer		AdamW	
Warmup Ratio	0.03	0.02	0.02
Batch Size (per device)	1	1	1
Gradient Accumulation Steps	64	64	4
Epochs	2	2	2
LR Schedule	Cosine	Cosine	Cosine
Rank ($r/r_0/r_x$)	$r = 8$	$r = 4$	$r_0 = r_x = 4$
Codebook Size	-	2048	1024
Learning Rate (LoRA / Base A_0, B_0)	3E-4	-	1E-4
Learning Rate (Codebook/Vector bank)	-	4E-3	5E-4
Learning Rate (Logits / Projector)	-	4E-3	1E-3
Quantization Loss Wt. (γ)	-	-	1E-4

Mathematical Reasoning on Mistral-7B and Gemma-7B. To ensure a fair and rigorous comparison, our implementation of `InLoRA` is developed upon the official VB-LoRA codebase. The hyperparameter configurations for the baseline methods strictly follow the optimal settings reported in their original literatures (Li et al., 2024b). Specifically, regarding the adaptation rank, we set $r = 8$ for LoRA, whereas for both VB-LoRA and our `InLoRA`, the rank is set to 4. Across all experiments, models are optimized using the AdamW optimizer with a weight decay of 0. The learning rate follows a cosine annealing schedule, with a warmup ratio of 0.03 for LoRA and 0.02 for VB-LoRA and `InLoRA`. To enhance training efficiency and manage memory footprint, we employ BFloat16 (BF16) mixed-precision training and enable gradient checkpointing. For mathematical reasoning tasks, the instruction tuning is conducted on a subset of 100,000 samples from the training set, with the maximum sequence length truncated to 512. All experiments are conducted on NVIDIA RTX 4090 GPUs. A comprehensive summary of the detailed hyperparameter configurations is provided in Table 9.

Instruction Tuning on Llama-2-7B. For the instruction tuning experiments on the Cleaned Alpaca dataset using Llama-2-7B model, we maintain the foundational memory-efficient training framework (*e.g.*, 4-bit quantization, paged optimizers) while adapting the hyperparameter suite to suit the generative nature of the task. Additionally, the Mistral-7B model is trained on the WizardLM-Evol-Instruct dataset (Xu et al., 2024).

Distinct from the GLUE benchmark, we set the codebook capacity to 1024 and uniformly increase the dynamic rank to $r = 8$ for both VB-LoRA (our reproduced baseline) and `InLoRA` to accommodate the complex reasoning requirements of instruction following. For our optimization strategy, we assign a learning rate of 5×10^{-4} to the globally shared base matrices $(\mathbf{A}_0, \mathbf{B}_0)$. To ensure synchronized updates across the dynamic generation branches, the learning rates for the codebook, and the dynamic projectors are uniformly set to 2×10^{-4}. Furthermore, the quantization loss weight γ is adjusted to 5×10^{-4} to tightly bound the codebook vectors to the instruction semantics space.

Offline codebook initialization time is measured using 2,000 training instances randomly sampled from Cleaned Alpaca with a fixed seed of 42. The reported time includes Python startup, model loading, token-level feature extraction, feature aggregation, MiniBatch K-means clustering, codebook serialization, and temporary-file cleanup. This is a one-time offline preprocessing cost and is excluded from both training and inference time.

Table 10 summarizes the hyperparameter settings and measured computational costs. Compared with static LoRA-based methods, `InLoRA` requires additional training time and peak GPU memory because it generates instance-conditioned adapters dynamically. This dynamic generation also incurs inference overhead: unlike static LoRA adapters, the instance-conditioned updates cannot be merged into the frozen backbone weights and must be generated online for each input. Thus, although `InLoRA` is efficient in terms of trainable parameter count and adapter storage, this parameter

Table 10: **Hyperparameters, computing resources, and inference efficiency for instruction tuning experiments**. We report the experimental configurations of Llama-2-7B on the Cleaned Alpaca dataset and Mistral-7B on the WizardLM-Evol-Instruct dataset (Xu et al., 2024). Training time and peak training GPU memory are measured on our hardware setup. Inference efficiency is evaluated on all 80 MT-Bench questions using a single NVIDIA RTX A6000 with batch size 1 and a maximum generation length of 1,024 tokens. End-to-end inference time includes Python startup, model loading, and response generation. Inference throughput is calculated as the total number of generated response tokens divided by the end-to-end inference time. Peak inference GPU memory is measured by sampling the whole-GPU memory usage with `nvidia-smi`. "h" and "min" denote hours and minutes, respectively.

Hyperparameter	**Llama-2-7B (Cleaned Alpaca)**			**Mistral-7B (WizardLM)**	
	LoRA	**VB-LoRA**	**InLoRA (Ours)**	**PiSSA**	**InLoRA (Ours)**
# GPUs	1			2	
Optimizer	AdamW			AdamW	
Warmup Ratio	0.1			0.03	
Batch Size (per device)	4			4	
Gradient Accumulation Steps	4			16	
Epochs	1			1	
LR Schedule	Linear			Cosine	
Rank ($r/r_0/r_x$)	$r = 16$	$r = 8$	$r_0 = r_x = 8$	$r = 16$	$r_0 = r_x = 8$
Codebook Size	-	2048	1024	-	1024
Learning Rate (LoRA / Base A_0, B_0)	4E-4	-	5E-4	2E-5	2E-5
Learning Rate (Codebook /Vector Bank)	-	1E-3	2E-4	-	1E-5
Learning Rate (Logits / Projector)	-	1E-2	2E-4	-	1E-5
Quantization Loss Wt. (γ)	-	-	5E-4	-	5E-5
Training Time (7B)	2.0h	2.5h	$\sim$8.4h	2.5h	6.2h
Training GPU Memory (MiB)	8,467	6,872	35,065	22,520	32,659
End-to-End Inference Time (80 MT-Bench Questions, min)	30.94	33.20	78.50	37.24	106.57
Average Inference Latency (s/question)	23.20	24.90	58.87	27.93	79.93
Inference Throughput (tokens/s)	17.08	18.09	3.79	38.41	5.23
Peak Inference GPU Memory (MiB)	19,503	17,959	28,276	15,045	29,960
Offline Codebook Initialization Time (min/task)	–	–	6.30	–	6.80

efficiency comes at the cost of increased inference latency and peak GPU memory under our current implementation.

Dimension-Aware Sharing and Selective Routing. Scaling dynamic adapters to billion-parameter LLMs (*e.g.*, Llama-2-7B and Mistral-7B) presents a severe risk of parameter explosion and routing instability. To maintain extreme parameter efficiency while avoiding optimization collapse, **InLoRA** introduces a dimension-aware sharing and routing mechanism.

First, instead of allocating independent MLP projectors for each module, we group modules based on their input/output dimensionalities. For instance, the query, key, and value projections (W_q, W_k, W_v) inherently operate on the same latent semantic manifold of the input token. By forcing these dimension-aligned modules to share a single MLP projector, we not only drastically reduce the parameter overhead but also provide a structural regularization that encourages the projector to learn robust, generalized mapping functions rather than overfitting to specific attention heads.

Second, for parameter efficiency, we enforce dimension-selective codebook routing in our implementation. The discrete codebook is explicitly anchored to the primary hidden dimension d of the LLM (*e.g.*, the input dimension of $\mathbf{W}_q$). Although separate codebooks could be maintained for different input dimensions, doing so would increase parameter overhead and compromise our global sharing paradigm. Consequently, for modules with unaligned input dimensions, such as $\mathbf{W}_{\text{down}}$ in the feed-forward network, **InLoRA** bypasses discrete nearest-neighbor matching and directly projects the continuous features. This dimension-selective fallback ensures that the codebook remains a high-density knowledge reservoir for the core semantic space while maintaining parameter efficiency and routing stability.

Commonsense Reasoning on Llama-2-7B and Llama-3.1-8B-Instruct. For the commonsense reasoning experiments, we build our implementation upon the GOAT-PEFT framework. With Llama-2-7B, we compare **InLoRA** against LoRA, VB-LoRA, and GOAT using the same backbone and target-module scope. Following the principal settings of GOAT, we apply **InLoRA** to all seven

Table 11: **Hyperparameter settings for the commonsense reasoning experiments.** We report the configurations of LoRA, VB-LoRA, GOAT, and `InLoRA` with Llama-2-7B, and that of `InLoRA` with Llama-3.1-8B-Instruct. The Llama-3.1-8B configuration uses rank 16 by default; rank 8 is additionally evaluated for comparison with T2L under the same target-module scope. "–" denotes not applicable.

Hyperparameter	Llama-2-7B				Llama-3.1-8B-Instruct
	LoRA	VB-LoRA	GOAT	InLoRA (Ours)	InLoRA (Ours)
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW
Batch size (per device)	2	2	2	2	2
Effective batch size	16	16	16	16	16
Training epochs	2	2	2	2	2
Warmup ratio	0.03	0.03	0.03	0.03	0.03
Dropout	0.05	0.05	0.05	0.05	0.05
Target modules	7 linear modules	7 linear modules	7 linear modules	7 linear modules	$\mathbf{W}_q, \mathbf{W}_v$
Rank ($r/r_0/r_x$)	$r = 32$	$r = 16$	$r = 16$	$r_0 = r_x = 16$	$r_0 = r_x = 16$ (default); 8 (T2L)
Learning rate (overall / base $\mathbf{A}_0, \mathbf{B}_0$)	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4} / 5×10^{-4}	1×10^{-4} / 2×10^{-4}
Learning rate (projector)	–	–	–	5×10^{-4}	1×10^{-4}
Learning Rate (Codebook /Vector Bank)	–	2×10^{-2}	–	1×10^{-4}	1×10^{-4}
Quantization-loss weight γ	–	–	–	1×10^{-4}	1×10^{-4}

linear modules in each transformer layer, including the query, key, value, and output projections ($\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o$) and the up, down, and gate projections ($\mathbf{W}_{\text{up}}, \mathbf{W}_{\text{down}}, \mathbf{W}_{\text{gate}}$). All methods are trained for two epochs. LoRA uses rank $r = 32$, whereas VB-LoRA and GOAT use rank $r = 16$, and `InLoRA` uses $r_0 = r_x = 16$. All methods use a per-device batch size of 2 and an effective batch size of 16.

For Llama-3.1-8B-Instruct, we apply `InLoRA` only to the query and value projection modules, following the target-module scope used by T2L. We train for two epochs with a micro-batch size of 2 and eight gradient-accumulation steps, resulting in an effective batch size of 16. The default configuration uses rank 16, while rank 8 is additionally evaluated for comparison with T2L under the same target-module scope. The maximum sequence length is set to 256.

As summarized in Table 11, we employ component-specific learning rates for the dual-branch architecture of `InLoRA`. With Llama-2-7B, the task-specific branch and instance projectors use a learning rate of 5×10^{-4}, whereas the codebook uses 1×10^{-4}. With Llama-3.1-8B-Instruct, the overall learning rate is 1×10^{-4}; the task-specific branch uses 2×10^{-4}, while the projectors, codebook, and coefficient parameters use 1×10^{-4}. For both backbones, we use a codebook of $J = 512$ vectors and set the quantization-loss weight γ to 1×10^{-4}.

D Proof of Theorem 3.1

Let $\mathbf{M}_l^{(n)} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ denote the ideal update matrix for the n-th input instance $x^{(n)}$, where $n \in \{1, \ldots, N\}$, at layer $l \in \{1, \ldots, L\}$. Let $R_{n,l} = \text{rank}(\mathbf{M}_l^{(n)})$, and denote its singular values by $\sigma_1^{(n,l)} \geq \cdots \geq \sigma_{R_{n,l}}^{(n,l)} > 0$. Vanilla LoRA uses a dataset-shared update $\mathbf{S}_l = \mathbf{B}_l \mathbf{A}_l$ at layer l. In contrast, `InLoRA` combines a task-specific update $\hat{\mathbf{S}}_l = \hat{\mathbf{B}}_l \hat{\mathbf{A}}_l$ with an input-dependent update generated for $x^{(n)}$. For notational convenience, we denote the latter by

$$\mathbf{D}_{\boldsymbol{\theta},l}^{(n)} := \mathbf{D}_{\boldsymbol{\theta},l}\Big(x^{(n)}\Big) = \mathbf{B}_{x,l}^{(n)} \mathbf{A}_{x,l}^{(n)}, \tag{7}$$

where $\text{rank}(\hat{\mathbf{S}}_l) \leq r_0$, $\text{rank}(\mathbf{D}_{\boldsymbol{\theta},l}^{(n)}) \leq r_x$, and $\boldsymbol{\theta} \in \Theta$ denotes a feasible parameterization of the constrained codebook-and-projector generator. Thus, $\mathbf{S}_l$ refers exclusively to the vanilla-LoRA baseline, whereas $\hat{\mathbf{S}}_l$ and $\mathbf{D}_{\boldsymbol{\theta},l}^{(n)}$ denote the task-specific and instance-customized branches of `InLoRA`, respectively.

For any rank q, the optimal total approximation error of vanilla LoRA is

$$E_{\text{LoRA}}^*(q) := \min_{\substack{\mathbf{S}_1, \ldots, \mathbf{S}_L \\ \text{rank}(\mathbf{S}_l) \leq q,\ \forall l}} \sum_{n=1}^{N} \sum_{l=1}^{L} \left\| \mathbf{M}_l^{(n)} - \mathbf{S}_l \right\|_F^2. \tag{8}$$

The optimization over $\mathbf{S}_l$ is outside the sum over samples because the same update is applied to every input at layer l.

The optimal total approximation error over the realizable `InLoRA` family is

$$E^*_{\texttt{InLoRA}} := \inf_{\substack{\hat{\mathbf{S}}_1,\dots,\hat{\mathbf{S}}_L,\,\boldsymbol{\theta}\in\Theta \\ \mathrm{rank}(\hat{\mathbf{S}}_l)\le r_0,\ \forall l}} \sum_{n=1}^{N}\sum_{l=1}^{L} \left\| \mathbf{M}_l^{(n)} - \hat{\mathbf{S}}_l - \mathbf{D}_{\boldsymbol{\theta},l}^{(n)} \right\|_F^2 . \tag{9}$$

Because $r_0 \le r$, every rank-r_0 LoRA solution is also feasible for rank-r LoRA. We therefore define the approximation penalty caused by reducing the task-specific rank from r to r_0 as

$$\Delta_{r_0,r} := E^*_{\mathrm{LoRA}}(r_0) - E^*_{\mathrm{LoRA}}(r) \ge 0. \tag{10}$$

Fix an optimal rank-r_0 vanilla-LoRA solution $\{\mathbf{S}_l^{*(r_0)}\}_{l=1}^{L}$ to Eq. (8), and define its residual for sample n at layer l as

$$\mathbf{R}_l^{(n)} := \mathbf{M}_l^{(n)} - \mathbf{S}_l^{*(r_0)}. \tag{11}$$

The supremal reduction of this residual error attainable by the constrained instance-customized generator is defined as

$$G_{\mathrm{gen}} := \sum_{n=1}^{N}\sum_{l=1}^{L} \left\| \mathbf{R}_l^{(n)} \right\|_F^2 - \inf_{\boldsymbol{\theta}\in\Theta} \sum_{n=1}^{N}\sum_{l=1}^{L} \left\| \mathbf{R}_l^{(n)} - \mathbf{D}_{\boldsymbol{\theta},l}^{(n)} \right\|_F^2 . \tag{12}$$

We assume that the instance-customized generator admits a zero-output parameterization: there exists $\boldsymbol{\theta}_0 \in \Theta$ such that

$$\mathbf{D}_{\boldsymbol{\theta}_0,l}^{(n)} = \mathbf{0}, \qquad \forall n, l. \tag{13}$$

Because $\boldsymbol{\theta}_0$ is feasible, the infimum in Eq. (12) cannot exceed the residual error obtained by setting the dynamic update to zero. Consequently,

$$G_{\mathrm{gen}} \ge 0. \tag{14}$$

Lemma D.1 (Eckart–Young–Mirsky)**.** *For any matrix $\mathbf{M}$ with singular values $\sigma_1 \ge \cdots \ge \sigma_R > 0$, the minimum squared Frobenius error over all matrices of rank at most d is*

$$\min_{\mathrm{rank}(\hat{\mathbf{M}})\le d} \left\| \mathbf{M} - \hat{\mathbf{M}} \right\|_F^2 = \sum_{i=d+1}^{R} \sigma_i^2, \tag{15}$$

where the sum is defined as zero when $d \ge R$.

Let $d = r_0 + r_x$. By Lemma D.1, the unconstrained per-instance rank-d oracle error is

$$E_{\mathrm{oracle}}(d) = \sum_{n=1}^{N}\sum_{l=1}^{L}\sum_{i=d+1}^{R_{n,l}} \left(\sigma_i^{(n,l)}\right)^2 . \tag{16}$$

Theorem 3.1 (restated). Assume that $r_0 \le r$, that $\mathrm{rank}(\hat{\mathbf{S}}_l) \le r_0, \mathrm{rank}\left(\mathbf{D}_{\boldsymbol{\theta},l}^{(n)}\right) \le r_x$ for every n, l, and feasible $\boldsymbol{\theta}$, and that there exists $\boldsymbol{\theta}_0$ such that $\mathbf{D}_{\boldsymbol{\theta}_0,l}^{(n)} = \mathbf{0}$ for all n, l. Then, with $d = r_0 + r_x$,

$$E_{\mathrm{oracle}}(d) \le E^*_{\texttt{InLoRA}} \le E^*_{\mathrm{LoRA}}(r) + \Delta_{r_0,r} - G_{\mathrm{gen}}. \tag{17}$$

Consequently, `InLoRA` achieves a strictly smaller approximation error than rank-r LoRA whenever $G_{\mathrm{gen}} > \Delta_{r_0,r}$.

Proof. For every feasible `InLoRA` parameterization and every sample-layer pair (n, l), rank subadditivity gives

$$\mathrm{rank}\left(\hat{\mathbf{S}}_l + \mathbf{D}_{\boldsymbol{\theta},l}^{(n)}\right) \le \mathrm{rank}\left(\hat{\mathbf{S}}_l\right) + \mathrm{rank}\left(\mathbf{D}_{\boldsymbol{\theta},l}^{(n)}\right) \le r_0 + r_x = d. \tag{18}$$

The rank-d oracle can independently select any rank-d matrix for each sample-layer pair. In contrast, the updates of `InLoRA` must be jointly realizable by its task-specific branch and constrained instance-customized generator. The oracle feasible set therefore contains the realizable `InLoRA` feasible set. Applying Lemma D.1 independently to every $\mathbf{M}_l^{(n)}$ gives

$$E_{\text{oracle}}(d) \le E^*_{\texttt{InLoRA}}. \tag{19}$$

To establish the upper bound, choose the optimal rank-r_0 vanilla-LoRA solution as the task-specific branch of `InLoRA`:

$$\hat{\mathbf{S}}_l = \mathbf{S}_l^{*(r_0)}, \qquad l = 1, \dots, L. \tag{20}$$

For any feasible $\boldsymbol{\theta} \in \Theta$, the approximation error of this `InLoRA` parameterization is

$$\sum_{n=1}^{N}\sum_{l=1}^{L} \left\| \mathbf{M}_l^{(n)} - \mathbf{S}_l^{*(r_0)} - \mathbf{D}_{\boldsymbol{\theta},l}^{(n)} \right\|_F^2 = \sum_{n=1}^{N}\sum_{l=1}^{L} \left\| \mathbf{R}_l^{(n)} - \mathbf{D}_{\boldsymbol{\theta},l}^{(n)} \right\|_F^2. \tag{21}$$

Since $\{\mathbf{S}_l^{*(r_0)}\}_{l=1}^{L}$ is an optimal rank-r_0 vanilla-LoRA solution,

$$E^*_{\text{LoRA}}(r_0) = \sum_{n=1}^{N}\sum_{l=1}^{L} \left\| \mathbf{R}_l^{(n)} \right\|_F^2. \tag{22}$$

It follows directly from the definition of G_{gen} in Eq. (12) that

$$\inf_{\boldsymbol{\theta}\in\Theta} \sum_{n=1}^{N}\sum_{l=1}^{L} \left\| \mathbf{R}_l^{(n)} - \mathbf{D}_{\boldsymbol{\theta},l}^{(n)} \right\|_F^2 = E^*_{\text{LoRA}}(r_0) - G_{\text{gen}}. \tag{23}$$

The parameterizations obtained by fixing the task-specific branch as in Eq. (20) form a subset of the full feasible set of `InLoRA`. Taking the infimum over the full feasible set therefore gives

$$E^*_{\texttt{InLoRA}} \le E^*_{\text{LoRA}}(r_0) - G_{\text{gen}}. \tag{24}$$

Using the definition of $\Delta_{r_0,r}$,

$$E^*_{\text{LoRA}}(r_0) = E^*_{\text{LoRA}}(r) + \Delta_{r_0,r},$$

we obtain

$$E^*_{\texttt{InLoRA}} \le E^*_{\text{LoRA}}(r) + \Delta_{r_0,r} - G_{\text{gen}}. \tag{25}$$

Combining Eqs. (19) and (25) completes the proof. □

For completeness, the zero-output setting required by Eq. (13) can be realized in the projector parameterization used by `InLoRA` by setting the final linear layer and output bias of the projector that generates $\mathbf{A}_{x,l}^{(n)}$ to zero. This gives $\mathbf{A}_{x,l}^{(n)} = \mathbf{0}$ and therefore

$$\mathbf{D}_{\boldsymbol{\theta}_0,l}^{(n)} = \mathbf{B}_{x,l}^{(n)} \mathbf{A}_{x,l}^{(n)} = \mathbf{0}$$

for every sample and layer.

Theorem 3.1 separates the approximation penalty $\Delta_{r_0,r}$ caused by reducing the rank of the task-specific branch from the residual-error reduction G_{gen} attainable by the constrained instance-customized generator. If $G_{\text{gen}} > \Delta_{r_0,r}$, then

$$E^*_{\texttt{InLoRA}} < E^*_{\text{LoRA}}(r).$$

If $G_{\text{gen}} = \Delta_{r_0,r}$, then

$$E^*_{\texttt{InLoRA}} \le E^*_{\text{LoRA}}(r).$$

If $G_{\text{gen}} < \Delta_{r_0,r}$, the theorem does not determine which method has the smaller optimal approximation error.

Finally, when $r_0 = r$, we have $\Delta_{r_0,r} = 0$, and therefore

$$E^*_{\texttt{InLoRA}} \le E^*_{\text{LoRA}}(r) - G_{\text{gen}} \le E^*_{\text{LoRA}}(r).$$

Thus, the original LoRA-containment result is recovered as a special case, with strict improvement whenever $G_{\text{gen}} > 0$. This is an approximation-capacity result rather than a guarantee of optimization, generalization, or downstream performance.

Table 12: **Results of RoBERTa-base with different fine-tuning methods on six datasets of the GLUE benchmark**. We report Matthews correlation coefficient (MCC) for CoLA, Pearson correlation coefficient (PCC) for STS-B, and accuracy (Acc.) for all remaining tasks. For `InLoRA`, we set $r_0 = 4$, and $r_x = 8$; for all comparison methods, the rank is set to $r = 8$. "*" indicates results reported in prior work (Li et al., 2024b). The best results are highlighted in **bold**.

Method	# Trainable Parameters	SST-2 (Acc.)	MRPC (Acc.)	CoLA (Mcc.)	QNLI (Acc.)	RTE (Acc.)	STS-B (Pcc.)	All Avg.
FFT*	125.0M	94.8	90.2	63.6	92.8	78.7	91.2	85.2
FFT	125.0M	94.3	89.0	62.9	92.9	78.0	90.8	84.7
LoRA	0.295M	94.2	88.7	62.3	92.5	78.7	90.7	84.5
VB-LoRA	0.023M	93.6	89.2	60.8	92.2	78.0	90.6	84.1
`InLoRA` ($J = 90$)	0.234M	**94.4**	88.7	62.6	92.8	**80.5**	90.7	**85.0**
`InLoRA` ($J = 270$)	0.372M	94.3	**89.2**	62.3	92.6	**80.5**	90.7	84.9
`InLoRA` ($J = 540$)	0.579M	94.0	88.7	**62.8**	**93.0**	79.8	90.8	84.9
`InLoRA` ($J = 1024$)	0.951M	94.0	89.0	61.6	92.9	79.4	**91.0**	84.7

E More Experimental Results

E.1 Performance on RoBERTa-base

Table 12 reports the results of various adaptation methods on RoBERTa-base. Our `InLoRA` outperforms LoRA and VB-LoRA across most tasks. Specifically, $J = 90$ achieves the highest average score of 85.0, outperforming LoRA and VB-LoRA by 0.5 and 0.9 points, respectively. Increasing the codebook size provides no consistent improvement, suggesting that a compact codebook is sufficient for these tasks. Notably, this configuration outperforms VB-LoRA by 2.5 percentage points on RTE (80.5% vs. 78.0%) and 1.8 percentage points on CoLA (62.6% vs. 60.8%). Our most efficient variant, `InLoRA` with $J = 90$, uses only 0.234M trainable parameters, fewer than LoRA's 0.295M. These findings echo those from the large-model experiments: instance-customized dynamic composition enables both better predictive performance and greater parameter efficiency than static low–rank adaptation.

E.2 Additional Comparison with Hypernetwork-Based LoRA Generation

To further examine whether the improvements of `InLoRA` can be attributed merely to parameter generation or a larger adapter rank, we compare it with Text-to-LoRA (T2L) (Charakorn et al., 2025), a representative hypernetwork-based method that generates task-specific LoRA adapters from natural-language task descriptions. We consider T2L (SFT-L), the strongest T2L variant reported with Llama-3.1-8B-Instruct, together with the corresponding base-model, in-context-learning, task-description, and multi-task LoRA baselines reported in the same study. T2L uses rank-8 adapters applied to the query and value projection modules. For a more controlled comparison, we additionally evaluate `InLoRA` with rank 8 and the same target-module scope, alongside our rank-16 configuration. All averages in Table 13 are recomputed over exactly the same seven tasks.

As shown in Table 13, the rank-8 configuration of `InLoRA` achieves an average accuracy of 87.77%, outperforming T2L (SFT-L) by 8.41 percentage points. Under the same adapter rank and target-module scope, `InLoRA` performs better on five of the seven benchmarks, with particularly large gains on HellaSwag (95.8% versus 72.8%) and WinoGrande (88.4% versus 60.0%). Improvements are also observed on PIQA, ARC-e and OBQA, whereas T2L remains stronger on BoolQ.

Importantly, the T2L-L hypernetwork contains 55M trainable parameters (Charakorn et al., 2025), whereas the rank-8 configuration of `InLoRA` uses only 5.8M trainable parameters, approximately $9.5\times$ fewer. Increasing the rank of `InLoRA` from 8 to 16 raises its trainable parameter count to 9.3M but yields only a modest improvement in average accuracy, from 87.77% to 88.01%. The rank-16 configuration still uses approximately $5.9\times$ fewer trainable parameters than T2L-L while outperforming it by 8.65 percentage points on average. These results suggest that the gains of `InLoRA` cannot be attributed solely to a larger adapter rank or trainable parameter budget, and instead support the effectiveness and parameter efficiency of its instance-conditioned adaptation mechanism.

Table 13: Comparison with the hypernetwork-based Text-to-LoRA baseline on seven overlapping commonsense reasoning benchmarks using Llama-3.1-8B-Instruct. The results of T2L and its associated baselines are taken from Charakorn et al. (2025). The average is recomputed over the seven tasks shown in the table. Both T2L and `InLoRA` target the query and value projection modules. We report `InLoRA` with configured ranks 8 and 16. The best result in each column is highlighted in bold.

Method	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
Llama-3.1-8B-Instruct	80.4	79.8	66.6	55.3	90.6	73.3	75.4	74.49
3-shot ICL	80.0	80.9	59.3	61.3	91.9	80.7	77.6	75.96
Prepending task description	79.9	81.7	69.8	62.4	92.5	80.2	78.4	77.84
Multi-task LoRA	83.3	83.8	70.8	60.3	92.8	82.0	81.8	79.26
T2L (SFT-L)	**84.4**	81.2	72.8	60.0	92.9	82.4	81.8	79.36
`InLoRA` **(Ours, rank 8)**	76.2	90.0	95.8	**88.4**	**93.9**	82.3	87.8	87.77
`InLoRA` **(Ours, rank 16)**	76.1	**90.3**	**96.1**	88.0	**93.9**	**83.5**	**88.2**	**88.01**

Table 14: **Ablation results of different codebook sizes** for RoBERTa-large on the GLUE benchmark. Results report the median over 5 random seeds to ensure statistical reliability.

Model (Method)	# Params	SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Avg.
InLoRA ($J = 90$)	0.504M	95.9	90.0	**66.8**	94.6	85.2	91.8	87.4
InLoRA ($J = 270$)	0.688M	**96.1**	**90.2**	**66.8**	**94.7**	85.2	**92.0**	**87.5**
InLoRA ($J = 540$)	0.965M	96.0	89.2	**66.8**	94.5	85.6	91.8	87.3
InLoRA ($J = 1024$)	1.460M	96.0	90.0	66.3	94.6	**86.4**	91.8	**87.5**

Nevertheless, the performance differences remain task dependent, as evidenced by the lower accuracy of `InLoRA` on BoolQ.

E.3 Ablation Study

In this section, we investigate the impact of the codebook size (*i.e.*, J) on the performance of our `InLoRA` across different architectures and tasks. It is important to note that while larger codebook sizes can occasionally yield marginal performance gains, they also increase the number of trainable parameters. As demonstrated in our main results, `InLoRA` with $J = 270$ achieves superior performance compared to vanilla LoRA while maintaining a strictly smaller parameter footprint. Therefore, we report `InLoRA` with $J = 270$ as our primary configuration in the main paper to highlight its optimal trade-off between parameter efficiency and task performance. Here, we present the comprehensive ablation results for larger J values.

Impact on NLU Tasks. We first evaluate the codebook size $J \in \{90, 270, 540, 1024\}$ on the GLUE benchmark using the RoBERTa-large model. To ensure the reliability of our findings and mitigate the impact of random initialization, we run each configuration across 5 different random seeds and report the median performance.

As shown in Table 14, increasing the codebook size from 90 to 270 slightly improves the average score from 87.4 to 87.5 and yields the best results on MRPC, QNLI, and STS-B. However, further increasing J to 540 or 1024 does not yield significant continuous improvements, and performance even fluctuates on certain tasks (*e.g.*, dropping to 89.2 on MRPC for $J = 540$). This indicates that for NLU tasks with encoder-only architectures, a relatively compact codebook (*e.g.*, $J = 90$ or 270) provides sufficient capacity to capture the necessary dynamic adaptations, whereas excessively large codebooks might introduce redundancy.

Impact on Generative Tasks. We extend our ablation study to decoder-only Large Language Models (LLMs), including Llama-2-7B, Gemma-7B, and Mistral-7B, across complex generative tasks such as Mathematical Reasoning, Instruction Tuning, and Commonsense Reasoning. We compare two larger codebook configurations: $J = 512$ and $J = 1024$, given the inherently larger hidden dimensions of these LLMs.

Table 15: **Ablation study of codebook sizes ($J \in \{512, 1024\}$) on three generative tasks across different LLMs**. The optimal size varies depending on the task difficulty and model architecture.

Task Category	Base Model	Dataset	InLoRA ($J = 512$)	InLoRA ($J = 1024$)
Mathematical Reasoning	Gemma-7B	GSM8K	**68.69**	68.54
		MATH	25.30	**27.58**
	Mistral-7B	GSM8K	61.94	**63.68**
		MATH	20.32	**21.38**
Instruction Tuning	Llama-2-7B	MT-Bench	4.82	**4.99**
	Mistral-7B	MT-Bench	6.88	**7.18**
Commonsense Reasoning	Llama-2-7B	ARC-c	**71.59**	65.53
		ARC-e	**87.33**	85.56
		Boolq	74.29	**74.53**
		PIQA	**86.45**	85.29
		SIQA	**82.24**	81.60
		OBQA	**85.00**	82.80
		HellaSwag	90.95	**91.20**
		WinoGrande	**86.35**	86.11

The results are summarized in Table 15. Interestingly, we observe that **the optimal codebook size is highly dependent on both the base model architecture and the specific task complexity**. For instance, in mathematical reasoning, Gemma-7B performs better with $J = 512$ on GSM8K (68.69) but benefits from a larger capacity $J = 1024$ on the more challenging MATH dataset (27.58 *vs*. 25.30). Conversely, Mistral-7B consistently prefers the larger codebook ($J = 1024$) across both mathematical tasks. Similar task-model specific preferences are observed in instruction tuning and commonsense reasoning. These findings suggest that scaling the codebook size can unlock additional reasoning capabilities for specific LLMs on harder datasets, though it is not a strictly monotonic relationship across all scenarios.

Table 16: **Ablation results of `InLoRA` using Llama-2-7B across two commonsense reasoning datasets.** `InLoRA` has a task-specific branch $(\mathbf{A}_0, \mathbf{B}_0)$ and an instance-customized branch $(\mathbf{A}_x, \mathbf{B}_x)$, learns a codebook of size $J = 512$, and uses $K = 2$ in TKAM by default. "No" removes the codebook and directly feeds the continuous instance features to the projectors. "Fixed" freezes the initialized codebook throughout training, while "Random" replaces K-means initialization with random vectors but keeps the codebook trainable. # Params denotes the percentage of trainable parameters relative to the total model parameters.

Variant	$(\mathbf{A}_0, \mathbf{B}_0)$	Codebook C	Top-K	# Params	ARC-e	ARC-c
`InLoRA` (default)	✓	Learn, $J = 512$	2	0.63%	**87.33**	**71.59**
No shared branch	×	Learn, $J = 512$	2	0.04%	83.63	61.18
Codebook property	✓	No, $J = 0$	–	0.60%	85.41	66.38
	✓	Fixed, $J = 512$	2	0.60%	85.35	65.70
	✓	Random, $J = 512$	2	0.63%	85.27	65.19
Codebook size	✓	Learn, $J = 1024$	2	0.66%	85.56	65.53
K in Top-K	✓	Learn, $J = 512$	1	0.63%	85.40	64.16
	✓	Learn, $J = 512$	4	0.63%	85.61	63.91

Impact on Commonsense Reasoning Tasks. As shown in Table 16, the default configuration consistently achieves the best performance on both datasets. Removing the shared branch reduces accuracy by 3.70 points on ARC-e and 10.41 points on ARC-c, indicating that task-level shared knowledge complements instance-specific adaptation. Removing, freezing, or randomly initializing the codebook also causes consistent degradation, particularly on ARC-c, supporting the use of a trainable codebook with K-means initialization. Moreover, increasing the codebook size to $J = 1024$ or changing the number of retrieved prototypes from $K = 2$ to $K = 1$ or $K = 4$ provides

no further improvement. These results support the effectiveness of the shared branch, the learned codebook, and the default $J = 512$, $K = 2$ configuration for commonsense reasoning.

F LIMITATIONS AND BROADER IMPACTS.

Our `InLoRA` reduces trainable parameters and makes LLM adaptation more accessible. However, it introduces two trade-offs: (1) the parameter count is slightly larger than extreme-compression methods (*e.g.*, VB-LoRA); (2) due to dynamic generation of instance-customized adapters, inference incurs additional latency compared to vanilla LoRA which can be merged with base weights. Future work could use custom CUDA kernels or knowledge distillation to mitigate this issue. We foresee no negative societal impacts beyond those general to LLMs, *e.g.*, generating misleading content.