CORAL: COOPERATIVE MULTI-AGENT ORCHESTRATION FOR LLM ADAPTATION ACROSS DIVERSE ENVIRONMENTS

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

ABSTRACT

Adapting large language models (LLMs) to diverse downstream tasks remains challenging: monolithic optimization approaches - such as bi-level meta-learning - simultaneously handle knowledge consolidation, semantic alignment and task specialization within a single optimization loop, producing diffuse "compromise" solutions under heterogeneous task distributions. We propose CORAL, a cooperative multi-agent framework that decomposes LLM adaptation into three specialized agents: a *Generator Agent* that uses a hyper-convolutional network to produce diverse LoRA adapters from task prompts, an *Alignment Agent* that employs a conditional variational autoencoder to fuse generated parameters with target-task semantics and a *Policy Agent* that leverages reinforcement learning to select task-specific adaptation strategies from a structured action space. Each agent operates with minimal, well-defined information requirements, communicating through compact intermediate representations rather than shared gradient signals. Evaluated on 12 benchmarks spanning mathematics, logic, code, social reasoning and medicine, CORAL consistently outperforms monolithic baselines, across various model sizes. Our results demonstrate that modular multi-agent coordination offers a principled alternative to centralized LLM adaptation.

1 INTRODUCTION

The convergence of multi-agent learning and generative AI presents new opportunities for building LLM-based systems that can reason, coordinate and adapt across diverse environments (Wei et al., 2022; Brown et al., 2020). Yet most LLM adaptation methods remain fundamentally *monolithic*: a single optimization process is tasked with simultaneously consolidating cross-task knowledge, aligning representations with target semantics and specializing for task-specific performance (Finn et al., 2017; Sinha et al., 2024; Zhang et al., 2025). Under heterogeneous task distributions - such as mathematics, clinical reasoning, code understanding and social inference impose conflicting optimization pressures - this centralized approach produces undifferentiated, compromise solutions.

We propose a different perspective: **LLM adaptation as cooperative multi-agent orchestration**. Rather than a single optimizer navigating conflicting objectives, CORAL decomposes adaptation into three specialized agents that coordinate through minimal information exchange:

- **Generator Agent** (§2.2): A generative AI module - a hyper-convolutional network - that produces diverse LoRA adapter configurations conditioned on task prompts. This agent generates diverse agent models (specialized parameter configurations) from natural language descriptions.
- **Alignment Agent** (§2.3): A conditional VAE that fuses the Generator's output with target-task semantic signatures, ensuring representational compatibility before specialization.
- **Policy Agent** (§2.4): An RL-trained LLM that observes task context in natural language and selects adaptation strategies from a structured action space, orchestrating the final specialization step.

This multi-agent decomposition directly addresses several topics central to the MALGAI workshop. Table 1 maps CORAL's design to the workshop's themes.

Table 1: **MALGAI Workshop Themes Alignment Map to CORAL's Multi-Agent Architecture**

MALGAI Theme	Core Mechanism	Operationalization in CORAL	Sec.
Generative AI to generate diverse agent models	**Hypernetwork Synthesis** (GenAI)	**Generator Agent**: Uses a hyper-network to dynamically synthesize diverse LoRA adapter parameters ("agent models") from natural language prompts, validating GenAI's ability to create specialized agents on demand.	§2.2
Cooperative MARL for LLM orchestration	**RL-Based Strategy Optimization**	**Policy Agent**: Orchestrates the adaptation pipeline using Reinforcement Learning. It observes task contexts and cooperatively selects optimal specialization strategies, ensuring coherent system behavior.	§2.4
Formalism of partial information	**Gradient-Free Coordination**	**Modular Communication**: Agents coordinate exclusively through compact intermediate representations (generated weights, aligned latents) without sharing gradients, simulating realistic partial-information constraints.	§2
Natural Language as action / observation space	**Semantic Control Interfaces**	**Natural Language I/O**: The Policy Agent treats natural language task descriptions as observations and outputs structured actions, demonstrating the viability of NL-centric agentic control.	§2.4
Multi-agent systems for modular generative models	**Cooperative Decomposition**	**Alignment Agent**: Enables exploration of the parameter space via a VAE that fuses Generator outputs with task vectors, bridging the gap between general pre-training and specific task requirements.	§2.3

Empirically, CORAL outperforms monolithic baselines (MAML-en-LLM, ABMLL) on both in-domain and out-of-domain benchmarks using Qwen2.5 models at 0.5B and 1.5B scales, demonstrating that multi-agent coordination enables more robust adaptation than centralized optimization.

2 CORAL Framework

2.1 System Overview

CORAL frames LLM adaptation as a cooperative multi-agent system. Given a pretrained LLM with frozen parameters θ_0 and LoRA adapters $\theta^a \in \mathbb{R}^D$ parameterized as $W = W_0 + BA^\top$ with $A \in \mathbb{R}^{r \times d_{\text{in}}}$, $B \in \mathbb{R}^{d_{\text{out}} \times r}$, the three agents coordinate as follows:

$$\underbrace{\mathcal{G}_\phi(\mathcal{C}_\mathcal{T})}_{\text{Generator}} \xrightarrow{\theta^a_{\text{gen}}} \underbrace{\text{VAE}_{\phi,\psi}(\theta^a_{\text{gen}}, \mathbf{v}_\mathcal{T})}_{\text{Alignment}} \xrightarrow{\theta^a_{\text{aligned}}} \underbrace{\pi_\phi(\alpha \mid \tau)}_{\text{Policy}} \rightarrow \theta^a_* \tag{1}$$

Each agent has a distinct *observation space* (its inputs), *action space* (its outputs) and *objective*. Crucially, no agent shares gradients with another - coordination occurs through compact intermediate representations (θ^a_{gen}, $\theta^a_{\text{aligned}}$), formalizing a partial-information multi-agent setting.

2.2 Generator Agent: Generative AI for Diverse Agent Models

The Generator Agent addresses a core MALGAI theme: *using generative AI to produce a diverse set of agent models*. It learns a prompt-conditioned parameter generator $\mathcal{G}_\phi$ that maps task descriptions to LoRA adapter configurations - effectively generating specialized "agent models" on demand. Representative task prompts $\{p_1, \ldots, p_N\}$ are encoded via a frozen text encoder into a conditioning tensor $\mathbf{C} \in \mathbb{R}^{B \times N \times L \times C}$. A hyper-convolutional decoder (Ha et al., 2016; Liang et al., 2025) transforms $\mathbf{C}$ into tokenized LoRA parameters through cascaded convolutional blocks with horizontal, vertical and layer-wise convolutions. The Generator is trained via supervised regression on paired prompt batches and task-specific LoRA checkpoints (Full details provided in Appendix C):

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

2.3 Alignment Agent: Cooperative Coordination

Generated parameters may be misaligned with target-task semantics, particularly under domain shift. The Alignment Agent performs explicit coordination between the Generator's output and target-task requirements using a conditional VAE. Following task-vector formulations (Ilharco et al., 2023), we compute $\mathbf{v}_{\mathcal{T}_*} = \mathbf{z}^{\text{ft}}_{\mathcal{T}_*} - \mathbf{z}^{(0)}$ as the difference between fine-tuned and pretrained last-layer

representations. In the conditional variational auto-encoder, the encoder models $q_\phi(\mathbf{z} \mid \theta^a_{\text{gen}}, \mathbf{v}_{\mathcal{T}_*})$ and the decoder reconstructs aligned parameters $\theta^a_{\text{aligned}} = \text{Dec}_\psi(\mathbf{z}, \mathbf{v}_{\mathcal{T}_*})$. The alignment objective combines reconstruction and regularization:

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

This agent communicates the *minimal information* needed for downstream specialization: aligned adapter parameters that respect both the Generator's consolidated knowledge and the target task's semantic structure. Full details provided in Appendix D.

2.4 Policy Agent: RL-Based Strategy Orchestration

The Policy Agent exemplifies *cooperative MARL for LLM orchestration* and the use of *natural language as both action and observation space*. Given aligned parameters from the Alignment Agent, it selects a task-specific adaptation strategy from a family of bounded inference-time transformations:

$$\alpha^* = \arg\max_{\alpha \in \mathcal{A}} \mathbb{E}_{\alpha \sim \pi_\phi(\cdot|\tau)}[R(\tau, \alpha)] \,, \tag{4}$$

where $\mathcal{A} = \{\texttt{TTL}, \texttt{LoRA}, \texttt{TTS}, \texttt{Latent}\}$ contains four strategy families:

- **Test-Time Learning (TTL):** Adapts parameters by minimizing input perplexity on test samples.
- **Two-Subspace LoRA Mixing:** Enables cross-subspace interaction through butterfly mixing factors.
- **Test-Time Scaling (TTS):** Aggregates predictions across multiple prompt batches via ensembling.
- **Latent Space Modification:** Applies per-sample additive parameters to the final hidden layer.

Observation and Action Spaces. The Policy Agent's observation τ is a natural-language task description (generated from few-shot examples). Its action is a structured JSON object specifying the strategy family and hyperparameters - a concrete instance of natural language serving as the observation space with structured outputs as the action space. The agent is trained via ReST$^{\text{EM}}$ (Singh et al., 2024) with binary reward: $R = 1$ if the selected strategy improves task performance, $R = 0$ otherwise. Full details provided in Appendix E.

3 Experiments

Setup. We evaluate CORAL on 12 benchmarks: 6 in-domain (ARC-c, ARC-e, HellaSwag, BoolQ, PIQA, WinoGrande) and 6 out-of-domain (GSM-8K, MATH, DivLogicEval, SocialIQA, CodeMMLU, JAMA Clinical Challenge), using Qwen2.5-Instruct at 0.5B and 1.5B parameters. Baselines include No Meta-Train LoRA which performs task-specific LoRA fine-tuning without meta-training; Union Train LoRA, which trains a single LoRA adapter on the union of all training tasks, MAML-en-LLM (Sinha et al., 2024) and ABMLL (Zhang et al., 2025), both monolithic bi-level meta-learning approaches, plus standard LoRA fine-tuning variants.

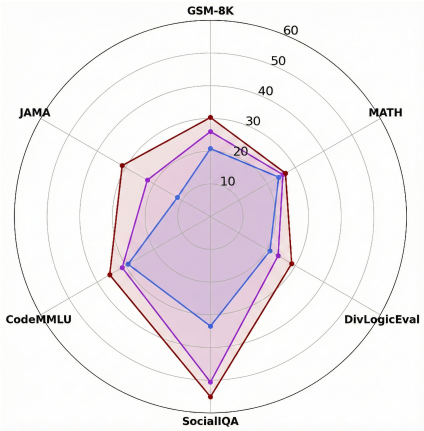

Figure 1: Agent contribution analysis. Each level adds agents cumulatively with Generator Agent, Alignment Agent and Policy Agent

Main Results. Table 2 shows that CORAL's multi-agent coordination consistently outperforms monolithic baselines. On the 1.5B model, CORAL achieves 67.7% average in-domain accuracy (+2.2 over MAML-en-LLM) and 47.5% out-of-domain (+5.2 over MAML-en-LLM), with the largest gains in domains requiring distinct reasoning patterns (GSM-8K: +15.8,

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

Out-of-Domain Tasks														
Model Size	**Qwen2.5-0.5B-Instruct**							**Qwen2.5-1.5B-Instruct**						
Method	GSM	MATH	DivL	SIQA	Code	JAMA	**Avg**	GSM	MATH	DivL	SIQA	Code	JAMA	**Avg**
Base Model	15.2	2.8	22.4	50.8	32.4	23.8	24.5	51.8	30.3	28.3	65.9	42.6	38.9	42.9
Union Train	15.6	6.8	20.3	39.5	29.8	29.9	29.9	34.2	32.2	24.1	51.4	34.7	34.7	36.1
ABMLL	20.4	7.1	23.7	53.1	28.2	16.8	24.9	28.7	15.9	26.9	66.3	39.6	28.5	34.3
MAML-en-LLM	29.1	**26.3**	25.1	54.9	34.1	26.4	32.6	35.6	43.5	31.2	68.7	42.3	32.5	42.3
CORAL	**30.3**	24.7	**28.7**	**55.1**	**35.6**	**31.2**	**34.2**	**51.4**	**46.9**	**31.4**	**69.5**	**44.6**	**41.5**	**47.5**

Table 2: Comparison of 0.5B and 1.5B model performance on In-Domain and OOD benchmarks, separated by a vertical demarcation.

JAMA: +9.0). The 0.5B model shows similar trends with +6.5 in-domain and +1.6 out-of-domain improvements.

Agent Contribution Analysis. Figure 1 isolates each agent's contribution. The Generator Agent alone ("Gen. only") provides substantial improvement over the base model - confirming effective generative production of diverse adapters. Adding the Alignment Agent ("+Align") further improves performance, particularly on out-of-domain tasks where semantic bridging is critical. The full three-agent system ("+Policy") achieves the best results, with the Policy Agent's RL-based strategy selection providing the largest out-of-domain gains.

Policy Agent Strategy Diversity. The Policy Agent learns task-appropriate strategies without explicit programming: TTL is selected for domain-specific tasks (medical, physical reasoning), LoRA Mixing for structured reasoning (math, boolean logic), TTS for adversarial/multi-interpretation tasks (HellaSwag, CodeMMLU) and Latent modification for abstract reasoning (ARC, DivLogicEval). This emergent specialization demonstrates effective RL-based orchestration within the multi-agent system.

4 Conclusion

We present CORAL, a cooperative multi-agent framework that decomposes LLM adaptation into three specialized agents - a Generator, an Alignment Agent and a Policy Agent - that coordinate through minimal information exchange. This perspective reframes LLM adaptation from a monolithic optimization problem to a multi-agent coordination challenge, yielding consistent improvements across 12 diverse benchmarks. Our work demonstrates that the convergence of multi-agent learning and generative AI offers practical benefits for robust LLM adaptation and we hope it stimulates further exploration of multi-agent architectures for foundation model specialization.

REFERENCES

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.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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 2025. 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/.

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.

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.

David Ha, Andrew Dai, and Quoc V Le. HyperNetworks. *arXiv preprint arXiv:1609.09106*, 2016.

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.

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.

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.

Shuo Liu, Tianle Chen, Zeyu Liang, Xueguang Lyu, and Christopher Amato. Llm collaboration with multi-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2508.04652.

Hao Ma, Tianyi Hu, Zhiqiang Pu, Boyin Liu, Xiaolin Ai, Yanyan Liang, and Min Chen. Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2410.06101.

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.

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.

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.

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

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

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.

APPENDIX NAVIGATION INDEX

Section	Content	Pg Rf
Reproducibility & Experimental Validation		
B	**Notation and Symbols** - Comprehensive reference table for all mathematical symbols, notations, and hyperparameters used throughout the paper	7
Theoretical Foundations & Related Work		
A	**Related Work** - Comprehensive review of bi-level meta-learning, self-play evolution, model fusion, parameter-efficient fine-tuning, and test-time training	7
Implementation Details		
C	**Stage 1: Catching** - LoRA checkpoint collection protocol, parameter tokenization, hyper-convolutional decoder architectures, hardware specifications	7
D	**Stage 2: Adapting** - Fusion VAE architecture, training procedure, inference pipeline, algorithmic formulation	11
E	**Stage 3: Operating** - Detailed specifications for RL training, Test-Time Learning, LoRA mixing, Test-Time Scaling and Latent Space Modification	13
Experimental Configuration		
E.6	**Strategy Generation Prompts** - Complete JSON schemas and prompting templates for RL-based strategy selection	18
F	**Baseline Hyperparameters** - Full hyperparameter specifications for MAML-en-LLM and ABMLL baselines ensuring fair comparison	22
C.7	**Dataset Descriptions** - Detailed characterization of all in-domain and out-of-domain evaluation benchmarks	11

A RELATED WORK

Multi-Agent Systems for LLMs. Recent work has explored multi-agent architectures for LLM collaboration. MAGRPO Liu et al. (2025) models LLM collaboration as cooperative MARL, enabling multi-turn collaboration through group relative policy optimization. CORY (Ma et al., 2025) extends RL fine-tuning to sequential cooperative multi-agent settings by duplicating LLMs into pioneer-observer pairs. Unlike these approaches, CORAL uses heterogeneous agents with distinct architectures (convolutional generator, VAE, LLM policy) rather than homogeneous LLM copies, targeting the *adaptation* problem specifically.

Monolithic LLM Adaptation. Methods like MAML (Finn et al., 2017), MAML-en-LLM (Sinha et al., 2024), and ABMLL (Zhang et al., 2025) employ nested optimization 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). CORAL offers a principled alternative through agent specialization and modular coordination.

B NOTATION AND SYMBOLS

Table 3 provides a comprehensive reference for all mathematical notation used in this paper.

C GENERATOR AGENT

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

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
Generator Agent	
$\mathcal{G}_\phi, \phi, \phi^*$	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
$k, I, K[i], k^j_i, e^j_i$	Token size; #layers; tokens; positional embed
Alignment Agent	
$\mathbf{v}_{\mathcal{T}_*} = \mathbf{z}^{\text{ft}} - \mathbf{z}^{(0)}$	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
Policy Agent	
$\mathcal{A} = \{\texttt{TTL}, \texttt{LoRA}, \texttt{TTS}, \texttt{Latent}\}$	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
$W_{\text{LM}}, \lvert V\rvert, d, n$	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 3: Notation reference for mathematical symbols and hyperparameters.

C.1 Prompt Encoding Tensor Structure

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

$$\begin{aligned} \mathbf{c}_i &= \text{Encoder}(p_i; \theta_{\text{enc}}), \\ \mathbf{C} &= [\mathbf{c}_1; \ldots; \mathbf{c}_N] \in \mathbb{R}^{B \times N \times L \times C} \end{aligned} \tag{5}$$

Here:

- B denotes the batch size (number of tasks processed in parallel),
- N is the number of representative prompts sampled per task,

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 4: Checkpoint collection settings for LoRA parameter generation

- 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.

C.2 LoRA Checkpoint Collection Protocol

CORAL 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 4 summarizes the checkpoint collection settings for different task families.

C.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,

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:

$$\begin{aligned} W &\xrightarrow{\text{split by layer}} [w[1], \ldots, w[I]] \\ &\xrightarrow{\text{normalize}} [\hat{w}[1], \ldots, \hat{w}[I]] \end{aligned} \tag{6}$$

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):

$$\hat{w}[i] \xrightarrow{\text{tokenize}} K[i] = \left[k_i^1, k_i^2, \ldots, \text{pad}\left(k_i^{J_i}\right)\right], \tag{7}$$

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

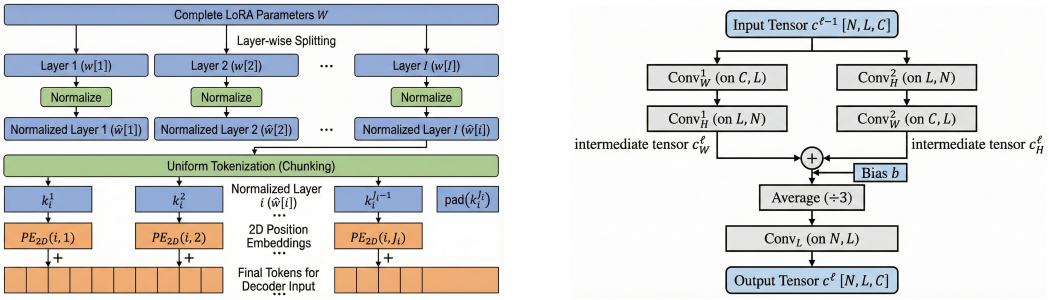

Figure 2: (a) Tokenization of LoRA parameters *(left)* and (b) Generator Agent Architecture *(right)*

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

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.

C.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 5 lists the decoder architectures used for different foundation model sizes.

C.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-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.

C.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.

Foundation Consolidation (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 4.

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 5.

C.7 Dataset Descriptions

We evaluate CORAL 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 foundation consolidation 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 foundation consolidation stage (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 foundation consolidation 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., 2025) consists of expert-curated medical case questions with detailed explanations, designed to evaluate clinical reasoning and decision-making.

D Alignment Agent

D.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

D.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, Stride: 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.

D.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:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{recon}} + \lambda_{\text{KL}} \cdot \mathcal{L}_{\text{KL}} \tag{8}$$

D.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.

D.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 foundational knowledge consolidated in Stage 1.

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.

E Policy Agent

This section provides a comprehensive treatment of the four Policy Agent of CORAL: 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.

Refinement is cast as a decision-making problem:

$$\alpha^* = \arg\max_{\alpha \in \mathcal{A}} \mathbb{E}_{\alpha \sim \pi_\phi(\cdot|\tau)}[\, R(\tau, \alpha)\,], \tag{9}$$

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.

E.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:

$$\mathcal{P}(x; \Theta) = \exp\Big(-\frac{1}{T}\sum_{t=1}^{T} \log p(x_t \mid x_{1:t-1}; \Theta)\Big). \tag{10}$$

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

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 $f_{\Theta+\Delta\Theta}$

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

$$S(x) = \lambda \cdot \exp\big(\log \mathcal{P}(x; \Theta) - \log \mathcal{P}_0\big)\, \mathbb{I}_{\{\mathcal{P}(x;\Theta) > \mathcal{P}_0\}}(x), \tag{11}$$

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:

$$\min_{\Delta\Theta} \; S(x)\, \mathcal{P}(x; \Theta + \Delta\Theta). \tag{12}$$

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
}
```

E.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.

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:

$$x\mathbf{W}_0 + x\Delta\mathbf{W} = x\mathbf{W}_0 + x\mathbf{A}\mathbf{B}, \tag{13}$$

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:

$$x\mathbf{A}\mathbf{B} = x \sum_{i=1}^{r} \mathbf{A}_i \mathbf{B}_i = x\mathbf{A}\mathbf{I}_{r\times r}\mathbf{B}. \tag{14}$$

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

$$x \sum_{i=1}^{r/2} (\mathbf{A}_i + \mathbf{A}_{i+r/2})(\mathbf{B}_i + \mathbf{B}_{i+r/2}) = x\mathbf{A} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{I} \end{bmatrix} \mathbf{B}. \tag{15}$$

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)
}
```

E.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.

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>"
}
```

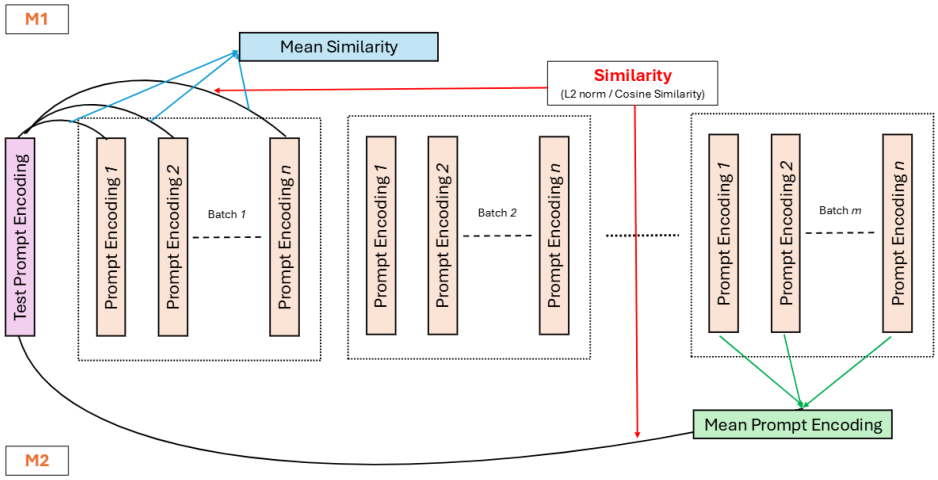

Figure 3: **TTS Router Architecture.** M1 computes mean similarity scores between the test prompt encoding and individual prompt encodings within each batch. M2 computes similarity between the test encoding and mean prompt embeddings per batch.

E.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:

$$H' = H + \delta, \qquad \text{logits} = W_{\text{LM}}(H + \delta). \tag{16}$$

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

$$\mathcal{L}(\delta) = -\sum_{i=1}^{n-1} \log p(x_{i+1} \mid x_{1:i}, \delta). \tag{17}$$

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:

$$x_{\text{next}} \sim \text{softmax}\big(W_{\text{LM}}(H_{\text{last}} + \delta_{\text{opt}})\big). \tag{18}$$

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>,
```

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

```
  // optimization steps (T)
  "slot_lr": <float>
  // learning rate (eta)
}
```

E.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) $\mathcal{D}$, CORAL requires, 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, CORAL 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_{θ_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`}.

E.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.
```

F 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 CORAL. 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.