Transformer^2: Self-adaptive LLMs

Qi Sun*, **Edoardo Cetin***, **Yujin Tang***
Sakana AI, Japan
{qisun,edo,yujintang}@sakana.ai
*Equal contributions

Abstract

Self-adaptive large language models (LLMs) aim to solve the challenges posed by traditional fine-tuning methods, which are often computationally intensive and inflexible for diverse tasks. We introduce Transformer^2, a novel framework that adapts LLMs for unseen tasks in real-time by selectively adjusting singular components of weight matrices, using a two-pass mechanism: task identification followed by mixing task-specific "expert" vectors to best cope with test-time conditions. Our approach outperforms ubiquitous methods like LoRA with fewer parameters and greater efficiency across various LLM architectures and modalities, and offers a scalable solution for enhancing the adaptability and task-specific performance of LLMs, paving the way for truly self-organizing AI systems.

1 Introduction

Self-adaptive large language models (LLMs) would represent a significant advancement in artificial intelligence, enabling real-time adaptation to various tasks and contexts. While compositionality and scalability are crucial for effective adaptation, current LLM training methodologies fall short of achieving both these properties simultaneously. Our research aims to present a solution to address these gaps.

In principle, the first step toward achieving self-adaptive LLMs can be realized through the development of specialized expert modules, each fine-tuned (Kaplan et al., 2020) via techniques such as low-rank adaptation (LoRA) (Hu et al., 2021). However, several challenges need to be addressed to make this approach both scalable and compositional: (1) multiple expert modules significantly increase the number of parameters; (2) expert modules are often prone to overfitting; and (3) flexible composition of these experts is still an open problem.

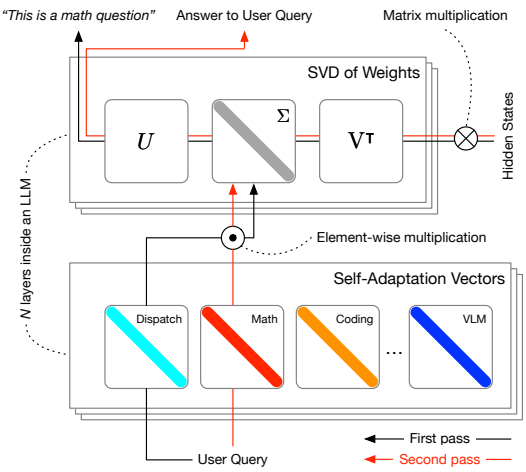

Figure 1: **Overview of Transformer^2.** In training, we tune the singular values of the weight matrices to generate a set of "expert" vectors specializing in different tasks. In inference, a two-pass process is adopted where the first applies the expert and the second generates the answer.

To overcome these limitations, we first propose SVF, a novel parameter-efficient fine-tuning (PEFT) method to obtain effective building blocks for self-adaptation. SVF works by extracting and selectively tuning only the singular values within the model's weight matrices. By focusing on

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

this essential and principled parameterization, our approach mitigates the risk of overfitting, drastically reduces computational demands, and allows for inherent compositionality.

We then introduce our full Transformer^2 framework, which entails a two-pass inference mechanism to produce dynamically adapted weights targeted for the test-time conditions (Figure 1). We design three different adaptation strategies that can be used within Transformer^2, which we show provide monotonic performance benefits with increasing access to the test-time conditions. We evaluate SVF and the full Transformer^2 framework through extensive experiments across a diverse range of LLMs and tasks. SVF outperforms traditional efficient fine-tuning methods like LoRA on domain-specific datasets with far fewer parameters. Transformer^2 further improves performance, even for out-of-distribution tasks like visual QA. Our analysis even shows that Transformer^2 allows the reuse of SVF experts across different LLMs. In summary, our key technical contributions are:

- The development of Transformer^2 as a pivotal self-adaptation framework for LLMs, providing a blueprint to adapt the behavior of LLMs from a growing set of pre-trained skills.
- The introduction of SVF, a novel PEFT method trainable with RL on small datasets, producing compact expert vectors with inherent compositionality.
- The implementation of three adaptation strategies, effectively dispatching SVF-trained experts with properties designed to cope with different deployment scenarios.

2 Related works

Self-adaptive LLMs operate at macro (multiple collaborating LLMs) and micro (internal adaptations) levels. Microview adaptations often use Mixture of Experts (MoE) for dynamic routing (Fedus et al., 2022). Low-rank adaptation methods like LoRA (Hu et al., 2021) enable efficient fine-tuning. Recent approaches leverage SVD for LLM fine-tuning, either using minor components (Wang et al., 2024) or top singular vectors (Bałazy et al., 2024). The most related work (Lingam et al., 2024) explores SVD-based sparsification but doesn't focus on self-adaptive LLMs or use reinforcement learning for efficiency. We refer to Appendix D for references to the wider literature.

3 Methods

3.1 Preliminaries

Singular value decomposition (SVD) offers a fundamental view of matrix multiplications. In neural networks, each weight matrix $W \in \mathbb{R}^{n\times m}$ can be decomposed into three components $W = U\Sigma V^\intercal$, yielding semi-orthogonal matrices $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ together with an ordered vector of r singular values arranged in the diagonal matrix $\Sigma \in \mathbb{R}^{r\times r}$.

Cross-entropy method (CEM) is a Monte Carlo method for importance sampling and optimization (Rubinstein & Kroese, 2004). The method is based on minimizing the KL divergence between two probability distributions $D_{\mathrm{KL}}(P\|Q)$, where P is the target distribution and Q is a maintained distribution. CEM generates a set of samples from Q, evaluates them, and updates the distribution Q with the characteristics of the elite samples. In the standard setup, Q is set to a diagonal multivariate Gaussian, reducing the problem to simply estimating the empirical mean and standard deviation of the latest elites. We illustrate a complete CEM step in the Python pseudocode in Appendix A.4.

3.2 TRANSFORMER^2

The construction of Transformer^2 comprises two main steps, for which we provide an illustrative overview in Figure 2. First, we introduce Singular Value Fine-tuning (SVF), a method to learn *compositional* expert vectors with RL. Then, we describe three different adaptation strategies within Transformer^2, inspired by orthogonal principles. We motivate how the properties of SVF are highly complementary to our adaptation strategies, making Transformer^2 an effective and scalable framework for the design of new self-adaptive LLMs.

Singular value fine-tuning is a key building block in Transformer^2. For any weight matrix W, SVF learns a simple vector $z \in \mathbb{R}^r$ that provides targeted modifications to each singular component of W independently, yielding a new weight matrix $W' = U\Sigma' V^\intercal$, where $\Sigma' = \Sigma \otimes \mathrm{diag}(z)$. This approach offers three main benefits: (1) Negligible parameters: efficient fine-tuning with far fewer

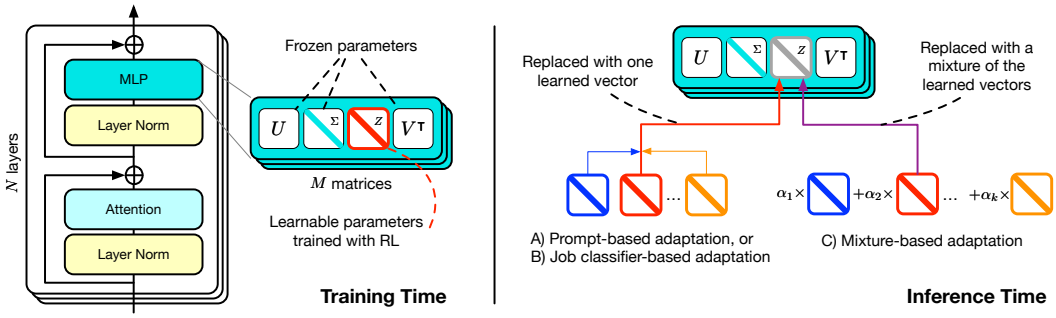

Figure 2: **Method overview.** Left) At training time, we employ SVF and RL to learn the "expert" vectors z's that scale the singular values of the weight matrices. Right) At inference time, we propose three distinct methods to adaptively select/combine the learned expert vectors.

optimized parameters than existing methods. (2) High compositionality: decomposition into independent singular components enables interpretable z vectors, unlike LoRA-based methods. (3) Principled regularization: modifying only pre-existing singular component magnitudes prevents overfitting, allowing fine-tuning on small datasets without risking model collapse. These properties make SVF a foundation block for adapting large language models efficiently and effectively.

End-to-end optimization with RL. We train a set of SVF vectors $\theta_z = \{z_1, \cdots, z_{N\times M}\}$ to fine-tune an arbitrary language model π_{θ_W} parameterized by θ_W with RL, optimizing directly for task performance. Here, $\theta_W = \{W_1, \cdots, W_{N\times M}\}$ is the set of weight matrices, where N is the number of layers and M is the number of weight matrices to fine-tune per layer. We use the seminal REINFORCE algorithm (Williams, 1992) and label each generated answer y_i (for the prompt $x_i \in D$) with a unitary reward based on its correctness $r \in \{-1, 1\}$. Inspired by related applications of RL for optimizing LLMs (Ouyang et al., 2022), we regularize the REINFORCE objective by adding a KL penalty for deviating from the original model's behavior, weighted by a small coefficient $\lambda \in \mathbb{R}^+$. Thus, our final objective function can be written as:

$$J(\theta_z) = \mathbb{E}\left[\log\left(\pi_{\theta_{W'}}(\hat{y}_i \mid x_i)\right) r(\hat{y}_i, y_i)\right] - \lambda D_{\mathrm{KL}}(\pi_{\theta_{W'}} \| \pi_{\theta_W}), \tag{1}$$

where we use $\pi_{\theta_{W'}}$ to denote the resulting language model after substituting the original weight matrices W with W'. While RL is generally considered less stable than next-token prediction objectives, we find the regularization properties of SVF avoid many of the failure modes of prior less-constrained parameterizations (see Section B.1). Thus, combining these complementary components effectively enables us to directly maximize task performance end-to-end.

Self-adaptation is a critical mechanism in nature that has established itself as a core guiding principle in modern system design (Klös et al., 2015). Our initial efforts toward self-adaptive foundation models focus on the inference stage of LLMs, where we devise a simple two-pass adaptation strategy that combines K sets of base "expert" vectors $z^{1:K}$ trained with SVF to provide different kinds of capabilities (e.g., coding, math, etc). In the first inference pass, given an individual input prompt, Transformer^2 executes the model and observes its test-time behavior to derive a new z' vector tailored to its test-time conditions. This adapted z' is then used in the second inference pass to provide an actual response with the newly adapted weights. In this first work, we propose three simple approaches to produce the vector z' during the first inference pass. Below, we provide an outline of each method and refer to Appendix A for additional implementation details.

A) Prompt engineering: Our most basic approach involves constructing an "adaptation" prompt which we use to *ask* the LLM to categorize the input prompt. Based on its response, we then extract one category out of the set of domain topics used to pre-train each SVF expert and, thus, we select the corresponding z' directly from $z^{1:K}$. We also explicitly provide the option for a generic "others" category, allowing the model to use its base weights in case no expert provides appropriate capabilities. We show the format used to construct the adaptation prompt in Appendix A.1

B) Classification expert: A direct extension of the prompt engineering approach comes from using a specialized system to handle task identification. Following the principles of self-adaptation, we apply SVF to fine-tune the base LLM itself to handle this task. In particular, we collect a dataset $D = \{(x_{1,1}, 1), \cdots, (x_{i,k}, k), \cdots\}$ from the K SVF training tasks, where $x_{i,k}$ is the i-th example

from the k-th expert task. Each tuple $(x_{i,k}, k)$ then forms an example to pre-train an additional job classification expert z^c learned in the same fashion as the others. During the first inference pass, we simply load z^c, intending to improve the inherent task classification capabilities of the base model.

C) Few-shot adaptation: Our third approach leverages additional task information by assuming extended access to its test-time conditions beyond individual prompts. Our method is inspired by few-shot prompting techniques, which have been shown to provide performance improvements, allow LLMs to "in-context" learn tasks that were entirely unseen prior to inference (Brown, 2020). For each optimized W, our approach entails producing a new $z' = \sum_{k=1}^{K} \alpha_k z_k$ by linearly interpolating between the K learned SVF vectors, each weighted by the coefficients α_k. We employ CEM to search over the α_k based on the performance on a set of "few-shot prompts", which are specifically held out from the rest of the test prompts and used to evaluate CEM's population samples. In the case of multiple population samples obtaining the same score on these held-out prompts, we break ties by favoring the one with the highest average log-likelihood across its own generated correct answers. We refer to Section A.4, for additional details and discussions of this final approach.

4 Experiments

4.1 Experimental setups

To validate the generality of Transformer^2 we consider three pre-trained LLMs ranging across different model families and architecture sizes: LLAMA3-8B-INSTRUCT, MISTRAL-7B-INSTRUCT-V0.3, and LLAMA3-70B-INSTRUCT. For each model, we obtain three sets of z vectors to maximize performance for GSM8K (Cobbe et al., 2021), MBPP-pro (Austin et al., 2021), and ARC-Easy (Clark et al., 2018), respectively. Additionally, we train a set of z vectors for LLAMA3-8B-INSTRUCT, when applied as the language backbone for TextVQA (Singh et al., 2019). Finally, we evaluate the Transformer^2 on four unseen tasks: MATH (Hendrycks et al., 2021), Humaneval (Chen et al., 2021), ARC-Challenge (Clark et al., 2018), and OKVQA (Marino et al., 2019). In our adaptation experiments, we only consider experts obtained in the pure-language settings. Please refer to Appendix A for additional details and hyper-parameters summary.

4.2 Experimental results

SVF performance We provide results training on each considered task with the LLAMA3-8B-INSTRUCT, MISTRAL-7B-INSTRUCT-V0.3, and LLAMA3-70B-INSTRUCT base models in Table 1. Remarkably, we find that SVF provides considerable performance gains across nearly all tasks and base models. Instead, LoRA experts yield smaller gains and even sporadic performance degradation. To ensure a fair comparison, we provide extensive ablations to both our model and the LoRA baseline considering different architecture and optimization objectives in Appendix B.1).

Adaptation performance With the SVF trained z vectors, we assess the self-adaptation capability of Transformer^2 on unseen tasks. As shown in Table 2, all of our Transformer^2 adaptation strategies demonstrate improvements across all tasks for nearly all the models. In contrast, even the best training LoRAs only provide marginal improvements on the ARC-Challenge task and still significantly deteriorate performance on both MATH and Humaneval. Comparing the three proposed adaptation strategies, we highlight a clear monotonic trend – with more involved strategies and additional information about the test-time condition, self-adaptation appears to be increasingly effective. This

Table 1: **Fine-tuning results.** LLM performance on the test splits of math, coding and reasoning. Normalized scores are in the parentheses.

Method	GSM8K	MBPP-Pro	ARC-Easy
LLAMA3-8B-INSTRUCT	75.89 (1.00)	64.65 (1.00)	88.59 (1.00)
+ LoRA	70.58 (0.93)	**67.68** (**1.05**)	88.97 (1.00)
+ SVF (Ours)	**79.15** (**1.04**)	66.67 (1.03)	**89.56** (**1.01**)
MISTRAL-7B-INSTRUCT-V0.3	42.83 (1.00)	49.50 (1.00)	49.50 (1.00)
+ LoRA	36.09 (0.84)	47.47 (0.96)	47.47 (0.96)
+ SVF (Ours)	**49.74** (**1.16**)	**51.52** (**1.04**)	**85.14** (**1.72**)
LLAMA3-70B-INSTRUCT	85.29 (1.00)	**80.81** (**1.00**)	**89.10** (**1.00**)
+ LoRA	77.26 (0.91)	68.69 (0.85)	88.55 (0.99)
+ SVF (Ours)	**88.32** (**1.04**)	**80.81** (**1.00**)	88.47 (0.99)

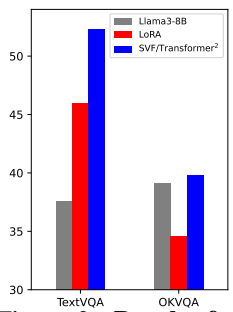

Figure 3: **Results for the VLM domain.**

Table 2: **Self-adaptation on unseen tasks.** Normalized scores are in the parentheses.

Method	MATH	Humaneval	ARC-Challenge
LLAMA3-8B-INSTRUCT 3	24.54 (1.00)	60.98 (1.00)	80.63 (1.00)
+ LoRA	21.68 (0.88)	52.44 (0.86)	81.06 (1.01)
+ Transformer^2 (Prompt)	25.22 (1.03)	61.59 (1.01)	81.74 (1.01)
+ Transformer^2 (Cls-expert)	25.18 (1.03)	62.80 (1.03)	81.37 (1.01)
+ Transformer^2 (Few-shot)	**25.47 (1.04)**	**62.99 (1.03)**	**82.61 (1.02)**
MISTRAL-7B-INSTRUCT-V0.3	13.02 (1.00)	43.29 (1.00)	71.76 (1.00)
+ LoRA	11.18 (0.86)	31.71 (0.73)	**75.77 (1.06)**
+ Transformer^2 (Prompt)	11.86 (0.91)	43.90 (1.01)	72.35 (1.01)
+ Transformer^2 (Cls-expert)	11.60 (0.89)	43.90 (1.01)	74.83 (1.04)
+ Transformer^2 (Few-shot)	**13.39 (1.03)**	**47.40 (1.09)**	75.47 (1.05)
LLAMA3-70B-INSTRUCT	**40.64 (1.00)**	78.66 (1.00)	87.63 (1.00)
+ LoRA	25.40 (0.62)	73.78 (0.94)	83.70 (0.96)
+ Transformer^2 (Prompt)	40.44 (1.00)	**79.88 (1.02)**	**88.48 (1.01)**

trend shows that providing additional or different kinds of information seems to be highly beneficial to our framework, suggesting that Transformer^2 could provide foundation models with new means to continually improve performance when deployed in lifelong settings.

4.3 Cross-model analysis

We explore the potential for our self-adaptation framework to be applied *across different LLMs*. We evaluate whether the SVF expert vectors trained on LLAMA3-8B-INSTRUCT can benefit MISTRAL-7B-INSTRUCT-V0.3, and whether we can perform adaptation across the expert vectors of these two models. We present our main findings in Table 3 and refer to Appendix B for additional detailed results. Surprisingly, positive transfer does occur across the two models, with visible benefits in 2 out of 3 tasks. We note these improvements are due to the inherent ordering of the SVF parameterization, as *randomly shuffling* each SVF vector before applying it to the Mistral model consistently degrades performance. This operation leads to notable performance degradation across tasks. Finally, by performing few-shot adaptation using the SVF vectors collected from both models, the performance of MISTRAL-7B-INSTRUCT-V0.3 further improves across the board. We observe that these gains even surpass the best score from adapting MISTRAL-7B-INSTRUCT-V0.3 with *all* the SVF vectors in the ARC-Challenge task reported in Table 2. While these results appear promising, we note that the surprising compatibility discovered through our naive transfer approach is potentially tied to the similarity between the architectures of the two considered LLMs. To this end, whether similar transfer can be replicated with models of different scales remains an open research question that could open the doors to disentangling and recycling task-specific skills for models, with important implications for the democratization and sustainability of the field.

Table 3: **Cross-model z vector transfer.** Results from transferring the expert vectors trained on LLAMA3-8B-INSTRUCT to MISTRAL-7B-INSTRUCT-V0.3 with cross model few-shot adaptation.

Method *SVF training task*	**MATH** *GSM8K*	**Humaneval** *MBPP-pro*	**ARC-Challenge** *ARC-Easy*
MISTRAL-7B-INSTRUCT-V0.3	**13.02 (1.00)**	43.29 (1.00)	71.76 (1.00)
+ Llama SVF (ordered σ_i)	11.96 (0.92)	45.12 (1.04)	72.01 (1.00)
+ Llama SVF (shuffled σ_i)	10.52 (0.81)	40.24 (0.93)	70.82 (0.99)
+ Few-shot adaptation (cross-model)	12.65 (0.97)	**46.75 (1.08)**	**75.64 (1.05)**

5 Conclusion

We introduced Transformer^2, a novel blueprint toward realizing self-adaptive LLMs. We first proposed SVF, offering superior performance, reduced costs, high compositionality, and overfitting regularization. Building on SVF experts, we developed three effective self-adaptation strategies, each providing unique benefits and scalable performance improvements. Future work could explore cross-model compatibility and expert skill recycling across model generations, drawing from recent model merging techniques (Yu et al., 2024; Goddard et al., 2024; Akiba et al., 2024) to overcome individual LLM limitations.

References

Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. Evolutionary optimization of model merging recipes. *arXiv preprint arXiv:2403.13187*, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*, 2024.

Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Alberto Cetoli. Fine-tuning llms with singular value decomposition. Hugging Face Blog, June 2024. URL `https://huggingface.co/blog/fractalego/svd-training`. Accessed: 2024-07-01.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

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

Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vlad Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. Arcee's mergekit: A toolkit for merging large language models. *arXiv preprint arXiv:2403.13257*, 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. *arXiv preprint arXiv:2103.03874*, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Verena Klös, Thomas Göthel, and Sabine Glesner. Adaptive knowledge bases in self-adaptive system design. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, pp. 472–478, 2015. doi: 10.1109/SEAA.2015.48.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*, 2023.

Vijay Lingam, Atula Tejaswi, Aditya Vavre, Aneesh Shetty, Gautham Krishna Gudur, Joydeep Ghosh, Alex Dimakis, Eunsol Choi, Aleksandar Bojchevski, and Sujay Sanghavi. Svft: Parameter-efficient fine-tuning with singular vectors. *arXiv preprint arXiv:2405.19597*, 2024.

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. *arXiv preprint arXiv:2402.09353*, 2024.

Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Ok-vqa: A visual question answering benchmark requiring external knowledge. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pp. 3195–3204, 2019.

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.

Qwen Team. Qwen1.5-moe: Matching 7b model performance with 1/3 activated parameters, March 2024. URL `https://qwenlm.github.io/blog/qwen-moe/`. Blog post.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pp. 18332–18346. PMLR, 2022.

Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*, volume 133. Springer, 2004.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards vqa models that can read. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8317–8326, 2019.

Chen Tianlong, Cheng Yu, Chen Beidi, Zhang Minjia, and Bansal Mohit. Mixture-of-experts in the era of llms: A new odyssey. ICML 2024 presentation slides, 2024. International Conference on Machine Learning (ICML).

Hanqing Wang, Zeguan Xiao, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. Milora: Harnessing minor singular components for parameter-efficient llm finetuning. *arXiv preprint arXiv:2406.09044*, 2024.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024.

Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, et al. Proagent: building proactive cooperative agents with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17591–17599, 2024.

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

Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. Llama-moe: Building mixture-of-experts from llama with continual pre-training. *arXiv preprint arXiv:2406.16554*, 2024.

Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, et al. Mindstorms in natural language-based societies of mind. *arXiv preprint arXiv:2305.17066*, 2023.

A Implementation details and hyper-parameters

A.1 SVF training

We obtain the expert vectors z as the base components in Transformer^2 by training the SVF fine-tunes with a consistent recipe across the considered training tasks and language models. We divide each dataset to produce equal-sized training and validation splits. We then apply our RL-based approach, optimizing θ_z with AdamW using a learning rate of 2×10^{-3} with cosine decay, a batch size of 256, and gradient clipping.

We employ early stopping and select the best λ (the coefficient of the KL divergence term) based on validation performance. For the LLAMA3-70B-INSTRUCT and Vision tasks experiments, we apply the SVF on half of the layers to reduce memory usage while maintaining considerable performance improvement. During the training of LLAMA3-8B-INSTRUCT on the vision language tasks, we apply a small negative reward (-0.1) for training stability.

Our system prompt used for training classification expert is in Figure 4

```
Analyze the given question and classify it into one of four categories:
'code', 'math', 'reasoning', or 'others'. Follow these guidelines:

1. Code: Questions asking for programming solutions...
2. Math: Questions involving mathematical calculations...
3. Reasoning: Questions requiring logical thinking....
4. Others: Questions not clearly fit into above categories...

Instructions:
- Consider the primary focus, skills, and knowledge required to answer
the question.
- If a question spans multiple categories, choose the most dominant one.
- Provide your final classification within \\boxed{} notation. Example: \
\boxed{reasoning}

Format your response as follows:
Classification: \\boxed{category}
```

Figure 4: **Prompt based adaptation.** Self-adaptation prompt used by Transformer^2 to classify the task prompt into pre-defined categories.

A.2 LoRA training

We follow community best practices for LoRA fine-tuning, applying it to query and value projection layers with learning rates around 5×10^{-5}. We set 200 total iterations with a 256 global batch size for sufficient training. For feasible LoRA instruction training, we collect solutions for all training tasks (GSM8K, MBPP, Arc-Easy, TextVQA) from official sources and append them to question prompts. Table 5 shows a sample math problem used for LoRA fine-tuning. Despite extensive hyperparameter tuning, we often observe test performance decay as discussed, which can be attributed to the small number of training samples and potential model requirements for instruction fine-tuning data (specifically, the highly detailed thinking process).

```
Below is an instruction that describes a task. Write a response that
appropriately completes the request.

Natalia sold clips to 48 of her friends in April, and then she sold half
as many clips in May. How many clips did Natalia sell altogether in
April and May?

Natalia sold 48/2 = <<48/2=24>>24 clips in May. Natalia sold 48+24
= <<48+24=72>>72 clips altogether in April and May. #### 72
```

Figure 5: **Sample problem and answer.** Math data sample used for LoRA instruction fine-tuning, text in blue is the unmasked solution.

A.3 Hyper parameters

We present a summary of the hyperparameters used in our experiments in Table 4. To optimize performance, we conducted sweeps across several hyperparameters and selected the most effective combination based on validation results. For SVF, our primary focus was on adjusting the KL coefficient to enhance training stability. In the case of LoRA, we concentrated on sweeping the learning rate and maximum gradient clip norm to identify optimal settings.

A.4 Few-shot adaptation

As described in the main text, our few-shot adaptation approach entails producing an entirely new $z' = \sum_{k=1}^{K} \alpha_k z_k$ for each W by linearly interpolating between the K learned SVF vectors, each weighted by the coefficients $\alpha \in \mathbb{R}^K$. We employ CEM to search for α_k's based on the performance on the few-shot prompts, which are specifically held out from the rest of the test prompts and used to obtain the elite set at each iteration. In the case of multiple sample solutions obtaining the same

Table 4: **Hyper-parameters used for SVF and LoRA training.** We perform a sweep on certain sensitive hyper-parameters across methods for fair comparison.

SVF Hyperparameters	
Initial mean value of z	0.1
Initial variance value of z	1×10^{-3}
Global batch size	256
Learning rate	2×10^{-3}
Clip max norm	1×10^{-3}
KL coefficient λ	0.0, 0.1, 0.2, 0.3
LoRA Hyperparameters	
Rank	16
LoRA alpha	32
LoRA dropout	0.05
Global batch size	256
Learning rate	1×10^{-5}, 5×10^{-5}, 1×10^{-4}
Clip max norm	1×10^{-3}, 1.0

score on these held-out samples, we break ties by choosing the sample solution with the highest average log-likelihood across the tokens of its generated correct answers.

In all of our main experiments we reserve only 10 samples of data for self-adaptation and perform up to 100 CEM iterations. For each setting, we consider both per-layer and per-vector adaptation, where the latter strategy has the advantage of greatly simplifying search (as we only have 3 α coefficients). Moreover, we experiment with both normalizing across the α of different tasks (such that their sum would be fixed to 1) or keeping them unconstrained. Due to the lack of a validation set, we simply report the performance attained by our best sample from these test configurations at the end of optimization, on the remaining unseen samples for each task.

We provide our python pseudo code of one CEM step as follows:

```
def cem_step(mu, sigma, num_elites, num_samples):
    samples = np.random.normal(loc=mean, scale=sigma, size=num_samples)
    scores = evaluate(samples)
    elites = samples[np.argsort(scores)[-num_elites:]]
    new_mu = np.mean(elites, axis=0)
    new_sigma = np.std(elites, axis=0)
    return (new_mu, new_sigma)
```

B Additional results

B.1 Ablation studies

Module sensitivity: We first compare the performance of SVF when it is applied to different modules (see trials 1-3). Under consistent conditions, both individual MLP and attention updates improve performance, with MLP updates resulting in more pronounced gains. Simultaneous updates to both module types yield even more significant enhancements.

Objective function: We are interested in the performance impact from different objective functions, and we compare the RL objective with next-token prediction loss (see trials 2 and 4). For the latter, we use instruction fine-tuning with official GSM8K solutions as target tokens. Results show clear performance gains with RL, demonstrating its effectiveness in task-specific fine-tuning. Conversely, next-token prediction even hinders performance. This highlights RL's ability to handle cases lacking detailed solutions, suggesting its superiority in this context.

SVF vs LoRA: Finally, we also evaluate LoRA using the RL objective (see trials 2 and 5). A significant performance disparity is observed, primarily attributed to the severe instability of the LoRA

training process. Despite exploring a wide range of learning rates, LoRA's performance consistently lagged behind. For further illustrations, see Figure 6 in the appendix.

Table 5: **Ablation studies.** We fine-tune LLAMA3-8B-INSTRUCT on the GSM8K training split with different settings and the results on the test split along with zero-shot transfer results on MATH.

#	Method	Objective Function	Module	#Params (↓)	GSM8K (↑)	MATH (↑)
0	LLAMA-3-8B-INSTRUCT				75.89 (1.00)	24.54 (1.00)
1	SVF	Policy gradient	MLP	0.39M	78.62 (1.04)	24.20 (0.99)
2	SVF	Policy gradient	attention	**0.16M**	76.19 (1.00)	24.20 (0.99)
3	SVF	Policy gradient	MLP + attention	0.58M	**79.23 (1.04)**	**25.04 (1.04)**
4	SVF	Next token pred	attention	**0.16M**	60.50 (0.80)	18.52 (0.75)
5	LoRA	Policy gradient	attention	6.82M	57.92 (0.76)	15.72 (0.64)

B.2 Impact from number of few-shots

We investigate the relationship between the number of samples available for few-shot adaptation and downstream performance. Our analysis focused on the test task where LLAMA3-8B-INSTRUCT demonstrates the highest baseline performance, to prevent the potential for a null signal in our CEM-based search.

As Table 6 shows, substantial benefits of our few-shot strategy are evident with as few as 3 to 5 test samples. Moreover, performance appears to plateau beyond 10 samples, underscoring how our essential and inherently regularized SVF parameterization effectively complements self-adaptation. This efficiency enables optimal use of data to enhance understanding of the test task.

Table 6: **Few-shot adaptation scaling.** Performance varies with number of examples.

Method	ARC-Challenge
LLAMA3-8B-INSTRUCT	80.63 (1.00)
+ 3-shot adaptation	82.18 (1.02)
+ 5-shot adaptation	82.38 (1.02)
+ 10-shot adaptation	**82.61 (1.02)**
+ 20-shot adaptation	**82.61 (1.02)**

B.3 Cross-model svf transfer on the training tasks

We provide complementary results to Table 3 in the main text, where we analyze the SVF cross-model transfer performance from training on GSM8K, MBPP-pro, and ARC-Easy to our considered test tasks. In Table 7, we show the results in the same transfer setting this time evaluating MISTRAL-7B-INSTRUCT-V0.3 on the same training tasks where the LLAMA3-8B-INSTRUCT SVF vectors were obtained from. Overall, we recognize a similar trend, albeit with less consistent improvement from the original model (only in 1 out of 3 tasks), but still much higher performance than the randomly shuffled baseline. These results further confirm that the canonical ordering of the SVF parameterization is key for cross-model transfer, highlighting once more its inherent suitability to empower self-adaptation.

Table 7: **Cross-model z Vector Transfer.** Results from transfering the SVF expert vectors trained on LLAMA3-8B-INSTRUCT to MISTRAL-7B-INSTRUCT-V0.3 in the respective training tasks.

Method	GSM8K	MBPP-pro	ARC-Easy
MISTRAL-7B-INSTRUCT-V0.3	**42.83 (1.00)**	**49.50 (1.00)**	81.65 (1.00)
+ Llama SVF (ordered σ_i)	42.61 (0.99)	48.48 (0.98)	**81.78 (1.00)**
+ Llama SVF (shuffled σ_i)	41.93 (0.98)	46.34 (0.94)	80.81 (0.99)

B.4 Training curve of LoRA and policy gradient

Figure 6 gives the learning curves for LoRA training on the GSM8K task.

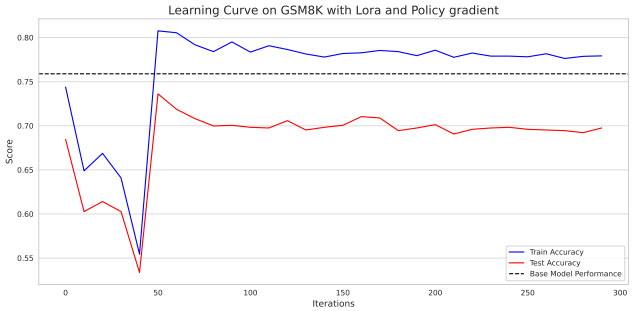

Figure 6: **Training LoRA with policy gradient.** The dashed line shows the performance of LLAMA3-8B-INSTRUCT on the test split. LoRA collapses at the beginning of the training stage and fails to recover, leading to negative effects on test performance. We swept a wide range of learning rates $(2 \times 10^{-4}, 5 \times 10^{-4}, \ldots, 2 \times 10{-}2, 5 \times 10^{-2})$, and all learning curves were similar to the one presented.

C PCA on llama3 and mistral

To investigate if the singular components that have the highest singular values are able to capture most of the information of a weight matrix, we conducted Principle Component Analysis (PCA) on the weight matrices in LLAMA3-8B-INSTRUCT and MISTRAL-7B-INSTRUCT-V0.3 (see Figures 7 and 8). In each figure, we plot the variance that is captured by the top r components across all the layers in each type of modules for a weight matrix $W \in \mathbb{R}^{m \times n}$:

$$\text{ratio} = \frac{\sum_{i=1}^{r} \sigma_i}{\sum_{j=1}^{\min(m,n)} \sigma_j}$$

Here, σ's are the ordered (from largest to smallest) singular values on the weight matrix W. It is easy to see from the figures that when $r = 256$, less than 50% of the variance is captured by these top components on average. For the MLP layers, this fraction is even lower than 20%. On the other hand, the ranks adopted by LoRA-XS or similar methods are much less than 256, resulting in even more information loss and restrictions in their modeling power that relies mostly on these r components.

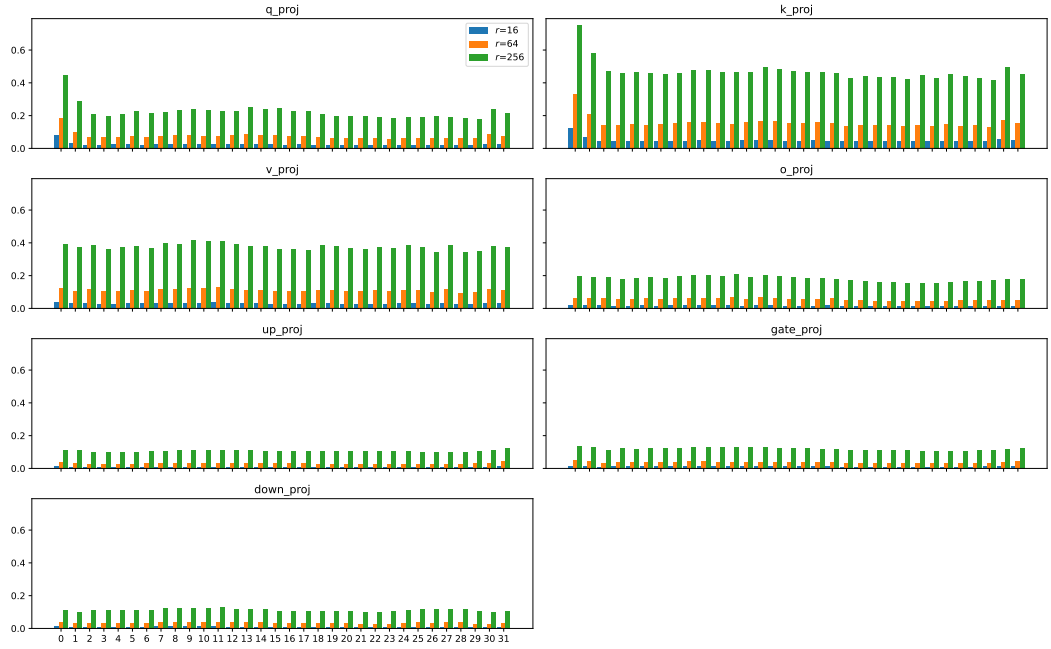

Figure 7: **PCA of LLAMA3-8B-INSTRUCT.** We show the ratio of the variance captured by the top r singular components on the y-axis, and the layer indices on the x-axis. Except for the Query, Key and Value projection matrices, small r values only capture a tiny fraction of variance in singular values in the parameter matrices.

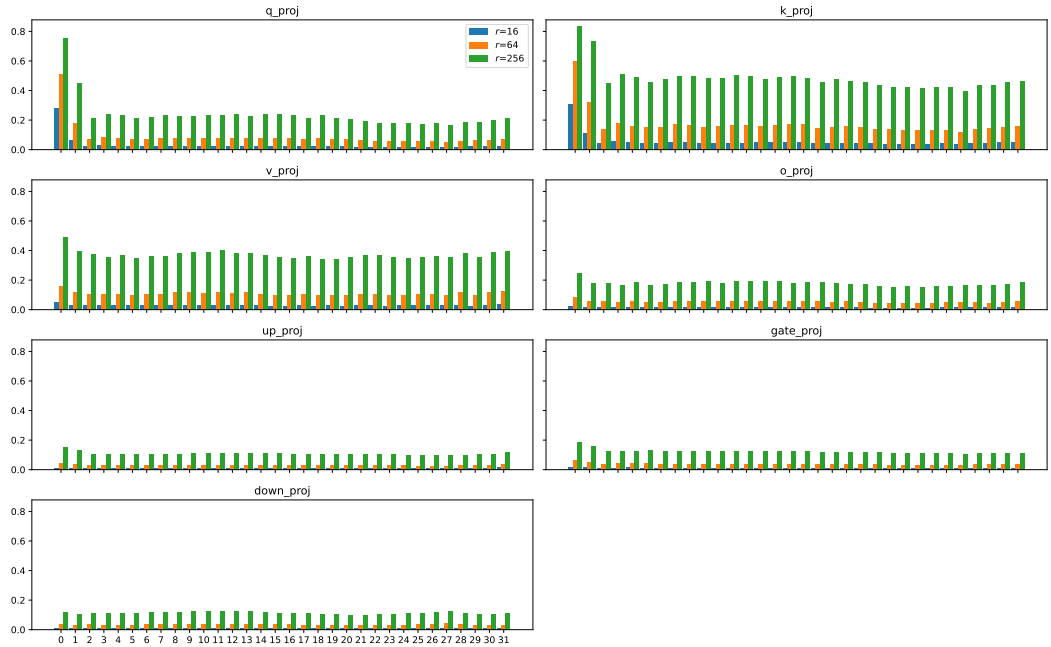

Figure 8: **PCA of MISTRAL-7B-INSTRUCT-V0.3.** We show the ratio of the variance captured by the top r singular components on the y-axis, and the layer indices on the x-axis. Except for the Query, Key and Value projection matrices, small r values only capture a tiny fraction of variance in singular values in the parameter matrices.

D Extended Related works

Self-adaptive LLMs We define self-adaptive LLMs as a group of LLMs or a standalone LLM that can evaluate and modify its behavior in response to changes in its operating environment or internal state, without external intervention. This adaptation can be explored from two perspectives:

a macroview, where multiple LLMs collaborate and/or compete, and a microview, where internal adaptations allow a single LLM to specialize in different tasks.

Macroview: From this perspective, the system directs queries to LLMs with domain specific expertise, prioritizing outputs from expert models, thereby achieving higher accuracy and task-specific optimization. Such task-specific ensembles can be realized through various mechanisms: multiple LLMs playing distinct roles and coordinate toward a shared goal (Zhuge et al., 2023), engaging in mutual listening and debate (Du et al., 2023), or using meticulously crafted prompt constructions (Zhang et al., 2024) to integrate knowledge library and skill planning. Naturally, the improvement in the specialization and adaptive capabilities of individual LLMs in the ensemble enhances the collective performance. Thus, in this paper, we focus on the microview of self-adaptive LLMs.

Microview: MoE in LLMs plays a critical role in this perspective (Tianlong et al., 2024). In MoE systems, inputs are dynamically routed to a subset of specialized modules or layers (e.g., MLPs) containing domain-specific knowledge (Rajbhandari et al., 2022; Fedus et al., 2022). To reduce inference time, researchers introduce sparsely activated MoE where only a subset of the experts are selected per token Jiang et al. (2024); Qwen Team (2024). While it is possible to view Transformer^2 loosely as a type of MoE, there are two major differences. In the aforementioned systems, self-adaptation is achieved through token-level routing, whereas Transformer^2 employs a sample-level module selection strategy. The second difference lies in the construction of expert modules. In traditional MoE systems, expert modules are either trained from scratch (Fedus et al., 2022; Jiang et al., 2024) or dense models (e.g., upcycling) (Qwen Team, 2024; Zhu et al., 2024), without an auxiliary loss to ensure module specialization. In contrast, Transformer^2 specifically trains expert vectors with RL to acquire domain specific-knowledge, making them true experts.

Low-rank adaptation PEFT methods such as LoRA (Hu et al., 2021) works by freezing the original model's parameters and introducing small trainable low-rank matrices for task-specific updates. It significantly lowers the computational and memory costs while providing performance comparable to full fine-tuning. Inspired by LoRA's design, various modifications have been proposed (Zhang et al., 2023; Kopiczko et al., 2023; Liu et al., 2024; Bałazy et al., 2024; Cetoli, 2024). Transformer^2 does not rely on low-rank matrices, and instead scales the singular vectors of the original parameter matrix that span the full rank space.

SVD for LLM Fine-tuning SVD is increasingly being used as an inductive bias for PEFT in LLMs. For example, Wang et al. (2024) decompose a weight matrix and use the minor singular components, associated with noisy or long-tail information, to initialize low-rank matrices for LoRA fine-tuning. In a similar vein, SVD is employed to approximate an original weight matrix with the top r singular vectors, corresponding to the highest singular values. A small trainable matrix is then introduced on top of the truncated singular value matrix to adjust the magnitude and orientations within this top-r subspace (Bałazy et al., 2024; Cetoli, 2024). However, the drawback of this approach is that retaining only the top singular components can result in the loss of important information, particularly when the singular values distribution is less skewed. The work most similar to ours is a concurrent effort by Lingam et al. (2024), where they introduce various sparsification methods that utilize the SVD of the weights. However, it is not for self-adaptive LLMs and does not use RL to enhance learning efficiency.