CAO-LLM: CATCHING, ADAPTING AND OPERATING UNDER DISTRIBUTION DRIFT FOR LARGE LANGUAGE MODELS

Nitin Vetcha[1,2]
[1]National University of Singapore, [2]Indian Institute of Science
ophv118@nus.edu.sg,nitinvetcha@iisc.ac.in

ABSTRACT

Large language models deployed in real-world environments inevitably encounter distribution drift like temporal shifts in data characteristics, emerging domains and evolving user requirements. Existing adaptation methods either ignore drift entirely, react post-hoc without anticipation or employ coupled optimization that fails to produce drift-specific responses. We propose **CAO-LLM**, a unified three-stage framework that **C**atches drift through representation-based monitoring, **A**dapts via calibrated parameter alignment with forgetting prevention and **O**perates at scale using test-time strategy selection. By temporally separating these complementary objectives, CAO-LLM avoids the interference that plagues joint optimization approaches. Experiments on Qwen2.5 models across 12 benchmarks spanning common-sense, coding, logic, social, medical and mathematical reasoning domains, demonstrate that CAO-LLM outperforms reactive and amortized adaptation baselines, achieving consistent gains across model sizes. With controlled behavioral analysis experiments and ablation studies on the full pipeline, we validate how and why, all three stages are essential for robust operation under drift.

1 INTRODUCTION

Large language models (LLMs) have achieved remarkable success as general-purpose systems (Achiam et al., 2023; Yang et al., 2024; Dubey et al., 2024), yet their deployment in dynamic environments exposes a fundamental vulnerability: **distribution drift**. When data characteristics shift - whether through temporal evolution, domain transfer, or concept drift - models trained on static distributions exhibit systematic performance degradation (Yue & Klein, 2025; Hager et al., 2024; Fei et al., 2024). Current approaches to handling drift span a spectrum of adaptation strategies, each with characteristic limitations:

- **Reactive Fine-tuning**: Per-task LoRA adaptation (Hu et al., 2022a) responds only *after* drift occurs, without anticipatory mechanisms.
- **Static Pooling**: Training on aggregated data assumes a stationary joint distribution, failing when drift introduces non-stationary dynamics.
- **Amortized Adaptation**: Meta-learning approaches (Finn et al., 2017; Sinha et al., 2024; Zhang et al., 2025) attempt rapid response through learned initializations, but coupled inner-outer optimization produces generic rather than drift-specific adaptations.
- **Self-Evolutionary Learning**: Methods like SeRL (Fang et al., 2025) use self-play and majority-voting to bootstrap training with limited data, but may struggle when model foundational expertise is insufficient across diverse heterogeneous domains.

We identify that robust operation under drift requires **three distinct capabilities** that must be addressed sequentially rather than jointly: (1) *detecting* distributional change before it impacts downstream performance, (2) *responding* with calibrated adaptations that preserve prior knowledge and (3) *operating* efficiently across heterogeneous data streams at scale.

CAO Theme	CAO-LLM Component	Mechanism
Sensing the Drift	**Catching** (Stage 1)	Representation-based monitoring via prompt encodings
Responding to Drift	**Adapting** (Stage 2)	VAE alignment with KL regularization
Operating at Scale	**Operating** (Stage 3)	RL-based test-time strategy selection

Table 1: **Workshop Alignment**: CAO-LLM directly addresses the three themes of the CAO workshop through temporally separated stages.

We propose **CAO-LLM**, a framework that algorithmically separates these capabilities into three temporally independent stages (Table 1):

1. **Catching**: A hyper-convolutional parameter generator monitors incoming task representations, serving as an early-warning system for drift detection.
2. **Adapting**: A task-aware conditional VAE aligns detected drift to the model's parameter space while KL regularization prevents catastrophic forgetting.
3. **Operating**: An RL-based policy selects from a family of test-time adaptation strategies (ex: Test-Time Learning, LoRA Subspace mixing) for scalable operation across diverse domains.

2 METHODOLOGY

2.1 PROBLEM SETTING

Let $\mathcal{D}_{\text{source}}$ denote the source distribution and $\mathcal{D}_{\text{target}}$ a shifted target distribution representing drift. Given a pretrained LLM with frozen parameters θ_0, we adapt via LoRA (Hu et al., 2022a): $W = W_0 + BA^\top$ where $B \in \mathbb{R}^{d_{\text{out}} \times r}$, $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $r \ll \min(d_{\text{in}}, d_{\text{out}})$. The goal is to produce adapted parameters θ^a_* that maintain performance on $\mathcal{D}_{\text{source}}$ (avoiding catastrophic forgetting) while generalizing to $\mathcal{D}_{\text{target}}$ (responding to drift).

2.2 CATCHING: DRIFT DETECTION VIA REPRESENTATION MONITORING

The Catching stage establishes proactive drift sensing through representation-based monitoring. Given representative prompts $\{p_1, \ldots, p_N\}$ characterizing a task distribution, we encode them via a frozen text encoder into a conditioning tensor $\mathbf{C}$. A **hyper-convolutional decoder** $\mathcal{G}_\phi$ transforms this representation into LoRA parameters, $\theta^a_{\text{gen}} = \mathcal{G}_\phi(\mathbf{C})$, trained via supervised regression against task-specific checkpoints:

$$\phi^* = \arg\min_\phi \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[\left\| \theta^a_{\text{gen}}(\phi; \mathbf{C}_\mathcal{T}) - \theta^a_{\text{gt}} \right\|^2 \right]. \tag{1}$$

The learned representation $\mathbf{C}$ serves as an **early-warning signal**: semantic shifts in incoming prompts produce distinct conditioning patterns before the LLM encounters the actual data, enabling anticipatory rather than reactive adaptation. Full details are provided in Appendix E.

2.3 ADAPTING: CALIBRATED RESPONSE WITH FORGETTING PREVENTION

Generated parameters may be misaligned with target task semantics, particularly under significant drift. The Adapting stage provides calibrated response through a **task-aware conditional VAE**. We extract a task vector capturing drift direction: $\mathbf{v}_{\mathcal{T}_*} = \mathbf{z}^{\text{ft}}_{\mathcal{T}_*} - \mathbf{z}^{(0)}$, where $\mathbf{z}^{(0)}$ and $\mathbf{z}^{\text{ft}}$ are last-layer representations of pretrained and fine-tuned models respectively (Ilharco et al., 2023). The VAE encoder models:

$$q_\phi(\mathbf{z} \mid \theta^a_{\text{gen}}, \mathbf{v}_{\mathcal{T}_*}) = \mathcal{N}\left(\mu_\phi([\theta^a_{\text{gen}}; \mathbf{v}]), \sigma^2_\phi([\theta^a_{\text{gen}}; \mathbf{v}])\mathbf{I}\right), \tag{2}$$

and the decoder produces aligned parameters: $\theta^a_{\text{aligned}} = \text{Dec}_\psi(\mathbf{z}, \mathbf{v}_{\mathcal{T}_*})$. The alignment objective incorporates **KL regularization** to prevent catastrophic forgetting:

$$\mathcal{L}_{\text{adapt}} = \mathbb{E}_{q_\phi}\left[\left\| \theta^a_{\text{gen}} - \theta^a_{\text{aligned}} \right\|^2 \right] + \lambda_{\text{KL}} D_{\text{KL}}\left(q_\phi(\mathbf{z} \mid \cdot) \| \mathcal{N}(0, \mathbf{I})\right). \tag{3}$$

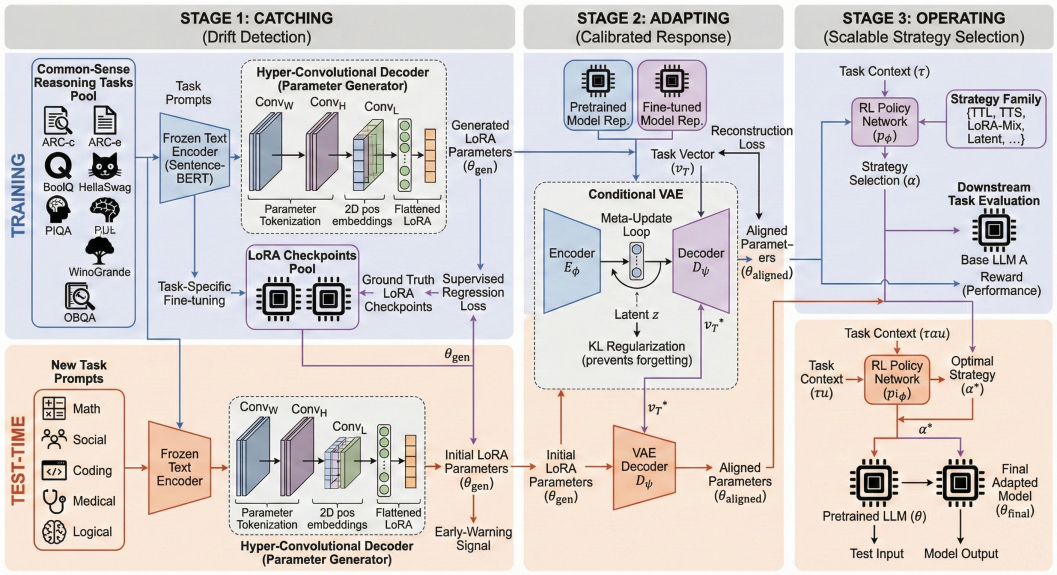

Figure 1: Multi-Stage Architecture of the Proposed CAO-LLM Framework

The KL term constrains the latent space to remain smooth and well-structured, preventing overreaction to drift that would impair consolidated knowledge. Full details are provided in Appendix F.

2.4 Operating: Scalable Test-Time Strategy Selection

Real-world deployment requires handling heterogeneous data streams with diverse drift characteristics. The Operating stage enables **scalable adaptation** through RL-based strategy selection. Given aligned parameters $\theta^a_{\text{aligned}}$, we define a family of adaptation strategies:

$$\mathcal{A} = \{\texttt{TTL}, \texttt{TTS}, \texttt{LoRA-Mix}, \texttt{Latent}, \ldots\}, \tag{4}$$

where each $\alpha \in \mathcal{A}$ specifies a bounded test-time transformation. A learned policy π_ϕ selects strategies based on task context τ:

$$\alpha^* = \arg\max_{\alpha \in \mathcal{A}} \mathbb{E}_{\alpha \sim \pi_\phi(\cdot|\tau)}[R(\tau, \alpha)], \tag{5}$$

where R evaluates adaptation utility through downstream task performance. This RL-based selection enables **monitoring-driven operation**: different drift types (covariate shift, concept drift, domain transfer) receive appropriate adaptation strategies without manual intervention. Full details are provided in Appendix G.

3 Experiments

3.1 Experimental Setup

Models. We evaluate Qwen2.5-Instruct at 0.5B and 1.5B parameters (Yang et al., 2024). Base parameters are frozen; adaptation is via LoRA.

Datasets. Consolidation (Catching stage) uses 6 common-sense reasoning benchmarks: ARC-Challenge, ARC-Easy, HellaSwag, BoolQ, PIQA, WinoGrande. We evaluate on these in-domain tasks plus 6 out-of-distribution tasks spanning mathematics (GSM-8K, MATH), logic (DivLogicEval), social reasoning (SocialIQA), coding (CodeMMLU) and medical QA (JAMA).

Baselines. We compare against methods representing the adaptation spectrum: (a) **Base Model**: No adaptation (drift ignored), (b) **No Meta-Train LoRA**: Per-task reactive fine-tuning, (c) **Union Train LoRA**: Static pooling across domains, (d) **SeRL** Fang et al. (2025): Self-Evolutionary Learning and

In-Domain Tasks (Consolidated Prior)							
Dataset	ARC-c	ARC-e	HellaSwag	BoolQ	PIQA	WinoGrande	**Avg**
Qwen2.5-1.5B-Instruct							
Base Model	71.5	83.0	50.9	56.3	45.8	50.6	59.6
No Meta-Train	**74.5**	84.4	55.8	55.6	65.6	48.2	64.0
Union Train	63.2	73.9	48.9	55.1	47.8	**61.3**	58.3
ABMLL	69.9	83.2	51.1	**63.2**	54.3	52.9	62.4
MAML-en-LLM	66.0	84.3	**59.3**	58.7	68.1	56.8	65.5
CAO-LLM	73.7	**88.4**	57.2	58.8	**70.7**	57.3	**67.7**
Qwen2.5-0.5B-Instruct							
Base Model	38.3	54.8	26.5	37.0	16.6	50.2	37.2
No Meta-Train	40.7	59.4	23.4	22.1	66.2	35.7	41.2
Union Train	39.7	47.4	26.3	14.7	51.1	50.5	38.3
SeRL	22.5	24.0	25.1	44.7	**96.7**	49.6	43.7
ABMLL	37.6	54.4	26.5	**62.2**	37.6	34.5	42.1
MAML-en-LLM	47.7	63.7	36.3	46.2	67.7	50.1	51.9
CAO-LLM	**55.5**	**74.7**	**48.3**	58.7	60.1	**52.8**	**58.4**

Test-Time Generalization under Shift (OOD)							
Dataset	GSM-8K	MATH	DivLogic	SocialIQA	CodeMMLU	JAMA	**Avg**
Qwen2.5-1.5B-Instruct							
Base Model	51.8	30.3	28.3	65.9	42.6	38.9	42.9
Union Train	34.2	32.2	24.1	51.4	34.7	34.7	36.1
ABMLL	28.7	15.9	26.9	66.3	39.6	28.5	34.3
MAML-en-LLM	35.6	43.5	31.2	68.7	42.3	32.5	42.3
CAO-LLM	**51.4**	**46.9**	**31.4**	**69.5**	**44.6**	**41.5**	**47.5**
Qwen2.5-0.5B-Instruct							
Base Model	15.2	2.8	22.4	50.8	32.4	23.8	24.5
Union Train	15.6	6.8	20.3	39.5	29.8	29.9	29.9
SeRL	23.9	24.5	27.4	32.9	26.3	12.4	24.6
ABMLL	20.4	7.1	23.7	53.1	28.2	16.8	24.9
MAML-en-LLM	29.1	**26.3**	25.1	54.9	34.1	26.4	32.6
CAO-LLM	**30.3**	24.7	**28.7**	**55.1**	**35.6**	**31.2**	**34.2**

Table 2: Performance comparing **CAO-LLM** against baselines on in-domain (consolidated prior) and OOD (test-time shift) benchmarks. **Bold**: best; underline: second best.

(e) **ABMLL** (Zhang et al., 2025), **MAML-en-LLM** (Sinha et al., 2024): Amortized or Meta-Learning based Adaptation.

Results Table 2 presents the main results and demonstrates that CAO-LLM is able to detect and address distribution drift shift across a wide variety of heterogeneous domains. Additionally, it consistently outperforms other considered adaptation baselines, across the spectrum as well. Ablation studies (Table 6, Figures 6 and 7) have also been conducted to understand the incremental contribution of each stage to the base model. Through several controlled behavourial analysis experiments conducted on LoRA weights sparsity, internal layer activations and downstream gradient analysis, we demonstrate the leverage of CAO-LLM across baselines and how it translates to genuine task-specific adaptation signatures. For more details, refer to Appendix B.

4 Conclusion

We presented CAO-LLM, a unified framework addressing the three challenges of operating LLMs under distribution drift: **Catching** drift through representation-based monitoring, **Adapting** via calibrated VAE alignment with forgetting prevention and **Operating** at scale through test-time strategy selection. The temporal separation of these objectives is the key mechanism enabling drift-specific responses that coupled approaches cannot achieve. Experiments validate that all three stages are necessary - drift detection alone can hurt performance, while the full pipeline achieves robust generalization across diverse OOD domains. Future work will explore continual adaptation with streaming drift and extend to larger model scales.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Emre Can Acikgoz, Cheng Qian, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. Self-improving llm agents at test-time, 2025. URL https://arxiv.org/abs/2510.07841.

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. *arXiv preprint arXiv:2411.07279*, 2024.

Rachit Bansal, Aston Zhang, Rishabh Tiwari, Lovish Madaan, Sai Surya Duvvuri, Devvrit Khatri, David Brandfonbrener, David Alvarez-Melis, Prajjwal Bhargava, Mihir Sanjay Kale, et al. Let's (not) just put things in context: Test-time training for long-context llms. *arXiv preprint arXiv:2512.13898*, 2025.

Alexander Bartler, Andre Bühler, Felix Wiewel, Mario Döbler, and Bin Yang. Mt3: Meta test-time training for self-supervised test-time adaption, 2022. URL https://arxiv.org/abs/2103.16201.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-lora: Instant transformer adaption, 2025. URL https://arxiv.org/abs/2506.06105.

Hanjie Chen, Zhouxiang Fang, Yash Singla, and Mark Dredze. Benchmarking large language models on answering and explaining challenging medical questions. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3563–3599, Albuquerque, New Mexico, April 2025a. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.182. URL https://aclanthology.org/2025.naacl-long.182/.

Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: Llm self-improve through co-evolution, 2025b. URL https://arxiv.org/abs/2510.23595.

Tsz Ting Chung, Lemao Liu, Mo Yu, and Dit-Yan Yeung. Divlogiceval: A framework for benchmarking logical reasoning evaluation in large language models, 2025. URL https://arxiv.org/abs/2509.15587.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

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, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI, :, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, Kaige Gao, Wenjun Gao, Ruiqi Ge, Kang Guan, Daya Guo, Jianzhong Guo, Guangbo Hao, Zhewen Hao, Ying He, Wenjie Hu, Panpan Huang, Erhang Li, Guowei Li, Jiashi Li, Yao Li, Y. K. Li, Wenfeng Liang, Fangyun Lin, A. X. Liu, Bo Liu, Wen Liu, Xiaodong Liu, Xin Liu, Yiyuan Liu, Haoyu Lu, Shanghao Lu, Fuli Luo, Shirong Ma, Xiaotao Nie, Tian Pei, Yishi Piao, Junjie Qiu, Hui Qu, Tongzheng Ren, Zehui Ren, Chong Ruan, Zhangli Sha, Zhihong Shao, Junxiao Song, Xuecheng Su, Jingxiang Sun, Yaofeng Sun, Minghui Tang, Bingxuan Wang, Peiyi Wang, Shiyu Wang, Yaohui Wang, Yongji Wang, Tong Wu, Y. Wu, Xin Xie, Zhenda Xie, Ziwei Xie, Yiliang Xiong, Hanwei Xu, R. X. Xu, Yanhong Xu, Dejian Yang, Yuxiang You, Shuiping Yu, Xingkai Yu, B. Zhang, Haowei Zhang, Lecong Zhang, Liyue Zhang, Mingchuan Zhang, Minghua Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, Qihao Zhu, and Yuheng Zou. Deepseek llm: Scaling open-source language models with longtermism, 2024. URL https://arxiv.org/abs/2401.02954.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Wenkai Fang, Shunyu Liu, Yang Zhou, et al. SeRL: Self-play reinforcement learning for large language models with limited data. In *Advances in Neural Information Processing Systems*, 2025.

Zhiwei Fei, Xiaoyu Shen, Dawei Zhu, Fengzhe Zhou, Zhuo Han, Alan Huang, Songyang Zhang, Kai Chen, Zhixin Yin, Zongwen Shen, et al. Lawbench: Benchmarking legal knowledge of large language models. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 7933–7962, 2024.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. PMLR, 2017.

Antonio Andrea Gargiulo, Donato Crisostomi, Maria Sofia Bucarelli, Simone Scardapane, Fabrizio Silvestri, and Emanuele Rodola. Task singular vectors: Reducing task interference in model merging. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 18695–18705, 2025.

Halil Alperen Gozeten, M Emrullah Ildiz, Xuechen Zhang, Mahdi Soltanolkotabi, Marco Mondelli, and Samet Oymak. Test-time training provably improves transformers as in-context learners. *arXiv preprint arXiv:2503.11842*, 2025.

Paul Hager, Friederike Jungmann, Robbie Holland, Kunal Bhagat, Inga Hubrecht, Manuel Knauer, Jakob Vielhauer, Marcus Makowski, Rickmer Braren, Georgios Kaissis, et al. Evaluation and mitigation of the limitations of large language models in clinical decision-making. *Nature medicine*, 30(9):2613–2622, 2024.

Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. Evotest: Evolutionary test-time learning for self-improving agentic systems, 2025. URL https://arxiv.org/abs/2510.13220.

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, 2021. URL https://arxiv.org/abs/2103.03874.

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. In *International Conference on Learning Representations*, 2022a.

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

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data, 2026. URL https://arxiv.org/abs/2508.05004.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations*, 2023.

Kaiyi Ji, Junjie Yang, and Yingbin Liang. Theoretical convergence of multi-step model-agnostic meta-learning. *Journal of machine learning research*, 23(29):1–41, 2022.

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, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Yulun Jiang, Liangze Jiang, Damien Teney, Michael Moor, and Maria Brbic. Meta-rl induces exploration in language agents, 2025. URL https://arxiv.org/abs/2512.16848.

Pascal Klink, Carlo D'Eramo, Jan R Peters, and Joni Pajarinen. Self-paced deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:9216–9227, 2020.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021. URL https://arxiv.org/abs/2104.08691.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021. URL https://arxiv.org/abs/2101.00190.

Zhiyuan Liang, Dongwen Tang, Yuhao Zhou, Xuanlei Zhao, Mingjia Shi, Wangbo Zhao, Zekai Li, Peihao Wang, Konstantin Schürholt, Damian Borth, Michael M. Bronstein, Yang You, Zhangyang Wang, and Kai Wang. Drag-and-drop llms: Zero-shot prompt-to-weights, 2025. URL https://arxiv.org/abs/2506.16406.

Bo Liu, Chuanyang Jin, Seungone Kim, Weizhe Yuan, Wenting Zhao, Ilia Kulikov, Xian Li, Sainbayar Sukhbaatar, Jack Lanchantin, and Jason Weston. Spice: Self-play in corpus environments improves reasoning, 2025. URL https://arxiv.org/abs/2510.24684.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022. URL https://arxiv.org/abs/2205.05638.

Dung Nguyen Manh, Thang Phan Chau, Nam Le Hai, Thong T. Doan, Nam V. Nguyen, Quang Pham, and Nghi D. Q. Bui. Codemmlu: A multi-task benchmark for assessing code understanding and reasoning capabilities of codellms, 2025. URL https://arxiv.org/abs/2410.01999.

Yulong Mao, Kaiyu Huang, Changhao Guan, Ganglin Bao, Fengran Mo, and Jinan Xu. Dora: Enhancing parameter-efficient fine-tuning with dynamic rank distribution. *arXiv preprint arXiv:2405.17357*, 2024.

Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects, 2013.

Wei Pang, Chuan Zhou, Xiao-Hua Zhou, and Xiaojie Wang. Phased instruction fine-tuning for large language models. *arXiv preprint arXiv:2406.04371*, 2024.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, August 2021. ISSN 0001-0782. doi: 10.1145/3474381. URL https://doi.org/10.1145/3474381.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions, 2019. URL https://arxiv.org/abs/1904.09728.

Yihua Shao, Xiaofeng Lin, Xinwei Long, Siyu Chen, Minxi Yan, Yang Liu, Ziyang Yan, Ao Ma, Hao Tang, and Jingcai Guo. Icm-fusion: In-context meta-optimized lora fusion for multi-task adaptation, 2025a. URL https://arxiv.org/abs/2508.04153.

Yihua Shao, Minxi Yan, Yang Liu, Siyu Chen, Wenjie Chen, Xinwei Long, Ziyang Yan, Lei Li, Chenyu Zhang, Nicu Sebe, Hao Tang, Yan Wang, Hao Zhao, Mengzhu Wang, and Jingcai Guo. In-context meta lora generation, 2025b. URL https://arxiv.org/abs/2501.17635.

Zhaofeng Si, Shu Hu, Kaiyi Ji, and Siwei Lyu. Meta-learning with heterogeneous tasks. In *International Joint Conference on Artificial Intelligence*, pp. 74–94. Springer, 2025.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models, 2024. URL https://arxiv.org/abs/2312.06585.

Deepanway Sinha, Anwesha Sahoo, Ayan Mondal, Anirban Mukherjee, and Utpal Garain. MAML-en-LLM: Model agnostic meta-training of LLMs for improved in-context learning. *arXiv preprint arXiv:2405.11446*, 2024.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts, 2020. URL https://arxiv.org/abs/1909.13231.

Zhenheng Tang, Xiang Liu, Qian Wang, Peijie Dong, Bingsheng He, Xiaowen Chu, and Bo Li. The lottery llm hypothesis, rethinking what abilities should llm compression preserve?, 2025. URL https://arxiv.org/abs/2502.17535.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pp. 35151–35174. PMLR, 2023.

Fanqi Wan, Xinting Huang, Deng Cai, Xiaojun Quan, Wei Bi, and Shuming Shi. Knowledge fusion of large language models, 2024. URL https://arxiv.org/abs/2401.10491.

Jiaxing Wang et al. Recurrent parameter generators. *arXiv preprint*, 2025.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-Merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Prateek Yadav, Tu Vu, Jonathan Lai, Alexandra Chronopoulou, Manaal Faruqui, Mohit Bansal, and Tsendsuren Munkhdalai. What matters for model merging at scale? *arXiv preprint arXiv:2410.03617*, 2024.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

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 *International Conference on Machine Learning*. PMLR, 2024.

Jonathan Yue and Daniel Klein. Benchmarking llms on advanced mathematical reasoning. Master's thesis, EECS Department, University of California, Berkeley, May 2025. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-121.html.

Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, and Michael Bendersky. Inference scaling for long-context retrieval augmented generation. *arXiv preprint arXiv:2410.04343*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Liyi Zhang, Jake Snell, and Thomas L. Griffiths. Amortized bayesian meta-learning for low-rank adaptation of large language models, 2025. URL https://arxiv.org/abs/2508.14285.

Yihua Zhang, Prashant Khanduri, Ioannis Tsaknakis, Yuguang Yao, Mingyi Hong, and Sijia Liu. An introduction to bilevel optimization: Foundations and applications in signal processing and machine learning. *IEEE Signal Processing Magazine*, 41(1):38–59, 2024.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data, 2025. URL https://arxiv.org/abs/2505.03335.

Shenghe Zheng, Hongzhi Wang, Chenyu Huang, Xiaohui Wang, Tao Chen, Jiayuan Fan, Shuyue Hu, and Peng Ye. Decouple and orthogonalize: A data-free framework for lora merging, 2025. URL https://arxiv.org/abs/2505.15875.

Yujun Zhou, Zhenwen Liang, Haolin Liu, Wenhao Yu, Kishan Panaganti, Linfeng Song, Dian Yu, Xiangliang Zhang, Haitao Mi, and Dong Yu. Evolving language models without labels: Majority drives selection, novelty promotes variation, 2025. URL https://arxiv.org/abs/2509.15194.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models, 2025. URL https://arxiv.org/abs/2506.10943.

APPENDIX NAVIGATION INDEX

A Related Work

Bi-Level Meta-Learning for LLMs: Methods like MAML (Finn et al., 2017), MAML-en-LLM (Sinha et al., 2024), and ABMLL (Zhang et al., 2025) try to resolve distribution shift by employing a nested (bi-level) optimization strategy i.e., to learn initializations enabling rapid adaptation. While theoretically elegant, these approaches couple generalization and adaptation objectives, creating optimization tensions (Rajeswaran et al., 2019; Ji et al., 2022). These tensions have been analyzed in both theoretical and survey work, which document sensitivity to task heterogeneity and optimization instability in bilevel meta-learning objectives (Si et al., 2025). CAO-LLM eliminates this coupling through temporal separation of the involved learning objectives via a sequential multi-stage framework.

Self-Play and Evolution: Recent work on self-play reinforcement learning (Huang et al., 2026; Fang et al., 2025; Zweiger et al., 2025; Liu et al., 2025; Chen et al., 2025b; He et al., 2025; Zhou et al., 2025; Jiang et al., 2025; Zhao et al., 2025; Sukhbaatar et al., 2017) enables autonomous improvement through self-generated curricula. While minimizing supervision requirements, effectiveness depends critically on autonomous curriculum generation capability (Klink et al., 2020). CAO-LLM provides more structured foundational knowledge consolidation through explicit meta-training (Finn et al., 2017; Sinha et al., 2024).

Model Fusion and Parameter Alignment: Methods for merging multiple fine-tuned models have emerged as alternatives to multi-task training Zheng et al. (2025); Wan et al. (2024). Task vector arithmetic (Ilharco et al., 2023) enables composing model capabilities through weight-space operations. TIES-MERGING (Yadav et al., 2023) addresses inter-weight conflicts through pruning and sign resolution, while DARE (Yu et al., 2024) applies sparsification before merging. Prior work identifies structural interference in task-vector merging arising from sign conflicts and parameter redundancy as a primary cause of performance degradation, and proposes geometric heuristics such as sign resolution or sparsification to mitigate these effects (Gargiulo et al., 2025; Yadav et al., 2024). CAO-LLM builds on these insights by employing a task-aware VAE to explicitly adapt generated parameters with target task semantics before operation. Few prior works also combine conditional VAEs with meta-learning for latent-space alignment of LoRA parameters Shao et al. (2025a;b).

Parameter-Efficient Fine-Tuning: LoRA (Hu et al., 2022a) and related adapter methods (Mao et al., 2024) enable efficient task-specific adaptation. Our work extends these techniques by addressing how to consolidate foundational knowledge across tasks and align it with target domains before application (Pang et al., 2024). Unlike coupled bi-level approaches (Finn et al., 2017; Zhang et al., 2024), self-play methods, or static fusion techniques, CAO-LLM's three-stage design temporally separates foundation consolidation, knowledge-task alignment, and task-specific refinement, providing the cognitive and optimization benefits of staged mastery learning (Pang et al., 2024; Mermillod et al., 2013) with explicit domain alignment.

Test-Time Training and Test-Time Learning. Test-time training was originally proposed in the context of computer vision as a general strategy for adapting models to distribution shift by updating parameters on unlabeled test inputs using auxiliary self-supervised objectives (Sun et al., 2020; Bartler et al., 2022; Acikgoz et al., 2025). Subsequent work extends this paradigm to transformers and large language models, showing that gradient-based test-time updates on in-context demonstrations can empirically and theoretically improve in-context learning, reducing sample complexity and improving robustness to distribution mismatch (Akyürek et al., 2024; Gozeten et al., 2025; Von Oswald et al., 2023). More recent test-time learning approaches focus specifically on LLMs, demonstrating that perplexity-based optimization of LoRA adapters on unlabeled target-domain inputs can yield substantial gains in domain adaptation without additional pre-training (Hu et al., 2022b). Orthogonal work studies restricted forms of test-time training for long-context inference, showing that targeted query-level updates can use inference-time compute more effectively than inference-only decoding strategies in retrieval-dominated regimes (Bansal et al., 2025; Yue et al., 2024).

B Controlled Experiments

Before presenting quantitative performance results, we first analyze the behavioral differences through controlled experiments between CAO-LLM and other baselines to see how effectively the proposed method address the issue of distribution shift, thereby resulting in task-specific adaptation.

B.1 Gradient Conflict Analysis

We analyze gradient structure across out-of-domain tasks by computing pairwise cosine similarity between task-specific gradient vectors. Higher similarity with lower variance indicates undifferentiated representations; lower similarity with higher variance suggests task-specific learning. MAML's gradients exhibit high mutual similarity (mean 0.884, std 0.047), confirming that coupled optimization produces undifferentiated representations. CAO-LLM shows lower mean (0.749) with higher variance (0.116), demonstrating that temporal separation preserves domain-specific optimization directions (Figure 2, Table 3).

Method	Mean	Std	Min	Max
Qwen2.5-0.5B-Instruct				
MAML-en-LLM	0.884	0.047	0.770	0.976
CAO-LLM (Ours)	**0.749**	**0.116**	0.476	0.940

Table 3: Pairwise gradient cosine similarity across 6 OOD tasks. **Lower mean with higher variance** indicates better task-specific differentiation.

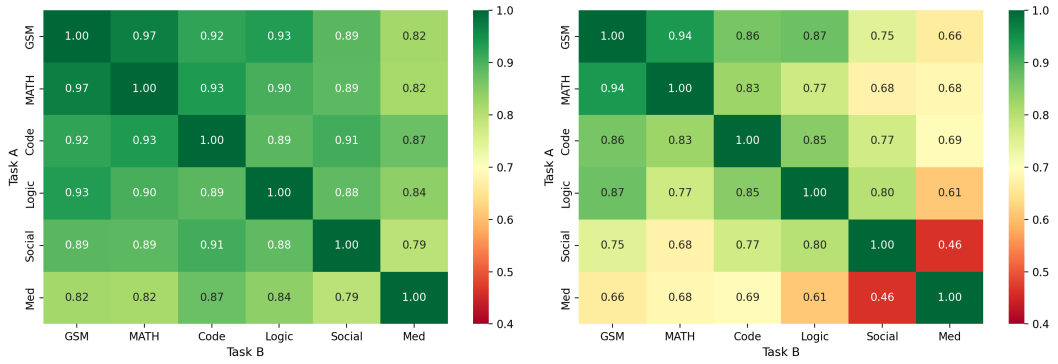

Figure 2: **Gradient Conflict Comparison.** MAML-en-LLM (left) shows uniformly high similarity; CAO-LLM (right) exhibits wider variance with mathematics tasks clustering tightly (GSM–MATH: 0.94) while distant pairs diverge (Social–Medical: 0.46).

B.2 LoRA Weight Sparsity and Structure

We analyze LoRA weight properties across six OOD tasks to characterize optimization dynamics. CAO-LLM exhibits **6.5× higher sparsity**, with 30% lower L2 and 65% lower L1 norms. This indicates selective parameter modifications: fewer parameters carry the adaptation signal with higher peak magnitudes, consistent with temporal separation enabling task-focused updates (Table 4).

Metric	MAML	CAO-LLM	Ratio
Qwen2.5-0.5B-Instruct			
Weight Sparsity	1.55%	**10.01%**	6.5× higher
L2 Norm	0.575	**0.400**	30% lower
L1 Norm	2512.1	**866.1**	65% lower
Max Weight	0.0017	**0.0031**	1.8× higher

Table 4: LoRA weight analysis across 6 OOD tasks. CAO-LLM exhibits **6.5× higher sparsity** with lower norms but higher peak magnitudes-focused adaptation rather than diffuse updates.

B.3 Visual Evidence: Adaptation Signatures

We visualize LoRA parameter updates for Layer 12 across six OOD tasks by computing $\Delta W = \frac{\alpha}{r} BA$. MAML produces "barcode-like" patterns with identical weights across tasks (Figure 4: vertical

Figure 3: **LoRA Weight Pattern Visualization across 6 OOD Tasks.** MAML-en-LLM (top) shows nearly identical weight patterns-*lines overlap*-for all tasks, indicating a generic "barcode-like" adaptation that fails to specialize. CAO-LLM (bottom) displays clearly separated "signatures" for each domain (e.g., Code vs. Medical vs. Math), proving that temporal separation unlocks genuine task-specific parameter updates.

stripes). CAO-LLM produces distinct "signatures" for each domain, visually confirming that temporal separation enables genuine task-specific specialization.

B.4 Task-Semantic Alignment Verification

We verify that Stage 2 preserves task semantics by computing pairwise cosine similarity between aligned LoRA parameters and VAE conditioning latents. Mathematics tasks exhibit 0.949 raw / 0.526 latent similarity, while JAMA (medical) shows near-zero similarity with math domains, confirming domain-appropriate separation.

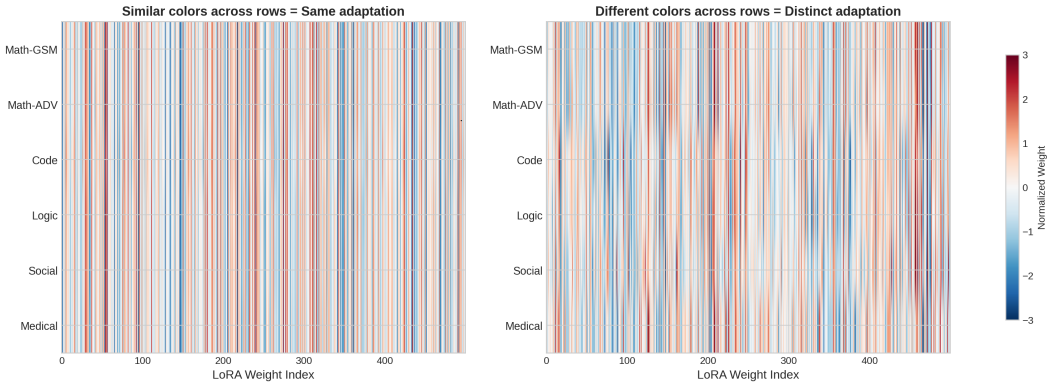

Figure 4: **LoRA Weight Heatmap.** MAML-en-LLM (left) shows uniform vertical stripes-identical parameters across tasks ("compromise"). CAO-LLM (right) shows varied patterns, demonstrating task-specific adaptations.

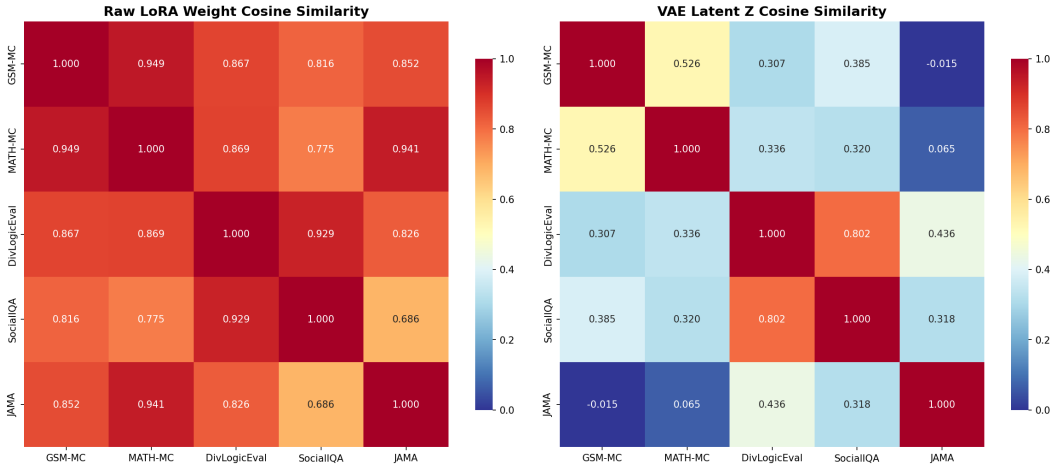

Figure 5: **Pairwise Similarity of Aligned LoRA Adapters.** Similar domains cluster (GSM–MATH: 0.949 raw, 0.526 latent) while distant domains separate (JAMA–Math: near-zero latent), confirming semantic preservation.

B.5 Task Specialization Analysis

In this ablation study, we show that providing different adaptation strategy families is necessary in order to provide flexibility. Results from Table 5 reveal that the different methods considered for adaptation in our work i.e., TTL (Test-Time Learning), TTS (Test-Time Scaling), LoRA (Two-Subspace Mixing LoRA) and Latent (Hidden Layer Output Optimization) are necessary for the optimizing the LLM performance in different downstream tasks.

In-Domain Tasks					
Dataset	ARC-c	ARC-e	HellaSwag	BoolQ	PIQA
TTL	45.8	65.9	32.1	48.6	**60.1**
TTS	49.1	68.9	**48.3**	46.4	46.7
LoRA	44.1	61.8	31.9	**58.7**	18.9
Latent	**55.5**	**74.7**	28.7	fail	51.9

Out-of-Domain Tasks					
Dataset	GSM	MATH	Logic	Social	Med
TTL	24.1	fail	26.3	32.9	**31.2**
TTS	24.3	**24.5**	22.9	**55.1**	13.4
LoRA	**30.3**	**24.5**	26.6	31.5	14.3
Latent	23.9	24.2	**28.7**	32.5	21.8

Table 5: Performance comparison of various adaptation strategy configurations on Qwen2.5-0.5B-Instruct across In-Domain and Out-of-Domain benchmarks.

B.6 Individual Stage Contribution

In-Domain Tasks							
Dataset	ARC-c	ARC-e	HellaSwag	BoolQ	PIQA	WinoGrande	**Avg**
			Qwen2.5-1.5B-Instruct				
Base Model	71.5	83.0	50.9	56.3	45.8	50.6	59.6
Detection	73.0 (+2.1%)	83.7 (+0.8%)	56.2 (+10.4%)	55.2 (-2.0%)	56.4 (+23.1%)	50.2 (-0.8%)	62.5 (+4.9%)
Adaptation	73.7 (+1.0%)	88.4 (+5.6%)	57.2 (+1.8%)	58.8 (+6.5%)	70.7 (+25.4%)	57.3 (+14.1%)	67.7 (+8.3%)
			Qwen2.5-0.5B-Instruct				
Base Model	38.3	54.8	26.5	37.0	16.6	50.2	37.2
Detection	42.7 (+11.5%)	63.2 (+15.3%)	25.9 (-2.3%)	44.9 (+21.4%)	47.6 (+186.7%)	50.0 (-0.4%)	45.7 (+22.9%)
Adaptation	55.5 (+30.0%)	74.7 (+18.2%)	48.3 (+86.5%)	58.7 (+30.7%)	60.1 (+26.3%)	52.8 (+5.6%)	58.4 (+27.8%)

Out-of-Domain Tasks							
Dataset	GSM-8K	MATH	DivLogicEval	SocialIQA	CodeMMLU	JAMA	**Avg**
			Qwen2.5-1.5B-Instruct				
Base Model	51.8	30.3	28.3	65.9	42.6	38.9	42.9
Detection	32.6 (-37.1%)	40.1 (+32.3%)	28.6 (+1.1%)	68.6 (+4.1%)	44.1 (+3.5%)	39.5 (+1.5%)	42.2 (-1.6%)
Adaptation	51.4 (+57.7%)	46.9 (+17.0%)	31.4 (+9.8%)	69.5 (+1.3%)	44.6 (+1.1%)	41.5 (+5.1%)	47.5 (+12.6%)
			Qwen2.5-0.5B-Instruct				
Base Model	15.2	2.8	22.4	50.8	32.4	23.8	24.5
Detection	20.8 (+36.8%)	24.1 (+760.7%)	21.0 (-6.3%)	33.5 (-34.1%)	29.1 (-10.2%)	11.7 (-50.8%)	25.7 (+4.9%)
Adaptation	30.3 (+45.7%)	24.5 (+1.7%)	28.7 (+36.7%)	55.1 (+64.5%)	35.6 (+22.3%)	31.2 (+166.7%)	34.2 (+33.1%)

Table 6: Ablation study analyzing the contributions of Detection (Stage 1: Catching) and adaptation (Stages 2+3: Adapting + Operating) components in CAO-LLM. **Base Model** denotes the frozen pretrained LLM without any LoRA adapters. **Detection** evaluates performance using LoRA parameters generated by the hyper-convolutional decoder $\mathcal{G}_\phi$ without alignment or refinement. **Adaptation** applies the full CAO-LLM pipeline: VAE-based alignment plus RL-based refinement to generated parameters. Percentages show relative improvement over the previous row.

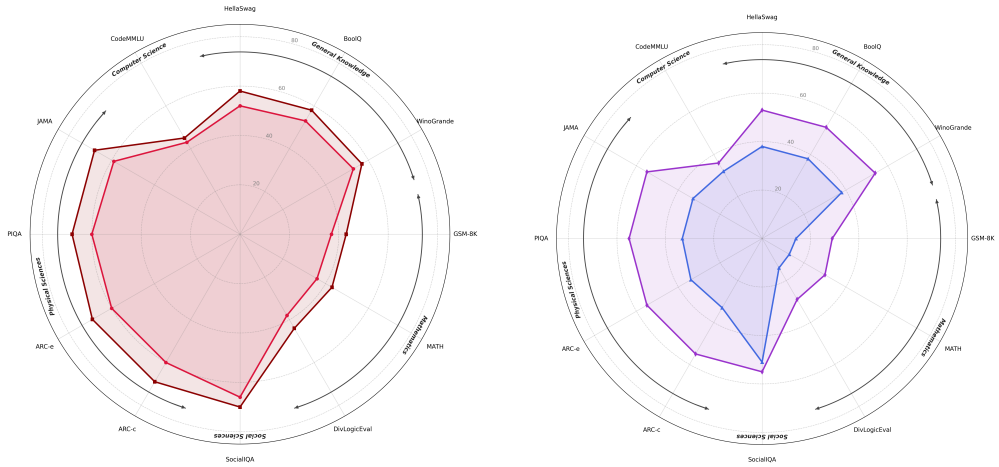

Figure 6: Radar Chart demonstrating performance impact of CAO-LLM on Base Model - (a) Qwen2.5-1.5B-Instruct *(left:* Base Model, after CAO-LLM*)* and (b) Qwen2.5-0.5B-Instruct *(right:* Base Model, after CAO-LLM*)*

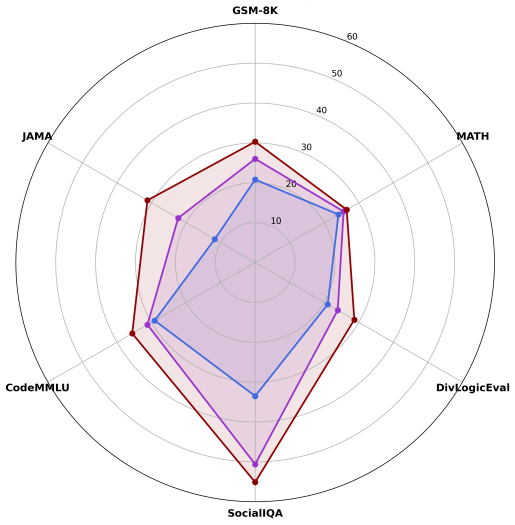

Figure 7: Ablation Study using Qwen2.5-0.5B-Instruct on out-of-domain tasks, revealing the incremental effectiveness of each CAO-LLM stage with Stage I, Stage II and Stage III

C Computational Complexity Analysis

We analyze the computational characteristics of CAO-LLM and compare it with bi-level meta-learning and self-evolution–based adaptation methods.

C.1 CAO-LLM

Stage 1: Foundation Consolidation. Foundation consolidation performs supervised meta-training over a set of training tasks $\{\mathcal{T}_i\}$ with datasets $\mathcal{D}_{\mathcal{T}_i}$. This incurs a one-time cost of

$$O\left(\sum_i |\mathcal{D}_{\mathcal{T}_i}|\right), \tag{6}$$

corresponding to standard LoRA fine-tuning runs used for checkpoint collection and parameter generator training. This cost is amortized across all future target tasks.

Stage 2: Knowledge–Task Alignment. For each target task, alignment requires a single forward pass through the conditional VAE operating on generated LoRA parameters. The computational cost scales linearly with the adapter parameter dimension D:

$$O(D). \tag{7}$$

No gradient-based optimization is performed at this stage.

Stage 3: Task-Specific Refinement. Task refinement operates at inference time using bounded update steps or policy-based strategy selection. Let K denote the number of refinement steps. The cost is

$$O(K \cdot D), \tag{8}$$

with K fixed and small in all experiments.

C.2 Comparison to Prior Paradigms

Bi-Level Meta-Learning. Methods such as MAML-en-LLM and ABMLL require executing inner-loop optimization for each target task, incurring per-task costs of

$$O(K \cdot |\mathcal{D}_{\mathcal{T}_*}|), \tag{9}$$

where K is the number of inner-loop gradient steps. This cost cannot be amortized and scales with the target dataset size.

Self-Evolution Methods. Self-play approaches such as SeRL avoid explicit meta-training but rely on iterative self-generated rollouts and on-policy updates for each target task. As a result, their per-task cost grows with the number of self-play iterations and sampled trajectories, often exceeding that of standard fine-tuning in practice.

D NOTATION AND SYMBOLS

Table 7 provides a comprehensive reference for all mathematical notation used in this paper.

E CATCHING: DRIFT DETECTION VIA REPRESENTATION MONITORING

This section provides additional implementation details for the catching stage (Stage 1) of CAO-LLM, focusing on (i) LoRA checkpoint collection and (ii) hyper-convolutional decoder architectures.

E.1 PROMPT ENCODING TENSOR STRUCTURE

In Eq. (4), prompt embeddings are constructed as

$$\begin{aligned} \mathbf{c}_i &= \text{Encoder}(p_i; \theta_{\text{enc}}), \\ \mathbf{C} &= [\mathbf{c}_1; \ldots; \mathbf{c}_N] \in \mathbb{R}^{B \times N \times L \times C} \end{aligned} \tag{10}$$

Here:

- B denotes the batch size (number of tasks processed in parallel),
- N is the number of representative prompts sampled per task,
- L is the token sequence length of each prompt,
- C is the hidden embedding dimension of the frozen text encoder.

This tensorized representation follows the prompt-to-weights formulation introduced in Drag-and-Drop LLMs (DnD) (Liang et al., 2025), and enables convolutional processing over task, prompt and token dimensions. Since B does not affect architectural design, it is omitted in subsequent architecture descriptions.

E.2 LORA CHECKPOINT COLLECTION PROTOCOL

CAO-LLM relies on a supervised prompt-to-weights mapping trained using LoRA checkpoints collected from task-specific fine-tuning runs. We adopt the checkpoint collection procedure proposed in DnD (Liang et al., 2025).

Each checkpoint collection run consists of two phases:

- **Pretraining phase**: the base model is trained on the target dataset for a fixed number of steps using a higher learning rate.
- **Fine-tuning phase**: training continues with a lower learning rate, and a LoRA checkpoint is saved at each step.

Except for learning rate and training steps, all other hyperparameters (e.g., batch size, optimizer, data sampling strategy) are kept identical between the two phases. For datasets with fewer samples than the specified number, the full dataset is used.

Table 8 summarizes the checkpoint collection settings for different task families.

E.3 PARAMETER TOKENIZATION

We employ a parameter tokenization strategy following Wang et al. (2025), which transforms LoRA adapter weights into a sequence of uniform tokens suitable for processing by the generalization model. It involves,

Symbol	Description
Spaces, Tasks & Distributions	
$\mathcal{X}, \mathcal{Y}$	Input and output spaces
$\mathcal{T}, \mathcal{T}_*$	Task; target (unseen) task
$p(\mathcal{T}), p_{\mathcal{T}}(x, y)$	Task distribution; task data distribution
$\mathcal{D}_{\mathcal{T}}$	Dataset for task $\mathcal{T}$
$\mathbb{R}, \mathbb{R}_+, \mathbb{E}$	Reals; non-negative reals; expectation
Model & Adapter Parameters	
$\theta_0, \theta, \theta^*$	Frozen pretrained; general; optimal params
$\theta^a \in \mathbb{R}^D$	LoRA adapter parameter vector
$\theta^a_{\text{gen}}, \theta^a_{\text{aligned}}, \theta^a_*$	Generated; aligned; refined adapters
$\Delta\theta, \tilde{\theta}$	Parameter update; adapted params
LoRA Parameterization	
$W = W_0 + BA^\top$	Weight matrix with LoRA adaptation
$A \in \mathbb{R}^{r \times d_{\text{in}}}, B \in \mathbb{R}^{d_{\text{out}} \times r}$	LoRA down/up projection matrices
r	LoRA rank ($r = 8$ for 0.5B; $r = 16$ for 1.5B)
$\alpha, \Delta W = \frac{\alpha}{r} BA$	Scaling factor; effective weight update
Stage 1: Catching	
$\mathcal{G}_\phi, \phi, \phi^*$	Hyper-conv generator; params; optimal
$\mathbf{C} \in \mathbb{R}^{B \times N \times L \times C}$	Prompt conditioning tensor
B, N, L, C	Batch size; #prompts; seq length; embed dim
$k, I, K[i], k_i^j, e_i^j$	Token size; #layers; tokens; positional embed
Stage 2: Adapting	
$\mathbf{v}_{\mathcal{T}_*} = \mathbf{z}^{\text{ft}} - \mathbf{z}^{(0)}$	Task vector (fine-tuned − pretrained repr.)
$q_\phi(\mathbf{z} \mid \cdot), \text{Dec}_\psi$	VAE encoder posterior; decoder
$\mu_\phi, \sigma^2_\phi, \mathbf{z}, \psi$	Posterior mean/variance; latent; decoder params
$\mathcal{L}_{\text{align}}, \mathcal{L}_{\text{recon}}, \mathcal{L}_{\text{KL}}$	Alignment; reconstruction; KL losses
$\lambda_{\text{KL}}, \text{KL}(\cdot\|\cdot)$	KL weight (0–0.005); KL divergence
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution
Stage 3: Operating	
$\mathcal{A} = \{\texttt{TTL}, \texttt{LoRA}, \texttt{TTS}, \texttt{Latent}\}$	Adaptation strategy set
$\alpha \in \mathcal{A}, \alpha^*$	Strategy; optimal strategy
$\pi_\phi, \tau, R(\tau, \alpha)$	Refinement policy; task context; reward
$\mathcal{L}_{\text{RL}}, \mathcal{E}$	RL objective; evaluation metric
TTL & Latent Space Modification	
$\mathcal{P}(x; \theta), \mathcal{P}_0$	Perplexity; threshold
$S(x), \lambda, T, M$	Selection weight; scaling; seq len; #samples
$H, H', \delta, \delta_{\text{opt}}$	Hidden features; modified; additive param
$W_{\text{LM}}, \|V\|, d, n$	LM head; vocab size; hidden dim; seq len
Two-Subspace Mixing LoRA	
$\mathbf{A}_i, \mathbf{B}_i$	Rank-1 LoRA components
$\mathbf{I}_{r \times r}, \lambda$	Identity mixer; interpolation param (0–1)
Loss Functions & Analysis	
$\ell, \mathcal{L}_{\mathcal{T}}, \eta$	Task loss; task-specific loss; learning rate
$\cos(g_i, g_j), \mathbf{g}_{\mathcal{T}}$	Gradient cosine similarity; gradient vector
$\|W\|_1, \|W\|_2$	L1 and L2 norms
Key Hyperparameters	
LR (Stage 1)	10^{-4} (pretrain), 10^{-5} (fine-tune)
Steps, Batch	75/50 steps; batch size 32
VAE Training	4000 epochs; LR 10^{-3}/10^{-4}

Table 7: Notation reference for mathematical symbols and hyperparameters.

Task	Pretraining				Fine-Tuning	
	Learning Rate	Steps	Batch Size	# Samples	Learning Rate	Steps
Common Sense	1×10^{-4}	75	32	5,000	1×10^{-5}	50

Table 8: Checkpoint collection settings for LoRA parameter generation

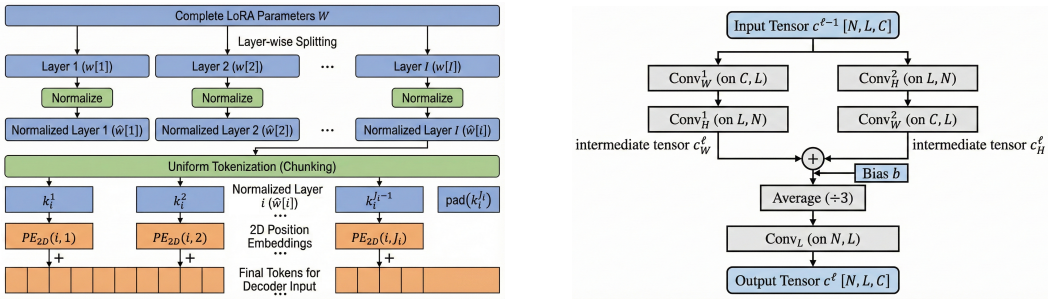

Figure 8: (a) Tokenization of LoRA parameters *(left)* and (b) Catching Module Architecture *(right)*

Layer-wise splitting & normalization- Given complete LoRA adapter parameters W spanning all layers, first parameters are segregated by layer index and then layer-wise normalization is applied to reduce distribution shifts across layers:

$$W \xrightarrow{\text{split by layer}} [w[1], \ldots, w[I]] \xrightarrow{\text{normalize}} [\hat{w}[1], \ldots, \hat{w}[I]] \tag{11}$$

Uniform tokenization- Each normalized layer $\hat{w}[i]$ is then partitioned into contiguous, non-overlapping chunks of uniform size k (with padding applied to the final chunk if necessary):

$$\hat{w}[i] \xrightarrow{\text{tokenize}} K[i] = \left[k_i^1, k_i^2, \ldots, \text{pad}\left(k_i^{J_i}\right)\right], \tag{12}$$

where J_i denotes the number of tokens for layer i and $\text{pad}(\cdot)$ indicates zero-padding to achieve uniform token length k. Each checkpoint W is then assigned a unique permutation state S encoded as a one-hot vector. Each token is further augmented with 2D sinusoidal position embeddings. For the j-th token in layer i, $e_i^j = \text{PE}_{2D}(i, j)$, is computed, where the first dimension encodes layer index i and the second dimension encodes in-layer token position j.

For Qwen2.5-0.5B-Instruct with LoRA rank $r = 8$, each layer's LoRA matrices have dimensions 8×896. With token size $k = 1024$, we obtain 7 tokens of size 8×128 per layer, with the final token padded to 10×130. For Qwen2.5-1.5B-Instruct with $r = 16$, matrices of size 16×1536 decompose into 6 tokens of 16×256, padded to 18×258. These tokenization schemes balance information density with computational tractability.

E.4 Hyper-Convolutional Decoder Architectures

The hyper-convolutional decoder maps prompt embedding tensors $\mathbf{C} \in \mathbb{R}^{N \times L \times C}$ to tokenized LoRA parameters. We denote decoder architectures using a tuple (N, L, C) representing the input prompt dimension, token length and channel width respectively. Table 9 lists the decoder architectures used for different foundation model sizes.

E.5 Hardware and Compute.

All experiments were conducted on a single HPC node running Ubuntu 22.04.1. The system was equipped with an AMD EPYC 8434P CPU (48 physical cores, 96 logical threads), 256 GB of system RAM and four NVIDIA RTX A6000 GPUs, each with 48 GB of dedicated VRAM. GPU-accelerated workloads were executed using CUDA 12.4 and all experiments were implemented in Python 3.12.11.

The same hardware configuration was used for all methods, including our approach and all baselines, ensuring identical compute conditions and avoiding hardware-induced advantages. No system-

Size	Channel Progression (N, L, C)
0.5B	$(n, 384, 384) \to (n, 200, 300) \to (n, 100, 256)$
	$(2n, 50, 200) \to (4n, 50, 200) \to (8n, 25, 200)$
	$(8n, 10, 200) \to (16n, 10, 200) \to (4296, 8, 128)$
1.5B	$(n, 384, 384) \to (n, 200, 300) \to (n, 100, 256)$
	$(2n, 50, 200) \to (4n, 50, 200) \to (8n, 25, 200)$
	$(8n, 10, 200) \to (16n, 10, 200) \to (4508, 18, 258)$

Table 9: Hyper-convolutional decoder architectures used for LoRA parameter generation across model sizes ($n < 128$)

level optimizations or specialized infrastructure beyond standard single-node multi-GPU execution were employed; reported performance differences therefore reflect algorithmic effects rather than differences in computational resources.

E.6 Hyperparameter Summary

This section summarizes hyperparameters and configuration choices that are explicitly specified in the main paper and appendix, collected here for ease of reproducibility.

LoRA Configuration. All experiments use LoRA adapters with identical parameterization across methods. For Qwen2.5-0.5B-Instruct, the LoRA rank is $r = 8$, yielding adapter matrices of size 8×896. For Qwen2.5-1.5B-Instruct, the rank is $r = 16$, yielding matrices of size 16×1536. LoRA adapters are applied to the query, key, value and output projections in attention layers, as well as the gate, up and down projections in MLP blocks. All base model parameters remain frozen.

Catching (Stage 1). LoRA checkpoints used for training the hyper-convolutional parameter generator are collected using a two-phase procedure. In the pretraining phase, models are trained for 75 steps with learning rate 1×10^{-4}. In the fine-tuning phase, training continues for 50 steps with learning rate 1×10^{-5}. Both phases use a batch size of 32 and up to 5,000 samples per dataset, as summarized in Table 8.

Prompt Encoding and Decoder Architecture. Task prompts are encoded using a frozen Sentence-BERT (all-MiniLM-L6-v2) encoder with maximum sequence length 384. The hyper-convolutional decoder consists of cascaded convolutional blocks, each containing five convolutional layers, with channel progressions specified in Table 9.

E.7 Dataset Descriptions

We evaluate CAO-LLM on a diverse collection of benchmarks spanning commonsense reasoning, mathematics, logic, social reasoning, medical question answering and code understanding. Following prior meta-learning and parameter-generation work (Liang et al., 2025), we distinguish between *in-domain* tasks used during catching and *out-of-domain* tasks used solely for evaluation.

In-Domain Tasks. By in-domain tasks, we refer to tasks that are included during the catching (Stage 1), following a leave-one-out meta-training protocol. ARC-Challenge and ARC-Easy (Clark et al., 2018) contain grade-school-level multiple-choice science questions designed to test elementary reasoning. HellaSwag (Zellers et al., 2019) evaluates commonsense inference by requiring models to select the most plausible continuation of a given context from adversarially constructed alternatives. BoolQ (Clark et al., 2019) consists of naturally occurring yes/no questions derived from real-world passages. PIQA (Bisk et al., 2019) focuses on physical commonsense reasoning in everyday situations, requiring selection of the most plausible solution. WinoGrande (Sakaguchi et al., 2021) is a large-scale dataset for commonsense reasoning framed as a fill-in-the-blank task with binary choices.

Out-of-Domain Tasks. Out-of-domain tasks are never used during catching and serve to evaluate robustness under domain shift. GSM-8K (Cobbe et al., 2021) consists of grade-school mathematical word problems requiring multi-step arithmetic reasoning. MATH (Hendrycks et al., 2021) contains challenging competition-level mathematics problems spanning algebra, geometry and number theory.

DivLogicEval (Chung et al., 2025) assesses logical reasoning through counterintuitive natural-language questions designed to isolate pure logical inference. SocialIQA (Sap et al., 2019) evaluates social and emotional commonsense reasoning in everyday interactions. CodeMMLU (Manh et al., 2025) is a multitask benchmark for code understanding, covering program analysis, bug detection and software engineering concepts across multiple programming languages. The JAMA Clinical Challenge (Chen et al., 2025a) consists of expert-curated medical case questions with detailed explanations, designed to evaluate clinical reasoning and decision-making.

F ADAPTING: CALIBRATED RESPONSE WITH FORGETTING PREVENTION

This section provides detailed implementation information for the Fusion VAE used in Stage 2 (Adapting).

F.1 TRAINING DATA PREPARATION

The LoRA checkpoints collected during Stage 1 for training the parameter generator are reused for Fusion VAE training. For each checkpoint, we compute its corresponding task vector using the training dataset of that task. Specifically:

- LoRA parameters are flattened into 1D vectors for processing
- Task vectors are computed as the difference between fine-tuned and pre-trained model outputs on task-specific prompts
- Training pairs $(l^{(i)}, v_{\mathcal{T}_i})$ are constructed from LoRA parameters and corresponding task vectors

F.2 FUSION VAE ARCHITECTURE

The VAE employs a 1D convolutional encoder-decoder architecture:

Encoder.

- 5-layer 1D CNN with channel progression $[1 \to 32 \to 64 \to 128 \to 256 \to 512]$
- Kernel size: 4, CAO-LLM: 4
- Adaptive average pooling to spatial dimension 32
- Separate linear heads for mean μ and log-variance $\log \sigma^2$

Task Vector Integration. Task vectors are projected to 512 dimensions via a linear layer and concatenated with compressed LoRA features before latent encoding. The decoder receives both the sampled latent $\mathbf{z}$ and the projected task vector for reconstruction.

Decoder. Mirror architecture of the encoder using transposed convolutions, with final output matching the original LoRA parameter dimension.

F.3 TRAINING PROCEDURE

Training follows a meta-learning framework with the following hyperparameters:

- **Meta-epochs**: 4000
- **Inner loop steps**: 1
- **Inner learning rate**: 1×10^{-3}
- **Meta learning rate**: 1×10^{-4}
- **KL weight**: Annealed from 0 to 0.005 over training
- **Optimizer**: Adam with default parameters

The loss function combines reconstruction error (MSE) and KL divergence regularization:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{recon}} + \lambda_{\text{KL}} \cdot \mathcal{L}_{\text{KL}} \tag{13}$$

Algorithm 1 Stage II: Task-Conditioned Fusion Alignment via Conditional VAE

Require: Training tasks $\{\mathcal{T}_i\}$, generator outputs θ^a_{gen}, encoder/decoder parameters (ϕ, ψ), KL weight λ_{KL}

for each alignment iteration **do**
  Sample task $\mathcal{T}_i \sim p(\mathcal{T})$
  Compute task semantic signature $\mathbf{v}_{\mathcal{T}_i}$
  Encode conditional posterior:

$$(\mu_i, \sigma_i) \leftarrow q_\phi\big(\mathbf{z} \mid \theta^a_{\text{gen}}, \mathbf{v}_{\mathcal{T}_i}\big)$$

  Sample latent:

$$\mathbf{z}_i \sim \mathcal{N}(\mu_i, \text{diag}(\sigma_i^2))$$

  Decode aligned adapter:

$$\theta^a_{\text{aligned}} \leftarrow \text{Dec}_\psi(\mathbf{z}_i, \mathbf{v}_{\mathcal{T}_i})$$

  Reconstruction loss:

$$\mathcal{L}_{\text{recon}} \leftarrow \left\|\theta^a_{\text{gen}} - \theta^a_{\text{aligned}}\right\|^2$$

  KL regularization:

$$\mathcal{L}_{\text{KL}} \leftarrow D_{\text{KL}}(q_\phi(\mathbf{z} \mid \cdot) \parallel \mathcal{N}(0, I))$$

  Full alignment objective:

$$\mathcal{L}_{\text{align}} \leftarrow \mathcal{L}_{\text{recon}} + \lambda_{\text{KL}}\mathcal{L}_{\text{KL}}$$

  Update encoder/decoder:

$$(\phi, \psi) \leftarrow (\phi, \psi) - \eta\nabla_{\phi,\psi}\mathcal{L}_{\text{align}}$$

end for
Return: aligned adapter distribution $\theta^a_{\text{aligned}}$ for downstream refinement.

F.4 Algorithmic Formulation of Stage II Alignment

For completeness, we provide the explicit optimization procedure used to train the task-aware conditional VAE in Stage II. The objective is to transform the generated adapter parameters from Stage I into semantically aligned parameters that respect the target task geometry before refinement.

F.5 Inference Pipeline

At inference time for an unseen task:

1. **Initial LoRA Generation**: Generate initial LoRA parameters using the Stage 1 parameter generator from target task prompts
2. **Task Vector Computation**: Compute task vector from target task prompts using the pre-trained model
3. **VAE Encoding**: Pass generated LoRA parameters and task vector through the Fusion VAE encoder to obtain latent representation
4. **Aligned Decoding**: Sample from the latent distribution and decode conditioned on the task vector to obtain aligned LoRA parameters

This process ensures that the output LoRA parameters are semantically aligned with the target task while preserving the knowledge in Stage 1.

G Operating: Scalable Test-Time Strategy Selection

This section provides a comprehensive treatment of the four adaptation strategy families available in Stage III of CAO-LLM: Test-Time Learning (TTL), Two-Subspace Mixing LoRA, Test-Time Scaling (TTS) and Latent Space Modification. Each strategy represents a bounded inference-time transformation that can be selected by the reinforcement learning–based refinement policy π_ϕ to optimize task-specific performance.

Recall from Section 2 that refinement is cast as a decision-making problem:

$$\alpha^* = \arg\max_{\alpha \in \mathcal{A}} \mathbb{E}_{\alpha \sim \pi_\phi(\cdot|\tau)}[\, R(\tau, \alpha)\,], \tag{14}$$

where each $\alpha \in \mathcal{A} = \{\texttt{TTL}, \texttt{LoRA}, \texttt{TTS}, \texttt{Latent}, \dots\}$ specifies a concrete adaptation operator with an explicit hyperparameter configuration. The strategy model emits structured JSON outputs that are parsed deterministically at deployment.

G.1 Test-Time Learning (TTL)

Test-Time Learning adapts the model using only unlabeled test inputs by minimizing input perplexity. The key insight is that reducing perplexity on a question x implicitly improves answer quality when the question-answer pair is semantically aligned.

Problem Setting. Let f_Θ be a pretrained autoregressive LLM with frozen base parameters Θ. Given unlabeled test inputs $\mathcal{D}_{\text{Test}} = \{x_j\}_{j=1}^M$, TTL adapts the model by updating only lightweight LoRA parameters $\Delta\Theta$, yielding adapted parameters $\tilde{\Theta} = \Theta + \Delta\Theta$.

Perplexity Objective. For token sequence $x = (x_1, \dots, x_T)$, perplexity is defined as:

$$\mathcal{P}(x; \Theta) = \exp\Big(-\frac{1}{T} \sum_{t=1}^{T} \log p(x_t \mid x_{1:t-1}; \Theta) \Big). \tag{15}$$

Since ground-truth answers y are unavailable at test time, TTL minimizes *input* perplexity: $\min_\Theta \mathcal{P}(x; \Theta)$.

Sample-Efficient Weighting. Not all test samples contribute equally. TTL uses a perplexity-based selection weight that prioritizes high-perplexity samples:

$$S(x) = \lambda \cdot \exp\big(\log \mathcal{P}(x; \Theta) - \log \mathcal{P}_0 \big)\, \mathbb{I}_{\{\mathcal{P}(x;\Theta) > \mathcal{P}_0\}}(x), \tag{16}$$

where $\mathcal{P}_0$ is a threshold and λ is a scaling constant. Low-perplexity samples are excluded ($S(x) = 0$), focusing adaptation on samples where the model is most uncertain.

LoRA-Based Adaptation. To prevent catastrophic forgetting and reduce compute, TTL updates only low-rank parameters $\Delta\Theta = \mathcal{B}\mathbf{A}$, yielding the weighted objective:

$$\min_{\Delta\Theta} \; S(x)\, \mathcal{P}(x; \Theta + \Delta\Theta). \tag{17}$$

Configuration Schema.

```
{
  ``family": ``TTT",
  ``ttl_steps": <integer>,
  // number of optimization steps
  ``learning_rate": <float>,
  // optimizer learning rate
  ``batch_size": <integer>,
  // samples per update
  ``shuffle_data": <boolean>
  // whether to shuffle test inputs
}
```

G.2 Two-Subspace Mixing LoRA

This strategy enhances standard LoRA by enabling subspace interaction through a fixed butterfly mixing factor, providing richer representational capacity without additional trainable parameters.

Algorithm 2 TTL: Test-Time Learning via Weighted Perplexity Minimization

Require: Test batch $\mathcal{X} = \{x_b\}_{b=1}^{B}$, pretrained LLM f_Θ, LoRA params $\Delta\Theta$, threshold $\mathcal{P}_0$
1: Initialize LoRA: $\mathbf{A} \sim \mathcal{N}(0, \sigma^2)$, $\mathcal{B} = 0$
2: $\tilde{\Theta} \leftarrow \Theta + \Delta\Theta$
3: **for** each batch $\mathcal{X}$ **do**
4: Compute perplexities $\mathcal{P}(x_b; \tilde{\Theta})$
5: Compute weights $S(x_b)$; exclude low-perplexity samples
6: Update $\Delta\Theta$ by minimizing $\sum_{b=1}^{B} S(x_b)\, \mathcal{P}(x_b; \tilde{\Theta})$
7: **end for**
8: **Return:** adapted model $f_{\Theta+\Delta\Theta}$

LoRA as Subspace Composition. For frozen pretrained weights $\mathbf{W}_0 \in \mathbb{R}^{d_1 \times d_2}$ and input x, standard LoRA computes:

$$x\mathbf{W}_0 + x\Delta\mathbf{W} = x\mathbf{W}_0 + x\mathbf{AB}, \tag{18}$$

with $\mathbf{A} \in \mathbb{R}^{d_1 \times r}$, $\mathbf{B} \in \mathbb{R}^{r \times d_2}$ and $r \ll \min(d_1, d_2)$. Decomposing into rank-1 components:

$$\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_r], \quad \mathbf{B}^\top = [\mathbf{B}_1^\top, \mathbf{B}_2^\top, \ldots, \mathbf{B}_r^\top],$$

vanilla LoRA can be viewed as:

$$x\mathbf{AB} = x\sum_{i=1}^{r} \mathbf{A}_i\mathbf{B}_i = x\mathbf{A}\mathbf{I}_{r\times r}\mathbf{B}. \tag{19}$$

Two-Subspace Mixing. The two-subspace mixing variant replaces the identity mixer with a butterfly factor that enables cross-subspace interaction:

$$x\sum_{i=1}^{r/2}(\mathbf{A}_i + \mathbf{A}_{i+r/2})(\mathbf{B}_i + \mathbf{B}_{i+r/2}) = x\mathbf{A}\begin{bmatrix}\mathbf{I} & \mathbf{I}\\ \mathbf{I} & \mathbf{I}\end{bmatrix}\mathbf{B}. \tag{20}$$

This formulation mixes information from $2r$ subspaces (compared to r in vanilla LoRA), modeling richer interactions without introducing additional learnable weights.

Interpolation Parameter. The λ hyperparameter controls the interpolation ratio between the two resulting subspace outputs during inference, allowing fine-grained control over the adaptation strength.

Configuration Schema.

```
{
  ``family": ``LORA",
  ``lambda": <float>
  // mixing ratio between subspaces
  // (0.0 to 1.0)
}
```

G.3 TEST-TIME SCALING (TTS)

Test-Time Scaling leverages multiple prompt batches to improve prediction stability through either routing or ensembling. This approach is particularly effective for tasks where multiple plausible interpretations compete.

Router Approach. Given a test question, TTS samples m prompt batches from the test split and selects the batch whose representation is closest to the question. Two routing methods are supported:

- **M1 (avg_sim_score):** Compute similarity scores between each prompt in a batch and the test question, then average across the batch. Select the batch with highest average similarity.
- **M2 (avg_prompt_embed):** Compute the mean prompt embedding for each batch, then select the batch whose mean embedding is closest to the test question embedding.

Euclidean distance was found to perform better empirically than cosine similarity for both methods.

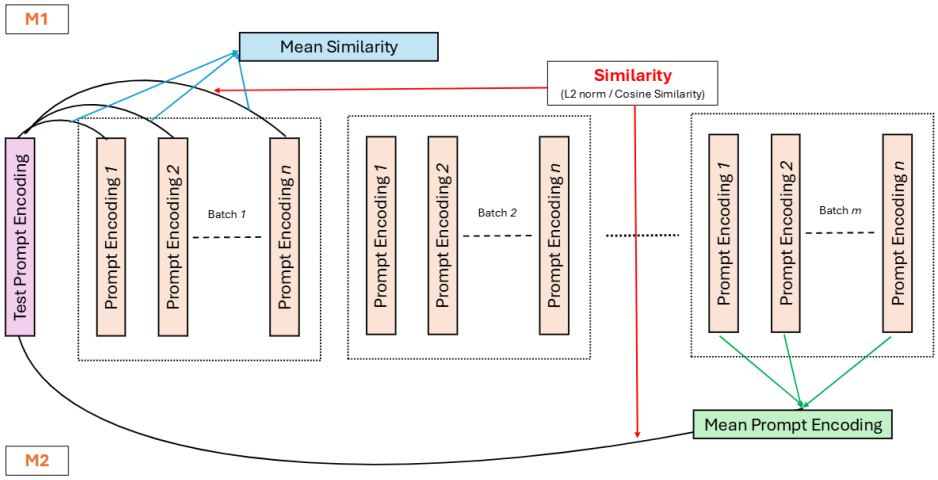

Figure 9: **TTS Router Architecture.** M1 computes mean similarity scores between the test prompt encoding and individual prompt encodings within each batch. M2 computes similarity between the test encoding and mean prompt embeddings per batch.

Ensemble Approach. For tasks requiring aggregation across multiple adapter configurations:

- **max_confidence:** Select the prediction with highest confidence score across all configurations.
- **majority_vote:** Aggregate predictions via voting across configurations.
- **sum_logprobs:** Sum log-probabilities across configurations for each candidate answer.

Configuration Schema.

```
{
  ``family": ``TTS",
  ``num_prompt_batches": <integer>,
  // batches sampled from test split
  ``method": ``<avg_sim_score |
  avg_prompt_embed |
  max_confidence |
  majority_vote |
  sum_logprobs>"
}
```

G.4 Latent Space Modification

We introduce a lightweight, per-sample parameter applied to the final hidden layer, enabling rapid adaptation without modifying backbone parameters.

Problem Setting. Let $\mathcal{M}$ be a pretrained autoregressive LM with fixed parameters θ. Given input prompt $x = (x_1, \ldots, x_n)$, the model produces final hidden features $H = \mathcal{M}_{\text{pre-LM}}(x) \in \mathbb{R}^{n \times d}$ before the LM head. Next-token probabilities are $p(y \mid x) = \text{softmax}(W_{\text{LM}} H)$.

Sample-Specific Parameter. SLOT introduces a per-sample additive parameter $\delta \in \mathbb{R}^{1 \times d}$, broadcast across sequence positions:

$$H' = H + \delta, \qquad \text{logits} = W_{\text{LM}}(H + \delta). \tag{21}$$

Algorithm 3 SLOT: Sample-Specific LM Optimization at Test-Time

Input: pretrained LM $\mathcal{M}(\theta)$, prompt x, steps T, lr η
Initialize $\delta \leftarrow 0 \in \mathbb{R}^{1\times d}$
for $t = 0$ to $T-1$ **do**
  $H \leftarrow \mathcal{M}_{\text{pre-LM}}(x)$ (can be cached)
  $\mathcal{L}(\delta) \leftarrow -\sum_{i=1}^{n-1} \log p(x_{i+1} \mid x_{1:i}, \delta)$
  $\delta \leftarrow \text{OptimizerStep}(\delta, \nabla_\delta \mathcal{L}; \eta)$
end for
Decode: generate with logits $W_{\text{LM}}(H_{\text{last}} + \delta)$
Return: generated continuation y

Prompt-Stage Optimization. Initialize $\delta^{(0)} = \mathbf{0}$ and optimize for T steps by minimizing the negative log-likelihood of the prompt sequence:

$$\mathcal{L}(\delta) = -\sum_{i=1}^{n-1} \log p(x_{i+1} \mid x_{1:i}, \delta). \tag{22}$$

Since δ is applied only to the final hidden layer, features H can be cached; each optimization step requires only forward/backward through the linear head W_{LM}, yielding negligible overhead.

Generation Stage. During autoregressive decoding, SLOT reuses δ_{opt} without further optimization:

$$x_{\text{next}} \sim \text{softmax}\big(W_{\text{LM}}(H_{\text{last}} + \delta_{\text{opt}})\big). \tag{23}$$

Computational Cost. Prompt-stage adaptation optimizes only d parameters with per-step cost $O(d|V|)$ and generation-time overhead $O(d)$ per token.

Configuration Schema.

```
{
  ``family": ``Latent",
  ``slot_steps": <integer>,
  // optimization steps (T)
  ``slot_lr": <float>
  // learning rate (eta)
}
```

G.5 Strategy Selection via Reinforcement Learning

The refinement policy π_ϕ selects strategies through a language-model-driven generation procedure. Given task context and few-shot exemplars, the strategy model emits structured JSON outputs specifying both the selected family and its configuration. This ensures refinement policies remain interpretable and directly executable.

Objective Formulation. We first formulate the objective for outer-loop RL training which generates adaptation strategies α. Let θ denote the parameters of the language model $\texttt{LM}_\theta$. In order to adapt to an unseen dataset (task) $\mathcal{D}$, CAO-LLM requires as specified in Section 2, τ which is a context containing information relevant to the task and $\mathcal{E}$ which is the evaluation strategy and metric used to assess the model's downstream adaptation. Based on τ, CAO-LLM generates an α and updates its parameters accordingly $\theta' \leftarrow \texttt{Update}(\theta, \alpha)$. We thus have an RL setup i.e., the model takes an *action* (generating α), receives a *reward* r based on $\texttt{LM}_{\theta'}$'s performance on $\mathcal{E}$ and updates its policy to maximize expected reward,

$$\mathcal{L}_{\text{RL}}(\theta_t) := -\mathbb{E}_{(\tau,\mathcal{E})\sim\mathcal{D}}\left[\mathbb{E}_{\alpha\sim \text{LM}_{\theta_t}(\cdot|\tau)}\left[r(\alpha, \mathcal{E}, \theta_t)\right]\right]$$

It is to be noted that the reward assigned to a given action depends on the model parameters θ at the time the action is taken (since θ is updated to θ', which is then evaluated). An implication of this is

that the while modeling the RL state, one must therefore include θ in the policy's parameters as well along with τ, even though the policy's observation is limited to τ (because it is extremely infeasible to directly place θ in the LLM's context window). Therefore, the (state, action, reward) triples which have been collected by using an older model weights, θ_{old}, will not be aligned for the current model θ_{current}. Hence, an on-policy approach should be adapted, by which adaptation strategies are sampled from and, even more importantly, the rewards itself will be calculated using the current model.

In particular, the specific on-policy approach used is ReST^{EM} Singh et al. (2024) where samples are first generated from the current model and are filtered by using binary feedback [$r(\alpha, \mathcal{E}, \theta_t)$ is 1 if on $\mathcal{E}$, α improves LM$_{\theta_t}$'s performance and is 0 otherwise]. The model is then fine-tuned on these samples and this continues in an iterative manner. Currently, only a deterministic number of samples are being generated, 20 to be precise. This could however be improvised to be dynamic in future version of the work wherein samples would continue to be generated until a particular confidence threshold, as determined by the model itself is reached instead. The same is true for number of iterations as well which is just 2 for now. The generation process employs a temperature of 1.0, nucleus sampling (`top-p = 0.9`) and top-k filtering (`top-k = 50`).

Dataset-Specific Configurations. Empirically discovered configurations exhibit intuitive alignment with task demands:

- **ARC-e, JAMA Clinical, PIQA → TTL:** Domain-specific or jargon-dense tasks benefit from token-level distribution correction via perplexity minimization. Example Configuration: {`ttl_steps: 25, learning_rate: 1e-5, batch_size: 4`}.
- **ARC-c, SocialIQA → Latent:** Abstract reasoning and social inference tasks require adjustment of hidden-state trajectories. Example Configuration: {`slot_steps: 5, slot_lr: 0.1`}.
- **BoolQ, GSM-MC, MATH-MC → LoRA:** Structured reasoning tasks benefit from subspace mixing. Example Configuration: {`lambda: 0.5`}.
- **HellaSwag, DivLogicEval, CodeMMLU → TTS:** Adversarial or multi-interpretation tasks benefit from ensemble-style aggregation. Example Configuration: {`num_prompt_batches: 20, method: max_confidence`}.

G.6 JSON Schemas and Prompting Template for Strategy Generation

For reproducibility, we provide the exact prompting template used to generate adaptation strategies. The model is instructed to output a single valid JSON object corresponding to one strategy family, with no additional text. Task context's are obtained using by prompting a foundational model with the few-shot examples of the unseen task analogous to Charakorn et al. (2025).

System Prompt.

```
You are an expert AI agent tasked with generating an optimal
adaptation strategy for a Large Language Model to improve its
performance on a new, unseen task. Your goal is to output a single,
structured JSON object that specifies the most promising strategy.
```

User Prompt.

```
<Task Context>

Your Instruction:
Based on the task context and method descriptions below, generate a
single JSON
object representing the most effective adaptation strategy. You must
choose one
strategy family and output strictly valid JSON with no extra text.
```

```
JSON Output Format \& Strategy Families:

(1) Test-Time Training (TTT)
{
  "family": "TTT",
  "ttl_steps": <integer>,
  "learning_rate": <float>,
  "batch_size": <integer>,
  "shuffle_data": <boolean>
}

(2) LoRA Modification (LORA)
{
  "family": "LORA",
  "lambda": <float>
}

(3) Test-Time Scaling (TTS)
{
  "family": "TTS",
  "num_prompt_batches": <integer>,
  "method": "<avg_sim_score | avg_prompt_embed | max_confidence |
  majority_vote | sum_logprobs>"
}

(4) Latent Space Modification (Latent)
{
  "family": "Latent",
  "slot_steps": <integer>,
  "slot_lr": <float>
}

Now, provide only the JSON object for the <task> dataset.
```

Task Contexts
Generation Prompt: This prompt is used for querying `GPT-4o mini` to obtain the task descriptions for datasets

```
You are given a small set of example question-answer pairs from an
unknown dataset. Your task is to infer the
underlying task definition and write a concise, professional task
description suitable for inclusion in a machine
learning benchmark paper.

Instructions:
1. Do not mention specific example questions or answers.
2. Infer the core objective, skills being evaluated, and
type of reasoning required.
3. Describe what the model is expected to do and what
competencies are being tested.
4. Write in neutral, academic language.

Output a single self-contained task description paragraph.

Examples from the dataset are provided below:
<question-answer examples>

Output only the task description. Do not include analysis, bullet
points, or headings.
```

Hereby, are the example task contexts used in this work, **(a) ARC-c**

```
This task is about analyzing questions which examine your grasp of
scientific ideas. You must connect conceptual knowledge with practical
examples from geology, ecology and environmental changes. The
objective here is to evaluate various scientific scenarios and infer
the most logical explanations or definitions based on established
knowledge. This task will strengthen your analytical and reasoning
skills in the context of natural science. Your role is to interpret
questions focusing on earth science and biological interactions. This
demands a clear understanding of relevant processes, such as
decomposition, weathering, and species adaptation.
```

(b) ARC-e

```
Your job is to discern which information best answers a posed
question, focusing on practical examples and scientific principles.
This requires a strong grasp of underlying concepts in ecology or
physics. You will analyze questions that explore important connections
such as environmental issues or animal adaptations. Utilize your
background knowledge to evaluate and select the most fitting answer.
This task involves selecting answers that reflect accurate
relationships or effects seen in nature or society. You will need to
sort through potential choices critically to find the appropriate one.
```

(c) HellaSwag

```
This task revolves around completing an unfinished text by selecting
an ending that matches its tone and context. It requires you to think
critically about how narratives develop and conclude effectively. This
task asks you to select a suitable conclusion for an unfinished
narrative or instructional content. It tests your comprehension and
reasoning skills as you assess how well each option aligns with the
given text. Your task involves completing an incomplete passage by
selecting the ending that logically continues the context provided.
This requires reading comprehension and the ability to infer meaning
from a text.
```

(d) PIQA

```
You will explore practical questions and select an answer that
presents a logical and widely accepted approach to solve a given
problem or complete a task successfully. Analyze the provided
scenarios where practical advice or solutions are required, focusing
on selecting the most. commonly used or convenient method. Given a
question related to common tasks, your responsibility is to discern
which proposed solution aligns with typical practices or makes the
task easier to achieve.
```

(e) WinoGrande

```
In this exercise, you need to read short narratives and discern which
person or object fits best within the context of the sentence.This
task requires synthesizing information from concise textual scenarios
to identify crucial elements that drive the narrative forward. The
goal is to evaluate descriptions and select the entity that best
aligns with the sentiments or actions presented in the scenario.
```

(f) BoolQ

```
Analyze the given details about various subjects, including movies,
sports, and television shows. Your role is to confirm whether certain
claims are true or false. Your task is to determine the truthfulness
of specific statements based on the provided background information.
This requires careful reading and comprehension of the content. The
goal is to evaluate factual claims made in relation to highlighted
texts. You will need to discern whether the statements align with the
information provided.
```

(g) GSM-8K

```
You will be tasked with interpreting mathematical situations described
in words. The goal is to use logical
reasoning and calculations to determine the numerical answers based on
the context provided. This task challenges your problem-solving
abilities through mathematical reasoning. You must carefully read each
scenario and systematically work through the data to compute the final
outcome. Your role is to engage with practical math scenarios
presented as questions. The task requires translating textual data
into numerical operations that will lead you to the final solution.
```

(h) MATH

```
This task focuses on solving challenging mathematical problems that
require multi-step logical reasoning rather than direct formula
application. You will analyze competition-level mathematics questions
spanning topics such as algebra, geometry, number theory, probability,
and calculus. The objective is to carefully interpret each problem,
identify appropriate problem-solving strategies, and carry out precise
symbolic or numerical reasoning to arrive at a correct final answer.
This task emphasizes structured thinking, the use of mathematical
heuristics, and the ability to connect multiple concepts within a
single solution. Your role is to reason through complex scenarios,
perform intermediate derivations when necessary, and produce an exact
answer that adheres to standard mathematical conventions. The task
evaluates deep mathematical understanding and disciplined reasoning
rather than surface-level computation or pattern matching.
```

(i) DivLogicEval

```
This task focuses on evaluating your ability to perform precise
logical reasoning over natural language statements. You will be given
a set of premises written in fluent but often counterintuitive
language, where commonsense intuition alone may be misleading. Your
objective is to analyze the logical structure underlying these
statements and determine which option is logically entailed, not
entailed, or required as a missing assumption. The task demands
careful attention to implications, negations, and dependencies between
statements rather than surface-level meaning. You must rely on formal
reasoning principles to assess whether conclusions follow from the
premises. This task is designed to isolate logical reasoning skills by
minimizing reliance on background knowledge or real-world
plausibility.
```

(j) SocialIQA

```
This task focuses on reasoning about everyday social situations that
involve human interactions, intentions, and emotional responses. You
are given a short context describing a social scenario, followed by a
question that probes implicit social commonsense. Your objective is to
select the most plausible answer from multiple choices based on how
people typically think, feel, or act in such situations. The questions
require you to infer motivations, emotional reactions, social norms,
or likely actions before or after an event. This task evaluates your
ability to reason about social dynamics, perspective-taking, and
cause-effect relationships in human behavior. Successfully completing
this task demands an understanding of common social expectations and
the ability to apply Theory of Mind reasoning to interpret the mental
states of individuals involved.
```

(k) CodeMMLU

```
This task focuses on evaluating your understanding of programming
concepts, code semantics, and software reasoning across a wide range
of difficulty levels. You will analyze questions that involve reading,
interpreting, and reasoning about code snippets written in common
programming languages. The objective is to assess your ability to
understand control flow, data structures, algorithms, and
language-specific behaviors. This task requires careful examination of
program logic, identification of errors or expected outputs, and
selection of the most appropriate answer based on correct
computational reasoning. Your role is to apply foundational and
advanced coding knowledge to infer how programs execute and how
modifications affect their behavior. The task emphasizes precise
reasoning about syntax, semantics, and program execution rather than
surface-level pattern matching.
```

(l) JAMA Clinical

```
This task focuses on answering and explaining complex real-world
clinical cases drawn from challenging medical scenarios. You are
required to analyze detailed patient case descriptions that may
include atypical presentations, incomplete information, or competing
diagnoses. The objective is to apply clinical reasoning to identify
the most appropriate diagnosis or next management step from multiple
answer choices. Beyond selecting the correct option, this task
emphasizes understanding why that choice is correct and why
alternative options are less appropriate. Successfully completing this
task requires synthesizing medical knowledge across domains,
interpreting clinical findings, and reasoning in a manner consistent
with expert decision-making in real clinical settings. The task
evaluates both diagnostic accuracy and the ability to justify medical
decisions using coherent, medically sound explanations.
```

H BASELINE HYPER-PARAMETERS

For fair comparison, all baseline methods use the same LoRA configuration (rank $r = 8$ for 0.5B and $r = 16$ for 1.5B), training datasets, and hardware setup as CAO-LLM. This section documents the specific hyperparameters used for each baseline method.

MAML-en-LLM: Following the implementation in Sinha et al. (2024), we use LoRA adapters with rank $r = 8$ for the 0.5B model and $r = 16$ for the 1.5B model, with LoRA alpha $\alpha = 16$ and scaling factor α/r applied to weight updates. LoRA adapters are applied to the Q, V, and O projections in attention layers. The method uses a MAML-2-1 configuration with $n = 1$ task per step and $k = 1$ adaptation step per task, trained with batch size 1 as specified in the original paper.

Both inner and outer learning rates are set to 1×10^{-5}, and training proceeds for 10 epochs with a maximum of 50,000 steps. The optimizer is a shared AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and weight decay 0.01, where first and second moment estimates are shared between inner and outer optimizers. Gradient clipping is applied with maximum norm 1.0, and test-time adaptation uses 10 gradient steps on 25 support samples.

ABMLL: Following the implementation in Zhang et al. (2025), we use LoRA rank $r = 8$ for both model sizes, with adapters applied to Q, K, V, and O projections in attention layers. Each LoRA layer maintains four adapters representing mean and variance for both global θ and task-specific ϕ parameters: B_{μ_θ}, A_{μ_θ}, B_{σ_θ}, A_{σ_θ}, B_{μ_ϕ}, A_{μ_ϕ}, B_{σ_ϕ}, and A_{σ_ϕ}, all initialized with scale 0.01. The method uses beta $\beta = 5 \times 10^{-7}$ to control $\mathrm{KL}(q(\phi|D)\|p(\phi|\theta))$ regularization, gamma $\gamma = 2 \times 10^{-20}$ to control $\mathrm{KL}(q(\theta)\|p(\theta))$ prior regularization, numerical stability constant $c = 1 \times 10^{-20}$, and sigma stabilizer 1×10^{-6} added to softplus outputs. The Gamma prior on precision uses hyperparameters $a_0 = 1.0$ and $b_0 = 0.01$. Training uses inner and outer learning rates of 1×10^{-5}, with 5 inner loop steps for task-specific adaptation and 5 outer loop batches for meta-gradient computation. The batch size is 2 for distributed training (with per-device theoretical limit of 16), and training proceeds for 10 epochs. The outer loop optimizer is AdamW with default parameters for global θ parameters, while the inner loop uses SGD with learning rate 1×10^{-5} for task-specific ϕ parameters. After each outer loop update, ϕ parameters are reset to match θ parameters. Gradient clipping with maximum norm 1.0 is applied, and test-time adaptation uses 10 gradient steps on 25 support samples.

I Limitations and Future Work

While CAO-LLM demonstrates consistent improvements over bi-level meta-learning baselines across multiple benchmarks, several limitations warrant acknowledgment, and promising directions for future research emerge from this work.

Model Scale and Architecture Coverage. Our experiments focus on Qwen2.5 models at 0.5B and 1.5B parameter scales. Validation on larger models (e.g., 7B, 13B, or 70B parameters) would strengthen claims regarding scalability and confirm whether the benefits of temporal separation persist at scales where models possess greater inherent capacity. Furthermore, extending evaluation to other LLM families beyond Qwen2.5-such as Llama Touvron et al. (2023), DeepSeek DeepSeek-AI et al. (2024), or Mixtral Jiang et al. (2024)-would provide evidence for the generality of CAO-LLM's design principles across architectural variations and pretraining regimes.

Parameter-Efficient Fine-Tuning Method Generalization. CAO-LLM's current implementation relies exclusively on Low-Rank Adaptation (LoRA) as the parameter-efficient fine-tuning mechanism. Exploring how distribution shift paradigm extends to alternative PEFT methods-such as prompt tuning Lester et al. (2021), prefix tuning Li & Liang (2021) or adapter layers Liu et al. (2022)-represents an important direction. Different PEFT methods impose distinct structural constraints on the adaptation process, and understanding whether CAO-LLM's three-stage decomposition remains effective under these alternative parameterizations would clarify the scope of its applicability.

Evaluation on Multi-Turn and Interactive Settings. Our evaluation is limited to single-turn benchmarks, where tasks can be evaluated through isolated question-answering or generation tasks. Multi-turn dialogue, interactive reasoning, and task-oriented conversation settings introduce temporal dependencies, context accumulation, and dynamic adaptation requirements that are not captured in our current experimental design. Extending CAO-LLM to these settings would require investigating how foundation consolidation, alignment, and refinement interact with conversation state and whether stage boundaries should be reconsidered in temporally extended interactions.

Identifying Task-Critical Parameters via Mechanistic Interpretability. Inspired by prior work in quantization, pruning, and the lottery ticket hypothesis Tang et al. (2025), we observe that only a subset of parameters may be crucial for adapting the LLM to incoming drift. CAO-LLM currently updates low-rank parameters uniformly across all adapter layers. A promising future direction involves integrating principles from mechanistic interpretability to *first identify task-critical parameter subsets* before applying updates. This could involve analyzing circuit-level activations, gradient saliency, or

attention attribution to selectively target parameters that contribute most to task-specific reasoning. Such an approach could further increase adaptation efficiency and sparsity beyond the improvement already observed.

Unbounded Exploration of Full-Model Weight Modification. CAO-LLM operates within the constraint of frozen base model parameters, performing adaptation exclusively through LoRA adapters. A more challenging and open direction is enabling *unbounded exploration of the hypothesis space for full-model weight modification*. This would involve relaxing the frozen-backbone assumption and developing principled methods for selective full-parameter updates that preserve foundation knowledge while enabling deeper specialization. Potential approaches include learned masking strategies, dynamic parameter allocation, or hierarchical adaptation policies that determine which layers and parameters should be modified based on task characteristics.

Continual and Sequential Adaptation. CAO-LLM's current formulation assumes a fixed training task distribution and one-time deployment to target tasks. Extending the framework to continual learning settings-where tasks arrive sequentially and the model must adapt without catastrophic forgetting-poses interesting challenges. Questions include: (1) how frequently should foundation consolidation be re-executed as task distributions shift? (2) can alignment and refinement stages be made incremental? and (3) what role does temporal separation play in mitigating interference between sequentially learned tasks?

J Declaration on Generative AI Usage

During the preparation of this work, the author used Large Language Models (GPT-5.2, Claude Opus 4.5 and Gemini-3) as a writing assistant tool for drafting content, to generate literature review, for abstract drafting, to paraphrase and reword, to improve writing style, for grammar and spelling check as well as to generate the images used in the paper. The process was interactive. After writing the core content, the author used LLMs with specific prompts to refine the text. These prompts included requests to "check for grammatical errors," "rephrase this sentence for clarity," "make this paragraph more concise," or "suggest alternative phrasing to improve flow." The LLMs were not used to generate any scientific ideas, experimental results, data analysis or other core intellectual contributions of the paper. After using these tool(s)/service(s), the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

K Acknowledgments

The authors would also like to thank **Professor Sashikumaar Ganesan**, Artificial Intelligence for Research and Engineering Excellence Lab, Indian Institute of Science[1] and Founder of ZenteiQ.ai[2], for offering valuable feedback and providing the adequate compute resources. The authors would also like to thank **Professor Dianbo Liu**, Artificial Scientific Intelligence Lab, National University of Singapore[3] for providing valuable supervision and insights on all project aspects.

[1] https://airexlab.cds.iisc.ac.in/
[2] https://zenteiq.ai/
[3] https://www.asintelligence.xyz/