Reusable Experiences: Latent Routing and Modular Composition in LLMs

Shuai Ling[1,2*] **Lizi Liao**[3] **Dongmei Jiang**[2†] **Weili Guan**[1†]

[1]Harbin Institute of Technology (Shenzhen) [2]Pengcheng Laboratory
[3]Singapore Management University

24b952037@stu.hit.edu.cn lzliao@smu.edu.sg
jiangdm@pcl.ac.cn honeyguan@gmail.com

Abstract

Large language models (LLMs) have remarkable capabilities, but adapting them to specialized domains poses a fundamental question: *how should accumulated experience be represented and leveraged?* Existing approaches represent experience either as explicit textual artifacts in prompts (*e.g.*, retrieved documents or dialogues) or implicitly within model weights via fine-tuning (*e.g.*, LoRA adapters). However, textual methods are limited by context windows and cannot internalize knowledge, while parametric fine-tuning yields one adapter per task with minimal cross-task skill reuse. We propose **ReX** (**Re**usable e**X**perience), an experience-centric adaptation framework that treats latent experiences — recurring reasoning patterns and skills — as fundamental units for LLM specialization. Our method learns a shared Experience Bank of foundational skill vectors and uses a VAE-based encoder to map each input to a low-dimensional experience code. An Experience Router then dynamically composes the relevant skill vectors from this bank into a lightweight adapter for that input. By reusing skills across inputs, **ReX** enables implicit knowledge sharing across tasks without any explicit task identifiers. Experiments on multi-task NLP benchmarks show that this approach outperforms standard task-specific fine-tuning, yielding improved generalization through flexible skill reuse. Code is available at https://github.com/iLearn-Lab/ACL26-ReX.

1 Introduction

Large language models (LLMs) have demonstrated impressive general capabilities across reasoning, understanding, and generation (OpenAI et al., 2024; DeepSeek-AI et al., 2025; Huang and Chang, 2023; Li et al., 2024b; Matarazzo and Torlone, 2025; Zhao et al., 2025a). Yet adapting them to specialized domains centers on two challenges: representing accumulated experiences and leveraging these experiences. Existing approaches fall into two primary paradigms. Textual experience encodes knowledge explicitly within prompts (Brown et al., 2020a)—such as retrieved documents (Lewis et al., 2020), conversation histories (Pink et al., 2025), or reflective summaries (Shinn et al., 2023; Madaan et al., 2023)—while parametric experience internalizes it implicitly within model weights through fine-tuning (Howard and Ruder, 2018), often via parameter-efficient methods (Wang et al., 2025) such as LoRA (Hu et al., 2021).

However, these paradigms exhibit complementary limitations. Textual methods are constrained by limited context windows and require repeated retrieval, preventing the durable internalization of knowledge (Wang et al., 2024a). Parametric fine-tuning, though effective at encapsulating experience, typically produces one adapter per task, offering limited opportunities for skill reuse or cross-task generalization (Caccia et al., 2023).

These problems stem from the common assumption that adaptation should be organized around human-defined tasks rather than the intrinsic structure of experience itself. While recent work has explored adapter composition (Huang et al., 2024), mixture-of-experts routing (Li et al., 2024a; Zhang et al., 2024a; Tian et al., 2025), and fusion-based mechanisms to improve flexibility (Wu et al., 2024b), these approaches still operate at the task level—combining or routing between task-specific adapters—and may even depend on explicit task identifiers or textual descriptions.

What remains missing is a unified framework that treats experience—not tasks—as the fundamental unit of adaptation, enabling models to discover, encode, and compose reusable skills directly from data. We argue that effective adaptation should instead be experience-centric (Silver and Sutton, 2025)—grounded in the modular represen-

[*]Work was done during an internship at SMU.
[†]Corresponding authors

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 30087–30100

tation and reuse of latent experiences, namely recurring reasoning patterns or skills that emerge across tasks. Such latent experiences transcend surface-level task boundaries, enabling models to compose relevant skills on demand rather than retraining for each domain or dataset.

To realize this idea, we propose **ReX**, an experience-centric adaptation framework that operationalizes learning from latent experiences. ReX constructs a shared Experience Bank—a parametric memory comprising foundational skill vectors—and employs a VAE-based encoder to map each input instance into a low-dimensional experience code that reflects the underlying reasoning pattern. An Experience Router then interprets this code to assign relevance scores over the Experience Bank, dynamically selecting and combining the most pertinent skill vectors to synthesize lightweight Low-Rank Adaptation (LoRA) adapters. Both the A and B matrices of LoRA maintain their own experience banks, allowing fine-grained and flexible composition. This instance-level routing mechanism enables dynamic, data-driven adaptation, where inputs invoking similar reasoning patterns reuse the same parametric components, achieving implicit knowledge sharing across tasks without reliance on task identifiers or external metadata.

Our key contributions are threefold:1) We introduce a principled formulation of LLM adaptation that shifts the focus from task-centric to experience-centric. 2) We present ReX, which employs a VAE-based encoder, an Experience Bank, and an Experience Router to dynamically compose adapters. 3) Experiments on multi-task NLP benchmarks show that ReX improves generalization and skill reuse over task-specific fine-tuning.

2 Related Work

2.1 Experience in Textual Form

Textual encoding represents experience as explicit, retrievable artifacts external to model parameters. Memory-augmented agents such as Generative Agents maintain chronological logs of interactions and retrieve salient episodes based on recency, relevance, or importance (Park et al., 2023). Extensions like Voyager (Wang et al., 2023) and MemoryBank (Zhong et al., 2024) introduce long-term episodic stores or skill libraries, enabling reuse of prior strategies across sessions without retraining. Reflection-based frameworks such as Reflexion (Shinn et al., 2023), Self-Refine (Madaan et al., 2023), and CRITIC (Gou et al., 2024) store verbal self-critiques as experiential lessons to refine reasoning through iterative feedback. Retrieval-augmented methods (Lewis et al., 2020) similarly treat external corpora as non-parametric memory, fetching relevant knowledge at inference time. Hybrid reasoning architectures like ReAct (Yao et al., 2023) interleave reasoning traces with tool calls, combining retrieval and execution in unified loops.

While these methods enhance adaptability, their experiences remain textual: they must be retrieved or appended to prompts, confined by context length and latency, and are never truly internalized.

2.2 Experience in Parameter Form

Parametric encoding internalizes experience within model weights through fine-tuning (Howard and Ruder, 2018). Full fine-tuning updates all parameters but is computationally expensive and lacks modularity. Parameter-efficient fine-tuning (PEFT) mitigates these issues by training small subsets of parameters while keeping the base model frozen (Li and Liang, 2021; Hu et al., 2021; Houlsby et al., 2019). Among these, LoRA has become dominant, introducing trainable low-rank matrices alongside frozen weights (Hu et al., 2021). Its successors extend adaptability through quantization-aware variants (Dettmers et al., 2023), reparameterized decompositions (Liu et al., 2024), dynamic rank allocation (Zhang et al., 2023), and so on (Wu et al., 2024a; Zhao et al., 2025b).

However, most LoRA methods remain task-centric: one adapter per task or domain. Compositional extensions—such as Mixture-of-LoRA, LoRAHub (Huang et al., 2024), DLP-LoRA (Zhang and Li, 2025), and Text-to-LoRA (Charakorn et al., 2025)—begin to relax this rigidity through routing, fusion, or hypernetwork generation (Tian et al., 2025; Mahabadi et al., 2021). Nevertheless, these methods continue to operate at the task level—composing or routing among task-specific adapters—and often require explicit task identifiers or textual prompts to steer composition.

ReX bridges the gap between textual shallowness and parametric rigidity by treating experience, rather than tasks, as the unit of adaptation. It distills training instances into latent experience codes that capture transferable reasoning patterns and dynamically composes adapter subspaces from a shared parametric memory. This enables modular, context-specific reuse of skills without predefined task su-

pervision, integrating flexibility and internalization within a unified framework.

3 Problem Setup

The goal of this work is to explore how accumulated experience should be represented and leveraged. We focus on the parametric representation of experience and use a learnable Experience Bank that stores a set of basic skill atoms, each representing a reusable unit of experience. For each input instance x, the model retrieves and combines a subset of these atoms to form its instance-specific parametric experience, which is realized as the LoRA parameters for that input.

Definition (latent experience unit). A latent experience unit (skill atom) in ReX is a task-agnostic, reusable adaptation basis that captures a recurring reasoning pattern shared across inputs and is reused via instance-level composition. In this work, units are instantiated as vectors in a shared Experience Bank (a column of $\mathbf{U}$ or a row of $\mathbf{V}$) and are routed and combined to synthesize input-adaptive LoRA factors. More generally, ReX is *adapter-agnostic in principle*: the same routing-and-composition mechanism can be applied to other PEFT adapters by redefining the Experience Bank atoms to match the target adapter parameterization.

Formally, let $f(\cdot;\boldsymbol{\theta}_0)$ denote a pretrained model with frozen parameters. Following LoRA, low-rank updates are injected at selected sites $\mathcal{S}$:

$$\Delta\mathbf{W}^{(\ell)} = \alpha^{(\ell)}\mathbf{A}^{(\ell)}\mathbf{B}^{(\ell)}, \quad \ell \in \mathcal{S},$$

where ℓ is the layer index, $\alpha^{(\ell)} \in \mathbb{R}$ is a scalar, and $\mathbf{A}^{(\ell)} \in \mathbb{R}^{d\times r}$ and $\mathbf{B}^{(\ell)} \in \mathbb{R}^{r\times d}$ are rank-r matrices with $r \ll d$.

We define a function ϕ that maps each input x to its corresponding LoRA parameters by routing through the Experience Bank:

$$\phi : x \mapsto \Delta\boldsymbol{\theta}(x) = \{\Delta\mathbf{W}^{(\ell)}(x)\}_{\ell\in\mathcal{S}}.$$

This function ϕ represents the process of experience composition, selecting and combining relevant skill atoms to construct instance-adaptive adaptation parameters.

4 The ReX Approach

Given an input x, ReX uses experience-centric routing at each injection site to select skill vectors and synthesize instance-specific LoRA matrices. It has three components: a VAE that maps layer observations to a latent code (§4.1), an Experience Bank of skill atoms (§4.2), and an Experience Router that selects and mixes skills based on the latent code (§4.3). The resulting LoRA matrices are optimized with the overall training objective (§4.4, §4.5).

We use N for batch size, T for sequence length, d for hidden size, d_z for latent size, r for LoRA rank, and r_b for bank size; $\mathbf{B}$ always denotes the LoRA right factor. We omit layer indices ℓ and instantiate all modules per layer. Unless noted, we describe operations for a single sample and apply them independently across the batch.

4.1 Experience Code Extraction

We use the Variational Autoencoder (VAE) (Kingma and Welling, 2022) to compress layer statistics into compact latent codes. The frozen base model produces hidden states $\mathbf{H} \in \mathbb{R}^{N\times T\times d}$ at a given layer. We aggregate these via a pooling operator to obtain a layer observation at sequence level:

$$\mathbf{r} = \mathrm{Pool}(\mathbf{H}) \in \mathbb{R}^{N\times d}.$$

In this work, we use mean-pooling, but other pooling strategies are also acceptable.

Our latent encoder is designed to (i) retain sufficient layer-level information for routing, (ii) induce a shared and well-structured latent space across heterogeneous reasoning domains, and (iii) remain lightweight under PEFT constraints. A compact VAE provides a practical balance between structure and efficiency under these requirements. By comparison, contrastive encoders (Zhang et al., 2024b) typically require pair construction, which is less aligned with our mixed-domain setting where cross-domain instances may share atomic skills. Sparse autoencoders (Olson et al., 2025) also often incur substantially larger per-layer parameter budgets to be effective.

Given the pooled layer observation $\mathbf{r}$, the VAE encoder $\mathrm{Enc}(\cdot)$ (implemented as a multilayer perceptron) outputs the mean and log-variance:

$$\boldsymbol{\mu}, \log\boldsymbol{\sigma} = \mathrm{Enc}(\mathbf{r}), \quad \boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^{N\times d_z},$$

where d_z denotes the dimension of the latent code $\mathbf{z}$.

During training, we sample the latent code via reparameterization:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I}_{d_z}).$$

At inference, we use the deterministic mode $\mathbf{z} = \boldsymbol{\mu}$ to ensure reproducibility.

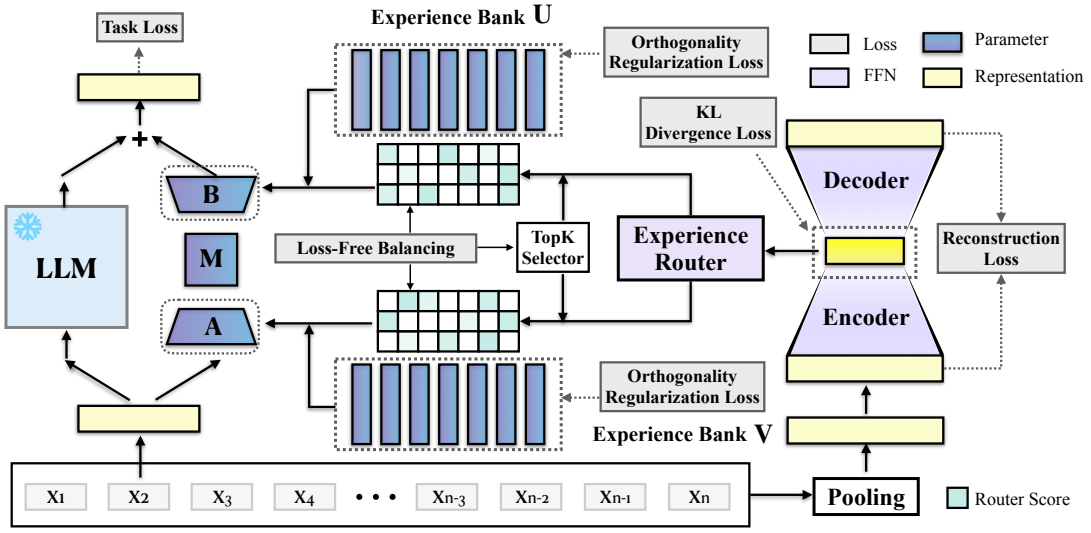

Figure 1: The ReX framework learns latent experiences through a VAE and dynamically composes instance-specific LoRA parameters via an experience router and shared experience banks, enabling reusable, task-agnostic adaptation.

A reconstruction decoder attempts to recover the original layer observation:

$$\hat{\mathbf{r}} = \mathrm{Dec}_{\mathrm{rec}}(\mathbf{z}).$$

The VAE is regularized by two losses. The KL divergence $\mathcal{L}_{\mathrm{kl}} = \mathrm{KL}(q(\mathbf{z}|\mathbf{r})\|\mathcal{N}(0,\mathbf{I}))$ regularizes the latent distribution and prevents posterior collapse, whereas the reconstruction loss $\mathcal{L}_{\mathrm{rec}} = \|\mathbf{r}-\hat{\mathbf{r}}\|_F^2$ ensures that the code retains layer-specific information.

4.2 Experience Bank

The Experience Bank maintains two sets of learnable skill atoms:

$$\mathbf{U} \in \mathbb{R}^{d\times r_b}, \quad \mathbf{V} \in \mathbb{R}^{r_b\times d},$$

where r_b is the bank size. Columns of $\mathbf{U}$ serve as building blocks for $\mathbf{A}(x)$, while rows of $\mathbf{V}$ form the basis for $\mathbf{B}(x)$. Both are the LoRA matrices constructed based on input x.

To improve identifiability and numerical stability, we encourage approximate orthogonality via the regularization loss:

$$\mathcal{L}_{\mathrm{orth}} = \|\mathbf{U}^\top\mathbf{U} - \mathbf{I}\|_F^2 + \|\mathbf{V}\mathbf{V}^\top - \mathbf{I}\|_F^2.$$

This orthogonality term is a *soft* regularizer: it reduces redundancy and encourages diverse, complementary primitive atoms without enforcing strict independence. Importantly, instance-specific skills are formed by top-k weighted compositions of atoms (Section 4.3 & Section 4.4), so overlapping or hierarchical skills can still emerge through reuse of partially shared subsets.

4.3 Experience Router

The Experience Router maps the latent code $\mathbf{z} \in \mathbb{R}^{d_z}$ to atom selection and weighting decisions. The following operations are performed independently for each sample.

Scoring. A feedforward network processes the latent code to produce scoring tensors:

$$\mathbf{S}_A, \mathbf{S}_B = \mathrm{Router}(\mathbf{z}), \quad \mathbf{S}_A, \mathbf{S}_B \in \mathbb{R}^{r\times r_b},$$

where the two dimensions index rank positions (r) and candidate atoms (r_b). Each entry $\mathbf{S}_A[j,i]$ represents the affinity between the j-th rank position and the i-th atom in $\mathbf{U}$; analogously for $\mathbf{S}_B$.

Selection. For each rank position j, we select top-k atoms based on bias-adjusted scores. For the left factor $\mathbf{A}$:

$$\mathbf{s}_A^{\mathrm{sel}}[j,:] = \mathbf{S}_A[j,:] + \boldsymbol{\beta}_A^\top,$$
$$\mathcal{S}_j^A = \mathrm{TopK}_i(\mathbf{s}_A^{\mathrm{sel}}[j,:], k),$$

where $\boldsymbol{\beta}_A \in \mathbb{R}^{r_b}$ is a non-trainable bias vector that balances atom usage. The construction of $\mathcal{S}_j^B$ from $\mathbf{S}_B$ and $\boldsymbol{\beta}_B$ follows the same procedure. Gradients do not propagate through the discrete top-k selections.

Auxiliary-Loss-Free Balancing. To prevent over-utilization of a small subset of atoms, inspired by (DeepSeek-AI et al., 2025; Zhao et al., 2025b), we adjust selection biases based on empirical usage statistics. Let $Z = N \cdot r \cdot k$ denote the total number

of selections per batch. For atom i:

$$\ell_i = \sum_{n=1}^{N} \sum_{j=1}^{r} \mathbb{1}(i \in \mathcal{S}_{n,j}),$$
$$q_i = \ell_i / Z, \quad u_i = 1/r_b,$$

where q_i represents empirical selection frequency. Given tolerance $\varepsilon \in (0, 1)$ and step size $\gamma > 0$, we update biases in the training stage:

$$\beta_i \leftarrow \begin{cases} \beta_i - \gamma, & \text{if } q_i > u_i(1+\varepsilon), \\ \beta_i + \gamma, & \text{if } q_i < u_i(1-\varepsilon), \\ \beta_i, & \text{otherwise.} \end{cases}$$

This doesn't update model parameters using the standard backpropagation algorithm: over-selected atoms have biases reduced, while under-utilized atoms receive increased biases, ensuring balanced skill utilization without conflicting loss terms.

Weighting. Within each selected subset, we compute normalized weights via softmax over only the selected atoms. For the left factor $\mathbf{A}$:

$$w^A_{j,i} = \frac{\exp(\mathbf{S}_A[j,i]/\tau_A)}{\sum_{i' \in \mathcal{S}^A_j} \exp(\mathbf{S}_A[j,i']/\tau_A)}, \quad \forall i \in \mathcal{S}^A_j,$$

where $\tau_A > 0$ is a temperature hyperparameter. Atoms not in $\mathcal{S}^A_j$ receive zero weight and are excluded from the synthesis. The weights $w^B_{j,i}$ for the right factor $\mathbf{B}$ are computed analogously with temperature τ_B.

4.4 LoRA Matrix Synthesis

Given selection sets $\mathcal{S}^A_j, \mathcal{S}^B_j$ and weights $w^A_{j,i}, w^B_{j,i}$ from the router, we construct instance-specific LoRA matrices through weighted combinations of selected skills.

For each rank position j, we synthesize the j-th column of $\mathbf{A}$ and j-th row of $\mathbf{B}$:

$$\mathbf{a}_{:,j} = \sum_{i \in \mathcal{S}^A_j} w^A_{j,i}\, \mathbf{u}_i, \quad \mathbf{b}_{j,:} = \sum_{i \in \mathcal{S}^B_j} w^B_{j,i}\, \mathbf{v}_i^\top,$$

where $\mathbf{u}_i$ denotes the i-th column of $\mathbf{U}$ and $\mathbf{v}_i$ the i-th row of $\mathbf{V}$. Assembling across all r rank positions yields complete LoRA matrices:

$$\mathbf{A} \in \mathbb{R}^{d \times r}, \quad \mathbf{B} \in \mathbb{R}^{r \times d}, \quad \Delta\mathbf{W} = \alpha \mathbf{AMB},$$

where $\mathbf{M} \in \mathbb{R}^{r \times r}$ is a learnable mixing matrix that scales and combines rank-wise contributions.

This construction enables each rank position to independently select and weight different skill vectors, providing fine-grained compositional flexibility while maintaining parameter efficiency through the shared banks $\mathbf{U}$ and $\mathbf{V}$.

4.5 Training Objective

The complete training objective combines supervised loss with regularization terms:

$$\mathcal{L} = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathrm{CE}(f(x; \boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}(x)), y) \right] + \sum_{\ell \in \mathcal{S}} \left(\lambda^{(\ell)}_{\text{kl}}\, \mathcal{L}^{(\ell)}_{\text{kl}} + \lambda^{(\ell)}_{\text{rec}}\, \mathcal{L}^{(\ell)}_{\text{rec}} + \lambda^{(\ell)}_{\text{orth}}\, \mathcal{L}^{(\ell)}_{\text{orth}} \right),$$

where CE denotes cross-entropy loss computed over target tokens. The cross-entropy term drives task-specific adaptation. The KL divergence prevents posterior collapse in the VAE. The reconstruction loss ensures latent codes retain layer-specific information necessary for routing. The orthogonality regularization promotes diversity among skill vectors in the Experience Bank.

5 Experiments

5.1 Experimental Setup

Datasets. We train ReX on two complementary reasoning corpora: (1) OpenR1-Math-220k (multi-step mathematical reasoning), and (2) OpenThoughts3-1.2M (covering mathematics, science, and code) (Guha et al., 2025). We evaluate on five benchmarks: (1) GPQA (graduate-level multiple-choice questions in biology, physics, and chemistry) (Rein et al., 2023), (2) three specialized subsets of MMLUPro (we denote as Chemistry, Math, and Economics respectively) (Wang et al., 2024b), (3) MATH500 (a subset of the MATH benchmark with advanced math problems) (Lightman et al., 2023). We report accuracy throughout; MATH500 uses strict exact-answer matching. All methods use the same filtered training instances and identical prompting/decoding budgets for fairness. Dataset filtering/sampling and evaluation subset construction are detailed in Appendix A.

Baselines. We compare against two families that operationalize "experience" differently.

- **Text-based.** (1) **In-context learning** (Brown et al., 2020b). (2) **LLM+RAG**: retrieval-augmented generation that injects retrieved passages into the prompt without updating model parameters (Lewis et al., 2020); (3) **Reflexion** (Shinn et al., 2023): language-level self-feedback that records and reuses textual insights to guide subsequent attempts; (4) **Self-Refine** (Madaan et al., 2023): iterative generate–critique–rewrite cycles that improve responses purely via text edits. Detailed descriptions are provided in the Appendix A.

Category	Method	GPQA	Chemistry	Economics	Math	MATH500	Average
	Base	0.1360	0.1450	0.2850	0.2950	0.3106	0.2343
Text	ICL	0.1919	0.1100	0.3550	0.0850	0.2942	0.2072
	LLM+RAG	0.2121	0.1500	0.2550	0.2250	0.2888	0.2262
	Self-Refine	0.1263	0.1000	0.3250	0.2600	0.2670	0.2157
	Reflexion	0.1768	0.1400	0.2600	0.2250	0.3106	0.2225
Parameter	LoRA	0.3232	0.2500	0.2750	0.4500	0.3678	0.3332
	HydraLoRA	0.3434	0.3050	0.3150	0.4400	0.3597	0.3526
	MoSLoRA	0.3687	0.2750	0.3150	0.4800	0.3651	0.3608
	VB-LoRA	0.2071	0.2650	0.3250	0.4550	0.3651	0.3234
	SMoRA	0.3485	0.2850	0.3350	0.4800	**0.3733**	0.3644
	ReX	**0.3889**	**0.3100**	**0.3450**	**0.5100**	0.3488	**0.3805**

Table 1: Overall performance comparisons on the Llama 3.2-3B-Instruct backbone across five reasoning benchmarks (*GPQA*, *Chemistry*, *Economics*, *Math*, and *MATH500*). Bold values indicate the best performance in each column.

Category	Method	GPQA	Chemistry	Economics	Math	MATH500	Average
	Base	0.1645	0.1150	0.3250	0.0900	0.0763	0.1542
Text	ICL	0.2121	0.0850	0.3450	0.1200	0.0654	0.1655
	LLM+RAG	0.1566	0.1150	0.3100	0.1300	0.0899	0.1603
	Self-Refine	0.1970	0.1350	0.2600	0.1050	0.0708	0.1536
	Reflexion	0.1364	0.1050	0.3200	0.1200	0.0681	0.1499
Parameter	LoRA	0.2525	0.2300	0.3650	0.2750	0.0899	0.2425
	HydraLoRA	0.2121	0.1850	0.3650	0.2700	0.1035	0.2271
	MoSLoRA	0.1818	0.1850	0.3250	0.2800	0.0926	0.2129
	VB-LoRA	0.2222	0.1800	0.3350	0.2200	0.0872	0.2089
	SMoRA	0.2323	0.1500	0.3700	0.2500	0.1008	0.2206
	ReX	**0.2727**	**0.2350**	**0.3900**	**0.2900**	**0.1199**	**0.2615**

Table 2: Results on the Mistral-7B-Instruct backbone using the same experimental setup.

- **Parameter-based.** (1) **LoRA** (Hu et al., 2021): parameter-efficient fine-tuning via low-rank updates injected into selected weight matrices; (2) **HydraLoRA** (Tian et al., 2025): asymmetric sharing of a single $\mathbf{A}$ matrix with routed, task-specialized $\mathbf{B}$ matrices for sample-wise selection; (3) **MoSLoRA** (Wu et al., 2024a): a lightweight mixing matrix $\mathbf{W}$ that flexibly combines r^2 rank-1 subspaces between $\mathbf{A}/\mathbf{B}$ with negligible overhead; (4) **SMoRA** (Zhao et al., 2025b): treats each LoRA rank as an expert and sparsely activates a subset per input to mitigate multi-task interference. (5)**VB-LoRA.** (Li et al., 2024c): composes LoRA's low-rank matrices from a shared vector bank via a differentiable top-k mixer, delivering extreme parameter efficiency with comparable or better performance.

Implementation Details. We use Llama-3.2-3B-Instruct (Grattafiori et al., 2024) and Mistral-7B-Instruct-v0.3 (Jiang et al., 2023) as the backbone models for all experiments. Considering computational constraints, we filter the OpenR1-Math-220k dataset and the science subset of OpenThoughts3-1.2M, then randomly sample 10,000 instances from their mixture to construct the training set. All PEFT methods share the same prompting and decoding protocol across datasets, and we avoid domain-specific hyperparameter tuning. ReX introduces only a small constant-factor overhead relative to standard LoRA, dominated by low-dimensional projections and routing; we provide an explicit compute and training-memory analysis in Appendix C. Further implementation details are provided in the Appendix B.

5.2 Main Results

ReX achieves the best overall performance among text-based and parameter-efficient methods. As shown in Tables 1 and 2, ReX consistently outperforms all baselines on both Llama3.2-3B-Instruct and Mistral-7B-Instruct. On Llama-3.2-3B, ReX achieves an average accuracy of 38.05%, compared to 36.44% for SMoRA and 22% for text-based methods, yielding absolute gains of +1.6

Model	Variant	GPQA	Chemistry	Economics	Math	MATH500	Average
llama3.2-3B	Original	**0.3889**	**0.3100**	**0.3450**	**0.5100**	0.3488	**0.3805**
	w/o $\mathcal{L}_{\text{kl}} + \mathcal{L}_{\text{rec}}$	0.3232	0.2100	0.3050	0.4750	0.3433	0.3313
	w/o $\mathcal{L}_{\text{orth}}$	0.3636	0.2800	0.3350	0.4450	**0.3597**	0.3567
Mistral-7B	Original	**0.2727**	**0.2350**	0.3900	**0.2900**	**0.1199**	**0.2615**
	w/o $\mathcal{L}_{\text{kl}} + \mathcal{L}_{\text{rec}}$	0.2121	0.2100	0.3800	0.2400	0.0954	0.2275
	w/o $\mathcal{L}_{\text{orth}}$	0.2576	0.1750	**0.3950**	0.2300	0.0681	0.2251

Table 3: Impact of loss components on ReX performance.

Model	Initialization	GPQA	Chemistry	Economics	Math	MATH500	Average
llama3.2-3B	*Identity*	0.3182	0.2800	0.3150	0.4800	**0.3733**	0.3533
	Orthogonal	0.3030	0.2600	**0.3900**	0.4450	0.3460	0.3488
	Kaiming_uniform	**0.3889**	**0.3100**	0.3450	**0.5100**	0.3488	**0.3805**
Mistral-7B	*Identity*	0.2273	0.1450	0.3800	0.2000	0.1035	0.2112
	Orthogonal	0.2525	0.1750	**0.4550**	0.2450	0.0736	0.2402
	Kaiming_uniform	**0.2727**	**0.2350**	0.3900	**0.2900**	**0.1199**	**0.2615**

Table 4: Effect of initialization strategies for the skill-mixing matrix $\mathbf{M}$.

points over the strongest parameter-efficient baseline and over +15 points relative to text-based approaches. On Mistral-7B, it reaches 26.15%, outperforming LoRA with +1.9 points and textual methods with at least +9.6 points. We also conducted experiments on Llama 3.1-8B, and the results were consistent (See result in Appendix Table 5). These results confirm that experience-centric composition enables more effective skill reuse than conventional task-specific adaptation.

Parameter-based methods substantially outperform text-based approaches. On Llama3.2-3B, parameter-based variants achieve 32–38%, while text-based methods remain around 19–23%. This gap suggests that under medium-scale models, in-context prompting and retrieval struggle to internalize complex reasoning, whereas parametric encoding supports more stable generalization.

ReX generalizes strongly across domains. As shown in Tables 1 and 2, on the out-of-domain Economics dataset, it manages to achieve 34.5% accuracy on Llama3.2 and 39.0% accuracy on Mistral-7B—outperforming the original LoRA by +7 points and +2.5 points, respectively. These results indicate that the learned latent experiences transfer effectively beyond training domains, capturing reusable reasoning patterns rather than dataset-specific shortcuts.

5.3 Ablation Study

Latent coding and orthogonal regularization are key to ReX's performance. As shown in Table 3, removing the VAE-related losses ($\mathcal{L}_{\text{kl}} + \mathcal{L}_{\text{rec}}$)

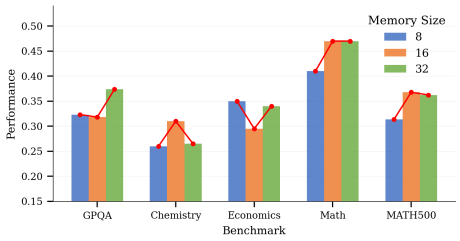

Figure 2: Influence of Experience Bank size.

causes the largest degradation across both backbones, confirming that latent experience codes are essential for capturing transferable reasoning patterns. Eliminating the orthogonality term ($\mathcal{L}_{\text{orth}}$) also consistently reduces accuracy, indicating that promoting diversity among skill vectors enhances compositional flexibility and mitigates redundancy. Together, these results validate that both the latent encoder and orthogonal regularization are critical for stable and generalizable adaptation.

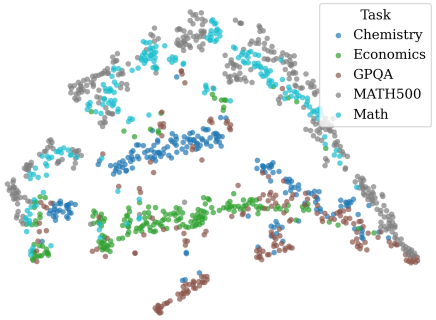

Figure 3: t-SNE visualization of latent experience codes.

Initialization and memory size further influence model stability and transfer. Table 4 shows that Kaiming_uniform initialization yields the most stable and accurate convergence for the skill-mixing

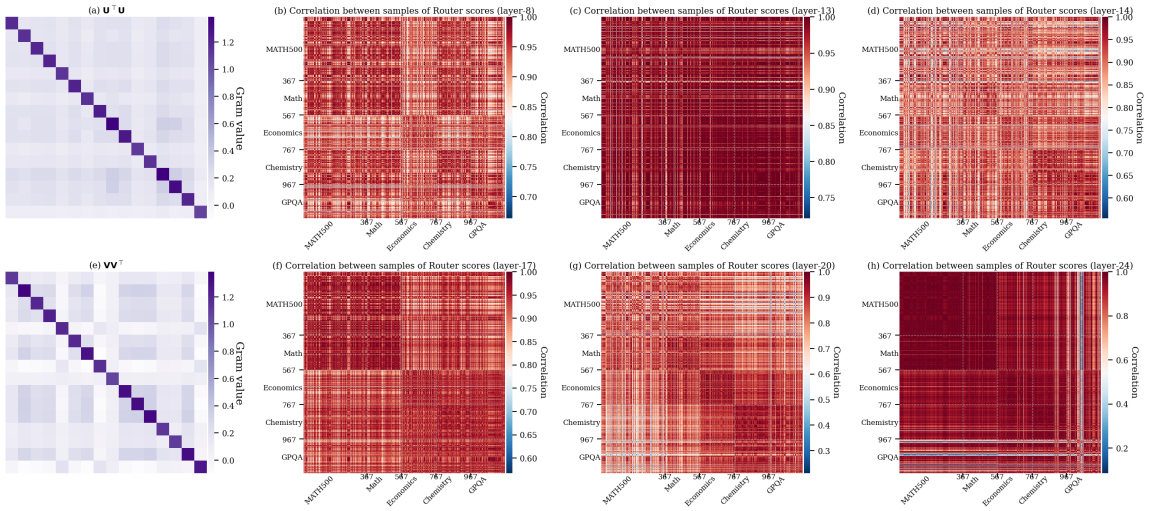

Figure 4: Correlation of the Experience Bank (Left) and Experience Routing outputs on different layers (Right).

matrix $\mathbf{M}$, outperforming Identity and Orthogonal schemes. As illustrated in Figure 2, enlarging the Experience Bank improves performance up to a saturation point, suggesting a balance between memory capacity and selection efficiency. Overall, these findings highlight that well-initialized mixing and appropriately scaled memory jointly enhance ReX's stability and generalization.

Effect of training set size. Under a fixed protocol, scaling the training set from 5k to 20k consistently improves average accuracy (Appendix D.2, Table 6), indicating that the shared bank and instance-wise routing benefit from more data.

Sensitivity to top-k. Varying the router top-k shows robust performance across a reasonable range and improved averages with larger k (Appendix D.3, Table 7).

5.4 Qualitative Analysis

Learned latent experience codes show cluster effect and related domain intersections. Figure 3 visualizes the latent experience codes extracted by the VAE using t-SNE. Samples from different domains form clearly separable clusters, indicating that the encoder manages to learn domain-specific experience code patterns. At the same time, meaningful overlaps appear between related domains, which also matches reality: GPQA and Chemistry share regions due to overlapping chemical knowledge, while Math and MATH500 exhibit partial proximity despite differing formats. Other related domain intersections such as those between Economics and scientific domains also align with the common computation skills required in both. Notably, such domain structure is *emergent* from unsupervised representations rather than imposed by task labels.

The Experience Bank preserves diverse skills and supports selective composition. Figure 4 analyzes the internal structure and routing dynamics of the Experience Bank. On the one hand, Subfigures (a) and (e) show that the base skill matrices $\mathbf{U}$ and $\mathbf{V}$ remain approximately orthogonal, validating that the orthogonality constraint $\mathcal{L}_{\text{orth}}$ effectively encourages non-redundant skill discovery. On the other hand, Subfigures (b–d) and (f–h) illustrate distinct routing behaviors across layers: some exhibit uniformly high correlations, reflecting the use of generic reasoning skills, while others show stronger within-domain correlations, indicating task-specific specialization. The presence of cross-domain correlations further demonstrates that ReX captures reusable skill components that generalize beyond individual domains, validating the effectiveness of its experience-centric composition mechanism.

6 Conclusion

In this paper, we introduced ReX, an experience-centric adaptation framework that treats latent experiences—recurring reasoning patterns and skills—as the fundamental units for LLM specialization. By integrating a VAE-based encoder, a shared Experience Bank, and an adaptive Experience Router, ReX enables dynamic composition of reusable skills across inputs and domains. Our experiments on diverse scientific and mathematical benchmarks demonstrate that ReX consistently outperforms text-based and parameter-based baselines, achieving strong generalization even on out-of-domain tasks. Through analyses of the latent space and routing behavior, we further show that the

model learns disentangled and transferable skills, validating the effectiveness of experience-centric adaptation. Future work will explore hierarchical experience organization and dynamic expansion of the Experience Bank to further enhance scalability and continual learning.

Limitations

While ReX achieves consistent improvements by representing and composing reusable experiences through latent routing, several limitations remain. First, the VAE-based encoder introduces additional computational overhead and potential instability, as ReX jointly optimizes the task loss, KL and reconstruction terms, and orthogonality regularization for the experience bank. Although these components empirically improve stability and generalization, they increase tuning complexity and may lead to occasional training variance. Second, our experiments mainly focus on reasoning and STEM domains; extending ReX to open-ended dialogue, writing, or other generative tasks remains future work. Finally, we have not yet evaluated the approach on multimodal data or analyzed its sensitivity to data scale, which will be explored in subsequent studies.

Acknowledgments

This research is supported by the National Natural Science Foundation of China under the Joint Fund Key Project (U24A20328) and the General Program (62476071), by the Guangdong Basic and Applied Basic Research Foundation (2025A1515011732), and by the Beijing Natural Science Foundation (4262074, L254018). This research is also supported by the National Research Foundation, Singapore under its National Large Language Models Funding Initiative (AISG Award No. AISG-NMLP-2024-002), and by the Ministry of Education, Singapore, under its AcRF Tier 2 Funding (Proposal ID: T2EP20123-0052). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of the National Research Foundation or the Ministry of Education, Singapore.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020a. Language models are few-shot learners. NIPS '20. Curran Associates Inc.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020b. Language Models are Few-Shot Learners.

Lucas Caccia, Edoardo Ponti, Zhan Su, Matheus Pereira, Nicolas Le Roux, and Alessandro Sordoni. 2023. Multi-head adapter routing for cross-task generalization. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, pages 56916–56931. Curran Associates Inc.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. 2025. Text-to-LoRA: Instant Transformer Adaption.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. DeepSeek-V3 Technical Report.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: efficient finetuning of quantized llms. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23. Curran Associates Inc.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The Llama 3 Herd of Models.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, and 31 others. 2025. Openthoughts: Data recipes for reasoning models. *Preprint*, arXiv:2506.04178.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP.

Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339. Association for Computational Linguistics.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2024. LoraHub: Efficient Cross-Task Generalization via Dynamic LoRA Composition.

Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards Reasoning in Large Language Models: A Survey. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1049–1065, Toronto, Canada. Association for Computational Linguistics.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B.

Diederik P. Kingma and Max Welling. 2022. Auto-Encoding Variational Bayes.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pages 9459–9474. Curran Associates Inc.

Dengchun Li, Yingzi Ma, Naizheng Wang, Zhengmao Ye, Zhiyuan Cheng, Yinghao Tang, Yan Zhang, Lei Duan, Jie Zuo, Cal Yang, and Mingjie Tang. 2024a. MixLoRA: Enhancing Large Language Models Fine-Tuning with LoRA-based Mixture of Experts.

Jiawei Li, Yizhe Yang, Yu Bai, Xiaofeng Zhou, Yinghao Li, Huashan Sun, Yuhang Liu, Xingpeng Si, Yuhao Ye, Yixiao Wu, Yiguan Lin, Bin Xu, Bowen Ren, Chong Feng, Yang Gao, and Heyan Huang. 2024b. Fundamental Capabilities of Large Language Models and their Applications in Domain Scenarios: A Survey. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11116–11141. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation.

Yang Li, Shaobo Han, and Shihao Ji. 2024c. VB-LoRA: Extreme Parameter Efficient Fine-Tuning with Vector Banks.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's Verify Step by Step.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. SELF-REFINE: Iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, pages 46534–46594. Curran Associates Inc.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks.

Andrea Matarazzo and Riccardo Torlone. 2025. A Survey on Large Language Models with some Insights on their Capabilities and Limitations. *Preprint*, arXiv:2501.04040.

Matthew Lyle Olson, Musashi Hinck, Neale Ratzlaff, Changbai Li, Phillip Howard, Vasudev Lal, and Shao-Yen Tseng. 2025. Probing the representational power of sparse autoencoders in vision models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 6167–6177.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. GPT-4 Technical Report.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA. Association for Computing Machinery.

Mathis Pink, Qinyuan Wu, Vy Ai Vo, Javier Turek, Jianing Mu, Alexander Huth, and Mariya Toneva.

2025. Position: Episodic Memory is the Missing Piece for Long-Term LLM Agents. *Preprint*, arXiv:2502.06975.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A Graduate-Level Google-Proof Q&A Benchmark.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, pages 8634–8652. Curran Associates Inc.

David Silver and Richard S Sutton. 2025. Welcome to the era of experience. *Google AI*, 1.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. 2025. HydraLoRA: An Asymmetric LoRA architecture for efficient fine-tuning. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, volume 37 of *NIPS '24*, pages 9565–9584, Red Hook, NY, USA. Curran Associates Inc.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models.

Luping Wang, Sheng Chen, Linnan Jiang, Shu Pan, Runze Cai, Sen Yang, and Fei Yang. 2025. Parameter-efficient fine-tuning in large language models: A survey of methodologies. *Artificial Intelligence Review*, 58(8):227.

Xindi Wang, Mahsa Salmani, Parsa Omidi, Xiangyu Ren, Mehdi Rezagholizadeh, and Armaghan Eshaghi. 2024a. Beyond the limits: A survey of techniques to extend the context length in large language models. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, pages 8299–8307, Jeju, Korea.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024b. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark.

Taiqiang Wu, Jiahao Wang, Zhe Zhao, and Ngai Wong. 2024a. Mixture-of-Subspaces in Low-Rank Adaptation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

Xun Wu, Shaohan Huang, and Furu Wei. 2024b. Mixture of LoRA Experts.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models.

Jingfan Zhang, Yi Zhao, Dan Chen, Xing Tian, Huanran Zheng, and Wei Zhu. 2024a. MiLoRA: Efficient Mixture of Low-Rank Adaptation for Large Language Models Fine-tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 17071–17084, Miami, Florida, USA. Association for Computational Linguistics.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning.

Yafeng Zhang, Zilan Yu, Yuang Huang, and Jing Tang. 2024b. CLLMFS: A Contrastive Learning enhanced Large Language Model Framework for Few-Shot Named Entity Recognition. *Preprint*, arXiv:2408.12834.

Yuxuan Zhang and Ruizhe Li. 2025. DLP-LoRA: Efficient Task-Specific LoRA Fusion with a Dynamic, Lightweight Plugin for Large Language Models.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2025a. A Survey of Large Language Models.

Ziyu Zhao, Yixiao Zhou, Zhi Zhang, Didi Zhu, Tao Shen, Zexi Li, Jinluan Yang, Xuwu Wang, Jing Su, Kun Kuang, Zhongyu Wei, Fei Wu, and Yu Cheng. 2025b. Each Rank Could be an Expert: Single-Ranked Mixture of Experts LoRA for Multi-Task Learning.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, volume 38 of *AAAI'24/IAAI'24/EAAI'24*, pages 19724–19731. AAAI Press.

A Datasets

A.1 Training Datasets

We train our models on two reasoning corpora: OpenR1-Math-220k and OpenThoughts3-1.2M.

OpenR1-Math-220k. OpenR1-Math-220k [1] is a large-scale dataset for mathematical reasoning, containing 220k math problems with reasoning traces generated by DeepSeek-R1.

[1] https://huggingface.co/datasets/open-r1/OpenR1-Math-220k

OpenThoughts3-1.2M. OpenThoughts3-1.2M [2] is an open-source reasoning dataset designed to enhance the generalization and compositional reasoning abilities of language models across mathematics, code, and science. It comprises approximately 1.2M high-quality samples curated to capture multi-step reasoning patterns and domain-specific logic. In this work, we select the *science* subset of OpenThoughts3-1.2M, which primarily consists of physics and chemistry problems.

Data Filtering and Sampling. Considering computational constraints, we remove intermediate *Thinking* traces and retain only the final reasoning outputs. We further filter out samples exceeding a total token length of 1024 and exclude responses shorter than 256 tokens to ensure training efficiency and output quality. Finally, we randomly sample 10,000 instances from a mixture of OpenR1-Math-220k and OpenThoughts3-1.2M at a 3:5 ratio to construct the final training set. Crucially, all PEFT baselines are trained on the exact same filtered instances with identical prompting/decoding settings, so the comparison does not favor ReX.

A.2 Evaluation Datasets

We evaluate our models on a diverse set of benchmarks covering mathematical and scientific reasoning:

- **MATH500** (Lightman et al., 2023): A subset of the MATH benchmark consisting of 500 math problems categorized by topic and difficulty. To maintain evaluation challenge, we select 367 problems with difficulty level ≥ 3 as the test set.
- **GPQA** (Rein et al., 2023): A graduate-level multiple-choice dataset spanning biology, physics, and chemistry. Following prior work, we adopt the GPQA-Diamond subset (198 questions) for testing.
- **MMLUPro-Chemistry, -Economics, and -Math** (Wang et al., 2024b): MMLUPro is a multi-domain benchmark covering 14 subjects. We employ three of its subsets (Chemistry, Economics, and Math) with 200 samples each, randomly selected from their respective domains for evaluation.

[2] https://huggingface.co/datasets/open-thoughts/OpenThoughts3-1.2M

B Baselines and Implementation Details

We compare our approach with two categories of adaptation methods: text-based and parametric. Text-based methods leverage external context or iterative refinement without modifying model weights, while parametric approaches introduce lightweight trainable parameters for efficient internal adaptation.

Text-based methods.

- **In-context Learning (ICL):** Provides the model with a few exemplars in the prompt to enable task adaptation through context rather than parameter updates. For each evaluation sample, one example from the same dataset is used as the in-context demonstration (Brown et al., 2020b).
- **LLM + RAG (Retrieval-Augmented Generation):** Augments the model with retrieved external documents to supply factual or domain knowledge during inference. We retrieve the most semantically similar instance from the training set and append it to the prompt (Lewis et al., 2020).
- **Reflexion:** Reinforces language agents via self-generated feedback. After task execution, the agent reflects on performance, stores textual reflections, and incorporates them for future reasoning. The maximum number of reflection iterations is set to three (Shinn et al., 2023).
- **Self-Refine:** Enables iterative self-improvement as the model critiques and refines its own outputs without external supervision. The refinement process runs for up to three iterations (Madaan et al., 2023).

Parametric methods.

- **LoRA:** Constrains fine-tuning updates to a low-rank subspace by injecting small trainable matrices into frozen weights. All LoRA-based baselines use a rank of 8, unless otherwise stated (Hu et al., 2021).
- **HydraLoRA:** Extends LoRA to heterogeneous data via one shared down-projection and multiple expert up-projections coordinated by a learnable router for domain-aware adaptation. We set the rank to 4 and the number of experts to 4 (Tian et al., 2025).

- **VB-LoRA:** Decomposes LoRA parameters into reusable sub-vectors sampled from a global vector bank, realizing extreme compression and sharing through differentiable top-k selection (Li et al., 2024c).
- **SMoRA:** Treats each LoRA rank as an individual expert activated sparsely per input, providing fine-grained specialization while mitigating task interference (Zhao et al., 2025b).
- **MoSLoRA:** Introduces a lightweight mixer between LoRA projections to blend multiple subspaces, enhancing representational flexibility with negligible additional cost (Wu et al., 2024a).

Compared to routed LoRA variants such as HydraLoRA, ReX differs in both the *unit* and the *signal* of routing: HydraLoRA routes among parameter-block experts (e.g., multiple dense $\mathbf{B}$ matrices) typically optimized for heterogeneous-task efficiency, whereas ReX routes over a shared bank of atomic *skill vectors* and synthesizes LoRA factors via top-k compositions. Moreover, ReX routes based on an instance-level latent code extracted from layer observations, aiming to capture recurring reasoning patterns without requiring task labels or task descriptions.

For our method, the composed LoRA rank is 8, with a memory size of 16 and a router top-k value of 8. During generation, the maximum number of output tokens is set to 1024, and sampling is disabled. All experiments were conducted on NVIDIA A100 GPUs with 40 GB of VRAM. We conducted our experiments using Huggingface Transformers and PEFT libraries.

C Efficiency and Resource Analysis

C.1 Compute overhead.

We analyze a single transformer layer and a single linear projection $W \in \mathbb{R}^{d\times d}$ with input hidden states $H \in \mathbb{R}^{T\times d}$. The dominant backbone computation is the matrix multiplication $HW^\top$, whose forward complexity is $\mathcal{O}(Td^2)$.

Standard LoRA adds a rank-r update $\Delta W = AB$, which can be implemented as $(HB^\top)A^\top$ in the forward pass, incurring an additional cost of $\mathcal{O}(Tdr)$. Under typical PEFT regimes where $r \ll d$, this term does not change the leading-order scaling relative to the backbone.

Keeping the final update rank fixed to r, ReX introduces three additional components on top of LoRA: (i) a sequence pooling to obtain a layer-level observation, costing $\mathcal{O}(Td)$; (ii) a single-hidden-layer VAE with latent size d_z to produce an experience code and optimize reconstruction and KL regularization during training, where the encoder/decoder projections contribute $\mathcal{O}(2dd_z)$, the reconstruction loss (e.g., $\|r-\hat{r}\|_2^2$) costs $\mathcal{O}(d)$, and the KL term costs $\mathcal{O}(d_z)$; and (iii) a router that outputs composition weights in the latent space, costing $\mathcal{O}(d_z r r_b)$.

Overall, the extra computation of ReX relative to LoRA is

$$C_{\text{ReX-overhead}} \approx \mathcal{O}\big(Td + 2dd_z + d + d_z r r_b\big).$$

In our setting, $d_z \approx r_b \approx 2r \ll d$, with r a small PEFT hyperparameter (so $d \gg r^2$). Consequently, $\mathcal{O}(2dd_z) = \mathcal{O}(dr)$ typically dominates $\mathcal{O}(d_z r r_b) = \mathcal{O}(r^3)$, and the added overhead can be approximated as $\mathcal{O}(Td + dr)$. The overall computation remains dominated by the backbone $\mathcal{O}(Td^2)$ and the LoRA term $\mathcal{O}(Tdr)$, implying that ReX introduces only a small constant-factor overhead without changing the leading-order scaling in T and d.

C.2 Memory overhead.

Training memory can be decomposed as

$$M = M_{\text{base}} + M_{\text{train}} + M_{\text{act}}.$$

Here M_{base} is the frozen backbone memory and is identical for ReX and LoRA. The activation term M_{act} is typically dominated by token-level transformer activations (scaling as $\Theta(Td)$). ReX adds only instance-level low-dimensional intermediates (the pooled observation $r \in \mathbb{R}^d$, the VAE latent $z \in \mathbb{R}^{d_z}$, and router scores $S \in \mathbb{R}^{r\times r_b}$), whose additional footprint is $\Theta(d + d_z + r r_b) \approx \Theta(d)$ and is negligible compared to $\Theta(Td)$.

The trainable-parameter term M_{train} scales linearly with the number of trainable parameters (including gradients and optimizer states). For a single $d \times d$ linear layer, LoRA has $P_{\text{LoRA}} = 2dr$, whereas ReX adds a per-layer bank (U, V) $(2dr_b)$, a single-hidden-layer VAE $(2dd_z)$, and a router that outputs rank-wise scores $(d_z r r_b)$, yielding $P_{\text{ReX}} \approx 2dr_b + 2dd_z + d_z r r_b$. Under $d_z \approx r_b \approx 2r \ll d$, we still have $P_{\text{ReX}} = \Theta(dr)$, i.e., ReX increases M_{train} by only a small constant factor relative to LoRA. Overall, ReX remains effectively comparable to LoRA in training-time memory, i.e., $M^{\text{ReX}} \approx M^{\text{LoRA}}$.

Category	Method	GPQA	Chemistry	Economics	Math	MATH500	Average
	Base	0.2121	0.1800	0.3350	0.3200	0.3161	0.2726
Text	ICL	0.2677	0.2000	0.5250	0.2800	0.3270	0.3199
	LLM+RAG	0.2677	0.2300	0.4100	0.3700	0.3079	0.3171
	Self-Refine	0.2576	0.2100	0.4600	0.3850	0.2806	0.3186
	Reflexion	0.2677	0.3050	0.3250	0.4750	0.4005	0.3546
Parameter	LoRA	0.3485	0.3850	0.5250	0.5250	0.3651	0.4297
	HydraLoRA	0.3636	0.3900	0.4900	0.5000	0.3624	0.4212
	MoSLoRA	0.3636	0.3850	0.5100	0.5300	0.3651	0.4308
	VB-LoRA	0.3384	0.3300	0.5300	0.5100	0.3869	0.4191
	SMoRA	0.3737	0.3650	0.4950	0.5000	0.3842	0.4236
	ReX	**0.3838**	**0.3900**	**0.5550**	**0.5650**	**0.3896**	**0.4567**

Table 5: Results for Llama3.1-8B-instruct backbone across tasks.

D Additional Experiments

D.1 Scaling to Larger Models

ReX continues to achieve the best performance as model size increases. Table 5 extends our evaluation to the larger Llama3.1-8B-Instruct backbone. ReX again outperforms all text-based and parameter-efficient baselines, achieving the highest average accuracy (45.67%) across all benchmarks. These results demonstrate that the experience-centric design of ReX scales effectively with model capacity, enabling consistent gains even under stronger base models.

Larger models show improved contextual learning ability. Compared with Llama3.2-3B, the 8B backbone exhibits notably higher base accuracy, indicating that larger LLMs can extract more useful information directly from context.

D.2 Scaling to Larger Training Sets

ReX consistently benefits from more training instances under a fixed training protocol. Table 6 varies the training set size (5k/10k/20k) while keeping the backbone and hyperparameters unchanged. ReX shows a clear upward trend in overall performance, improving the average accuracy from 35.11% (5k) to 37.16% (20k), suggesting that the shared Experience Bank and instance-wise routing can effectively absorb additional supervision.

Table 6: Results for Llama-3.2-3B-Instruct backbone across tasks with varying training set sizes.

Size	GPQA	Chemistry	Economics	Math	Math500	Average
5k	0.3232	0.2500	0.3800	0.4550	0.3488	0.3511
10k	0.3485	0.2850	0.3300	0.4350	0.3651	0.3546
20k	0.3788	0.2950	0.3700	0.4650	0.3597	0.3716

D.3 Sensitivity to top-k in routing

ReX generally improves with larger top-k. Table 7 reports results for $k \in \{2, 4, 8\}$ under the same backbone and training setup. Overall performance increases as k grows, with the average accuracy rising from 34.95% ($k = 2$) to 38.05% ($k = 8$), indicating that allowing each rank position to aggregate more skill atoms typically yields stronger composed adapters.

Table 7: Results for different top-k settings with the Llama-3.2-3B-Instruct backbone.

Top-k	GPQA	Chemistry	Economics	Math	Math500	Average
2	0.3333	0.2550	0.2800	0.4700	0.3815	0.3495
4	0.3333	0.2800	0.3250	0.5300	0.3379	0.3579
8	0.3889	0.3100	0.3450	0.5100	0.3488	0.3805