NAVIGATING THE COST-PERFORMANCE PARETO FRONTIER OF TEST-TIME LLM AGENT ADAPTATION

Konrad Szafer[*1], **Xiaozhe Yao**[*1], **Maximilian Böther**[1], **Gregor Bachmann**[2], **Tiago Pimentel**[1], **Ana Klimovic**[1]
[1]ETH Zurich [2]Apple

ABSTRACT

Large language model (LLM) agents deployed in real systems often face distribution shift relative to training. This motivates test-time adaptation: improving agents' behavior after deployment using verifiable feedback (e.g., binary correctness signals or unit tests) rather than ground-truth training data. The design space for improving test-time performance is broad: it includes scaling inference compute (e.g., longer reasoning) and test-time adaptation methods that update context, external memory, or model parameters. We present a unified empirical characterization of test-time adaptation for LLM agents with verifiable feedback, and we measure adaptation compute in wall-clock time. We compare approaches from in-context adaptation (e.g., updating context or external memory) and fine-tuning via reinforcement learning under a data budget while tracking compute. We find that adaptation helps most when tasks benefit from improved reasoning using the model's existing knowledge and skills, but offers limited gains when tasks demand learning new facts from the adaptation data. We summarize results via an accuracy vs. adaptation compute Pareto frontier showing the efficiency trade-offs across methods. By making these efficiency trade-offs explicit, this frontier helps practitioners choose suitable adaptation method for their deployment scenario.

1 INTRODUCTION

Once deployed, LLM agents routinely face distribution shifts relative to their training data. For example, a coding agent inside a company must operate over private codebases, conventions, and tooling that differ from the public corpora it was trained on. Because such environments evolve after deployment and often provide verifiable feedback (e.g., unit tests, compilers, or formal checks), a natural goal is **test-time adaptation**: improving an agent's behavior *after deployment* using limited data and compute. In parallel, recent work on test-time scaling suggests that allocating more compute during inference for extended reasoning can also improve performance at test-time, often improving accuracy at the cost of latency and tokens (Jaech et al., 2024; Guo et al., 2025).

We identify the following approaches in prior work: i) **In-context learning** (Brown et al., 2020; Wei et al., 2025) adapt an agent's behavior by the mechanism of in-context learning: the model conditions on examples, instructions, and feedback provided in the context, without changing model weights. In practice, this is often combined with scaffolding, such as retrieval and external memory, that builds the context over time by selecting and inserting relevant past experiences for each query. ii) **In-weights learning** (Zuo et al., 2025; Wang et al., 2025) adapts an agent's behavior by optimizing model parameters (fine-tuning), which can produce persistent behavioral change across tasks. These families of methods have different accuracy–cost trade-offs, yet no existing study systematically compares them under matched data budgets while tracking adaptation compute. The above approaches are used together with test-time scaling.

We study a spectrum of adaptation mechanisms, from *in-context* methods augmented with retrieval and external memory to *in-weights* methods–LoRA and full fine-tuning–and evaluate them under the same setup.

[*]Equal contribution. Correspondence to konrad.szafer@inf.ethz.ch and xiaozhe.yao@inf.ethz.ch.

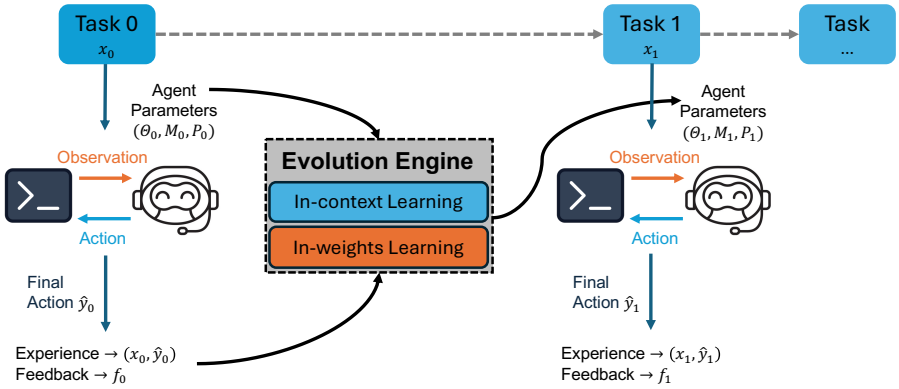

Figure 1: **Test-time adaptation and evaluation across a task stream.** At each time step t, the agent (with model weights θ_t, external memory M_t, and prompt P_t) receives input x_t and produces a prediction $\hat{y}_t$, yielding experience $\exp_t = (x_t, \hat{y}_t)$. The prediction is evaluated against the ground-truth label y_t, resulting in binary feedback $f_t \in \{0, 1\}$. Performance is tracked cumulatively over the stream using the agent's pre-update answers. After receiving feedback, an *evolution engine* updates the agent state via *in-context learning* or *in-weights learning*, producing $(\theta_{t+1}, M_{t+1}, P_{t+1})$ for the next task.

2 Background

Consider a language model agent deployed to assist users, for instance a coding agent operating over a private codebase. The agent has no access to ground-truth solutions at test time; instead, it can only rely on limited *feedback*, such as whether code passes a unit test, a user accepting or rejecting a response, or rankings over candidate outputs. Given this constrained feedback, how should the model adapt? Prior work falls into two broad families of approaches, detailed below.

In-context Learning Methods. These methods adapt behavior without modifying model parameters. Instead, they learn from the provided context within the model's activations. Methods utilize this mechanism by accumulating experiences in an external memory and injecting relevant entries into the prompt at inference time, allowing the model to adjust its responses. Examples include *ExpRAG* and *ReMem* (Wei et al., 2025). *ExpRAG* encodes structured feedback using a template S, retrieving k relevant experiences from memory to condition response generation. *ReMem* extends this by incorporating a memory reasoning module, which allows the agent to evaluate, reorganize, and evolve its internal memory during task execution. However, memory-based methods are fundamentally constrained by the context window: as the experience store grows, retrieval quality degrades, and prompt length inflates inference cost, limiting their scalability.

In-weights Learning Methods. In contrast to in-context adaptation, in-weights learning approaches adapt by optimizing the model online from scalar feedback. Because updates are written directly into the model weights, fine-tuning can offload recurring patterns from the context window into persistent model knowledge, maintaining a compact context regardless of how much experience has been accumulated. Reinforcement Learning with Verifiable Rewards (RLVR) enables LLMs to learn from automatically verifiable feedback, such as unit test outcomes or exact matches. More broadly, weight updates can also be driven by supervised fine-tuning (SFT) or preference-based objectives such as DPO, but these typically require curated training data; here, we focus on verifiable feedback to enable online learning without additional data preparation. In our setup, we instantiate in-weights adaptation with two methods: *full fine-tuning* and *LoRA* (Hu et al., 2022). In our experiments, both methods optimize the model online using Group Relative Policy Optimization (GRPO) (Shao et al., 2024; Guo et al., 2025), differing mainly in what parameters are updated (all weights vs. low-rank adapter weights).

3 EXPERIMENTAL SETUP

Test-Time Adaptation Protocol. Traditional LLM benchmarks evaluate static models on static test sets, scoring each example independently. Deployed agents, however, encounter tasks *sequentially* in a streaming fashion, and can potentially update their behavior based on the feedback over time. To measure this ability, the key change is the *evaluation protocol*: performance needs to be measured *over the sequence of tasks* using the agent's pre-update predictions, reporting accuracy computed before receiving the feedback. This setup ensures the model does not have access to the feedback or the ground truth before each prediction.

We adopt the streaming evaluation protocol introduced by Evo-Memory (Wei et al., 2025). Given a static labeled dataset $D = \{(x_i, y_i)\}_{i=1}^{|D|}$, we construct a *task stream* τ by choosing an ordering π over examples and defining $\tau = ((x_{\pi(1)}, y_{\pi(1)}), \ldots, (x_{\pi(T)}, y_{\pi(T)}))$ with $T = |D|$. Each time step corresponds to one dataset example, and we run a single pass over τ. We use the default ordering provided by the Hugging Face datasets as π and keep it fixed across all evaluations.

At each time step t, the agent, parameterized by model weights θ_t, external memory M_t, and a prompt template P_t, receives input x_t and produces a prediction $\hat{y}_t$. We evaluate this prediction before any feedback or adaptation step, reflecting real-world deployment where labels are initially unknown. The environment then reveals the ground-truth label y_t, providing feedback in the form of a verifiable reward signal. Conditioned on this signal, the agent updates its state via in-context learning, memory operations, and/or weight updates, yielding $(\theta_{t+1}, M_{t+1}, P_{t+1})$ for the next task (Figure 1). While Wei et al. (2025) updates after every sample, we generalize it to perform single gradient-based updates every N labeled examples (i.e., using a fixed batch size $N = 4$ throughout this study), which stabilizes fine-tuning. For methods without gradient-based weight updates (e.g., in-context learning), the stream is still processed in the same N-step chunks for consistency. This is not a concern in the original setup, as it does not perform gradient-based weight updates.

Tasks and Datasets. We run streaming evaluation on seven benchmarks spanning mathematical reasoning and knowledge-intensive domains: AIME-24, AIME-25 (competition mathematics requiring symbolic reasoning), GPQA-Diamond (graduate-level science) (Rein et al., 2023), and four MMLU-Pro (Wang et al., 2024) subsets (Economics, Philosophy, Biology, Health). Each benchmark is converted into a streaming task sequence following the protocol in Section 3.

Models and Infrastructure. We experiment with two open-weight reasoning LLMs from different model families: Qwen3-8B (Yang et al., 2025) and Olmo3-7B (Olmo et al., 2025). Both have been post-trained for multi-step reasoning but differ in pre-training data, architecture, and post-training recipes. Full hyperparameter details are provided in Appendix B. Implementation and infrastructure details can be found in Appendix C.

4 RESULTS

Table 1 and Fig. 2 summarize the accuracy–compute trade-offs across benchmarks, adaptation mechanisms, and backbones. Overall, we observe a cost–performance Pareto frontier: (i) in-weights learning methods deliver the largest accuracy gains when the distribution shift requires acquiring new reasoning strategies, while (ii) in-context learning methods can reduce end-to-end time by shortening generations through reuse of reasoning patterns from memory. The frontier becomes task-dependent once we distinguish whether adaptation requires learning *new reasoning* vs. acquiring *missing facts*.

Two kinds of test-time adaptation. We find that our benchmarks fall into two qualitatively different settings. Some primarily require improved problem-solving strategies using knowledge already present in the model (e.g., AIME24/25). Others require reasoning while also relying on domain facts the model may not have learned during training (e.g., GPQA and MMLU-Pro). In the latter case, adaptation from verifiable feedback is often bottlenecked by missing factual knowledge, so updates mostly refine behavior and strategy rather than adding new facts, which translates to very limited improvements.

On *mathematical benchmarks*, particularly on AIME-24 and AIME-25 problems that demand novel problem-solving strategies rather than factual recall, in-weights learning methods achieve the largest

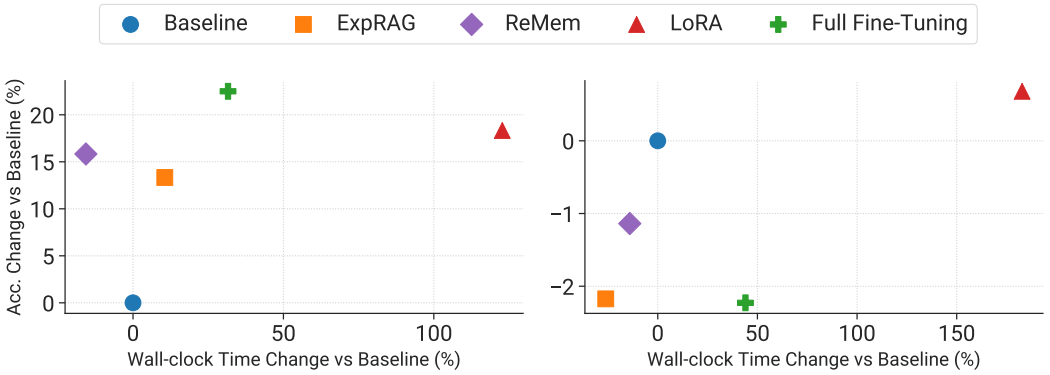

Figure 2: **Relative change in accuracy and wall-clock time vs. the *Baseline* (Qwen3-8B Thinking).** Left: reasoning-heavy mathematical benchmarks (averaged over AIME24 and AIME25); right: knowledge-intensive benchmarks (averaged over four MMLU Pro subsets: Economics, Philosophy, Biology, Health and GPQA). In-weights learning methods (LoRA, full fine-tuning) consistently improve accuracy on math tasks at the cost of additional compute, whereas on knowledge-demanding domains, the gains from gradient updates diminish.

gains: Qwen3-8B, both LoRA and full fine-tuning improve AIME24 from 0.536 to 0.642 (+10.6 pp), and LoRA improves AIME25 from 0.429 to 0.500 (+7.1 pp) while full fine-tuning reaches 0.464 (+3.5 pp). These gains align with the intuition that reinforcement learning fine-tuning internalizes procedural reasoning patterns into the model's weights, making them persistently available for subsequent problems without consuming context-window capacity.

On *knowledge-heavy* MMLU-Pro subsets we do not observe consistent improvements from any adaptation method. This is also visible in the streaming view for MMLU-Pro Economics (Fig.4), where methods track similar cumulative accuracy but separate more clearly in wall-clock time as adaptation progresses. These findings align with the intuition that RL-based fine-tuning cannot teach the model facts it never learned during pre-training (Yue et al., 2025).

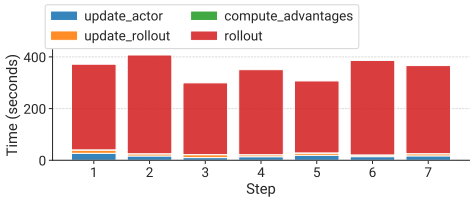

Figure 3: Timing Breakdown per Step for full fine-tuning Qwen3-8B model.

In-context learning based methods exhibit a complementary profile. Retrieved experiences tend to shorten generation length, often resulting in wall-clock times at or below the no-adaptation baseline despite the overhead of retrieval and longer context, while accuracy impact is mixed. That said, *ExpRAG* frequently underperforms both the *Baseline* and *ReMem*, suggesting that naively prepending raw experience tuples to the prompt can confuse the model when retrieved examples are semantically distant from the current problem. *ReMem's* active memory curation, which prunes irrelevant entries and reorganizes retained ones, appears to mitigate this failure mode.

Finally, the overhead of in-weights learning methods turns out to be more modest than one might expect. In our setting, forward-pass generation dominates end-to-end time, not backpropagation. Fig 3 demonstrates that for full fine-tuning, the backward pass (update actor) accounts for only a small fraction of the total time per step, while the forward pass (rollout) dominates. Note that LoRA's higher wall-clock times relative to full fine-tuning in Table 1 reflect suboptimal adapter merge-and-reload in our serving infrastructure, not an inherent cost of the method.

5 Discussion and Future Work

In this work, we focus on adapting LLM agents post-deployment and evaluating their quality on a stream of tasks. In practical deployments, agents are expected to retain proficiency in prior tasks

LLM	Method	AIME24		AIME25		GPQA		MMLU-Pro (Eco.)		MMLU-Pro (Philo.)		MMLU-Pro (Bio.)		MMLU-Pro (Health)		Avg. Acc.↑
		Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓	
Qwen3 8B	Baseline	0.536	0.494	0.429	0.515	0.520	4.966	**0.397**	3.000	0.345	1.249	0.393	2.187	0.332	2.387	0.421
	ExpRAG	0.500	0.606	0.429	0.544	0.473	3.780	0.390	2.216	0.351	0.847	**0.395**	1.701	**0.346**	1.575	0.412
	ReMem	0.571	0.447	0.357	0.418	0.515	3.407	0.385	2.344	0.345	1.109	0.391	2.369	0.339	1.798	0.414
	LoRA	**0.642**	1.198	**0.500**	1.204	**0.531**	12.086	0.376	11.359	**0.357**	3.290	0.392	6.328	0.340	5.260	**0.448**
	Full FT	**0.642**	0.634	0.464	0.739	0.489	6.769	0.380	4.811	0.353	1.957	0.389	2.684	0.341	4.124	0.436
Olmo3 7B	Baseline	0.500	0.624	0.357	0.640	0.288	9.814	0.328	9.599	0.270	4.058	0.353	5.737	**0.280**	5.249	0.339
	ExpRAG	0.464	0.719	0.286	0.692	0.308	9.198	0.313	8.604	0.244	4.221	**0.354**	5.686	0.251	6.011	0.317
	ReMem	**0.571**	0.453	0.428	0.610	**0.366**	7.892	0.296	8.290	0.254	3.732	0.328	6.774	0.263	6.674	0.358
	LoRA	**0.571**	2.129	**0.464**	1.923	0.319	20.769	0.334	8.049	**0.284**	12.184	0.336	18.126	0.269	15.675	**0.368**
	Full FT	0.536	1.011	0.393	1.590	0.321	13.953	**0.332**	14.525	0.262	6.074	0.346	8.697	0.267	10.151	0.351

Table 1: Cross-dataset results on single-turn reasoning and QA benchmarks across LLM backbones. Each benchmark reports final cumulative Accuracy (Acc.↑) and end-to-end wall-clock time in hours (Time↓), measured for running the full benchmark (evaluation + adaptation) on a 4×NVIDIA GH200. The last column (Avg. Acc.↑) reports the average across all per-benchmark accuracy columns. Horizontal rules separate the *Baseline* (the backbone queried as-is), in-context learning approaches (*ExpRAG/ReMem*), and in-weights learning methods (*LoRA* and *full fine-tuning*).

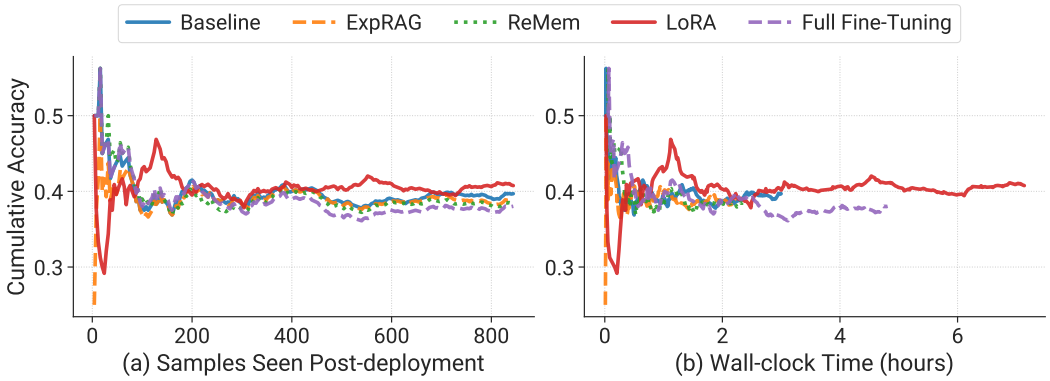

Figure 4: **MMLU-PRO (Eco.) (Qwen3-8B Thinking):** Cumulative average accuracy vs. verified samples (left) and vs. wall-clock time (right). On this knowledge-heavy task, in-weights learning methods achieve accuracies similar to the *Baseline* but take considerably longer.

while learning new ones. Therefore, it is important to also quantify the degree of catastrophic forgetting across various adaptation methods and analyze the inherent trade-off between retention and adaptation. We also plan to extend our evaluation to more agentic applications, including coding agents and multi-turn problem solving, where adaptation must operate over longer horizons.

Our empirical results show that post-deployment adaptation significantly improves math task performance but struggles with knowledge-heavy problems. This performance gap likely stems from low inter-sample fact correlation in our datasets, which limits transfer learning. Efficient knowledge acquisition in these domains may require higher sample efficiency than current methods provide. Future work will compare distillation-based in-weights adaptation–where a teacher model provides training targets Hinton et al. (2015); Yue et al. (2025)–against methods evaluated in this work.

From a systems perspective, scaling post-deployment adaptation introduces distinct architectural trade-offs. In-context learning methods are readily deployable using existing model serving infrastructure, as they only require managing the prompt content without modifying the model itself, and are easily extendable with approaches like retrieval-augmented generation. Conversely, in-weights learning methods require specialized infrastructure for online learning Böther et al. (2025), model versioning Hao et al. (2023), and zero-downtime updates Baylor et al. (2017). We intend to investigate these scaling challenges to develop systems for LLM agent adaptation.

REFERENCES

Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pp. 1387–1395, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi:10.1145/3097983.3098021. URL https://doi.org/10.1145/3097983.3098021.

Maximilian Böther, Ties Robroek, Viktor Gsteiger, Robin Holzinger, Xianzhe Ma, Pınar Tözün, and Ana Klimovic. Modyn: Data-centric machine learning pipeline orchestration. *Proceedings of the ACM on Management of Data*, 3(1):1–30, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, and Xiao Bi et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Wei Hao, Daniel Mendoza, Rafael da Silva, Deepak Narayanan, and Amar Phanishaye. Mgit: A model versioning and management system, 2023. URL https://arxiv.org/abs/2307.07507.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

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, 2022.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, and Alex Carney et al. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of

mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Chenmien Tan, Simon Yu, Lanbo Lin, Ze Zhang, Yuanwu Xu, Chenhao Jiang, Tianyuan Yang, Sicong Xie, and Guannan Zhang. Rl2: Ray less reinforcement learning. https://github.com/ChenmienTan/RL2, 2025. GitHub repository.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Reinforcement learning for reasoning in large language models with one training example, 2025. URL https://arxiv.org/abs/2504.20571.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory, 2025. URL https://arxiv.org/abs/2511.20857.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. Ttrl: Test-time reinforcement learning, 2025. URL https://arxiv.org/abs/2504.16084.

A Cumulative average reward on more datasets

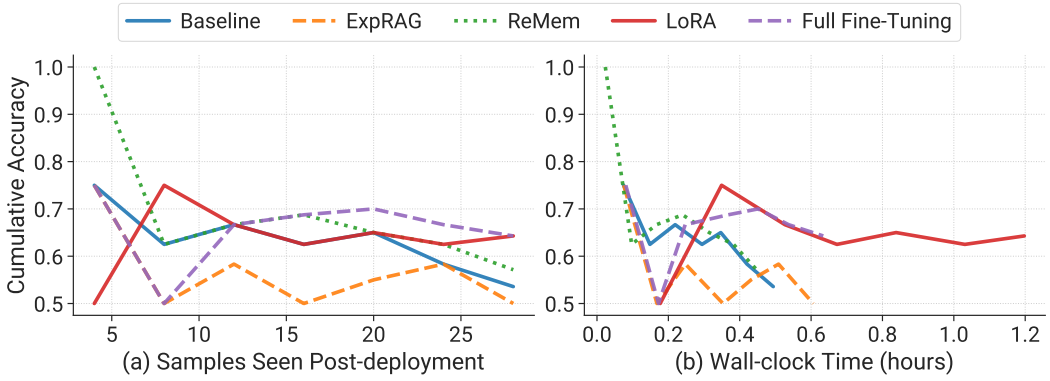

Figure 5: AIME 2024 (Qwen3-8B Thinking): Cumulative average reward (accuracy) vs. verified samples (left) and vs. adaptation compute / wall-clock time (right).

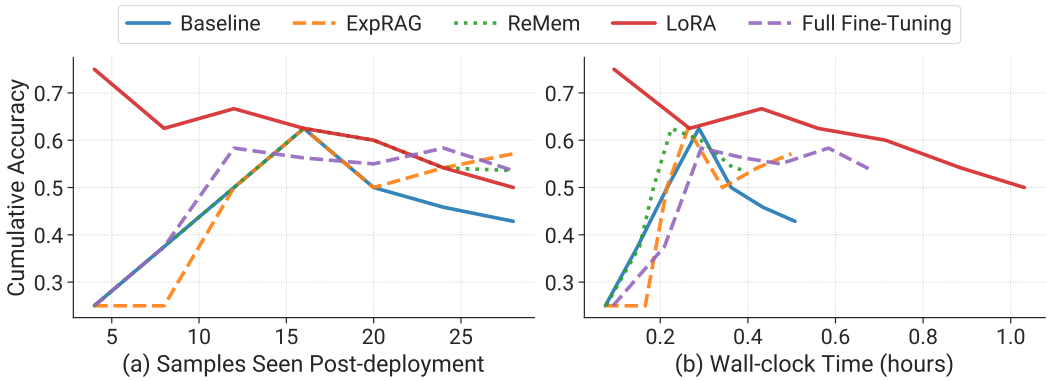

Figure 6: AIME 2025 (Qwen3-8B Thinking)

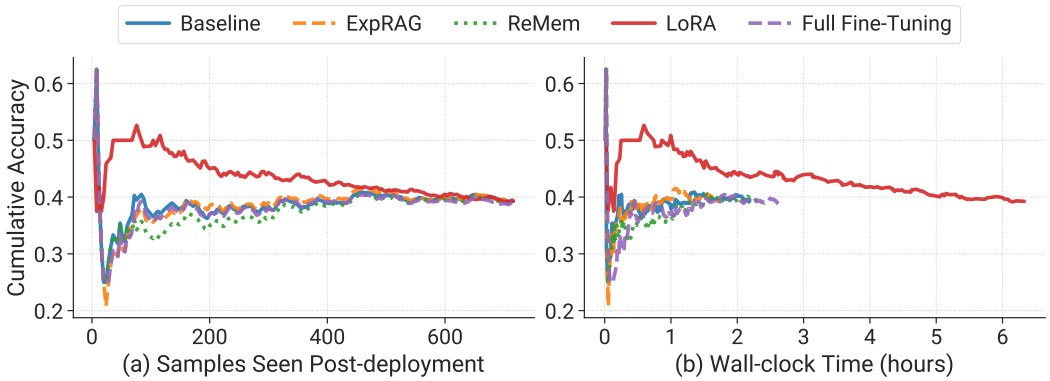

Figure 7: MMLU-PRO (Bio.) (Qwen3-8B Thinking)

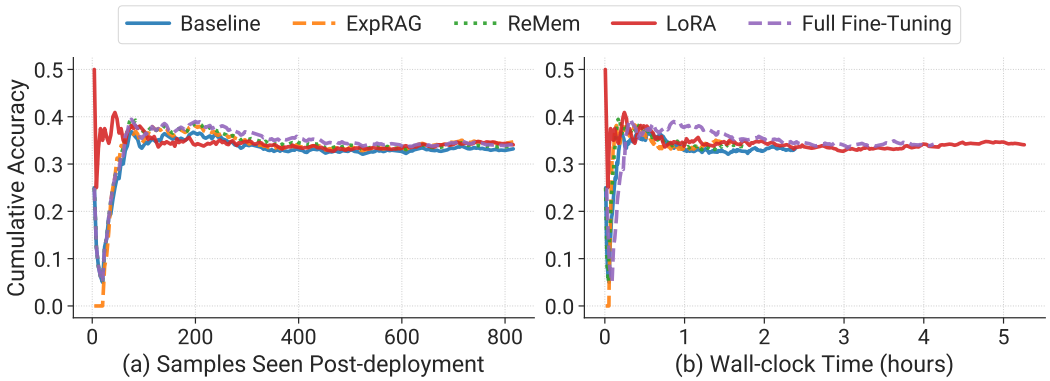

Figure 8: MMLU-PRO (Health.) (Qwen3-8B Thinking)

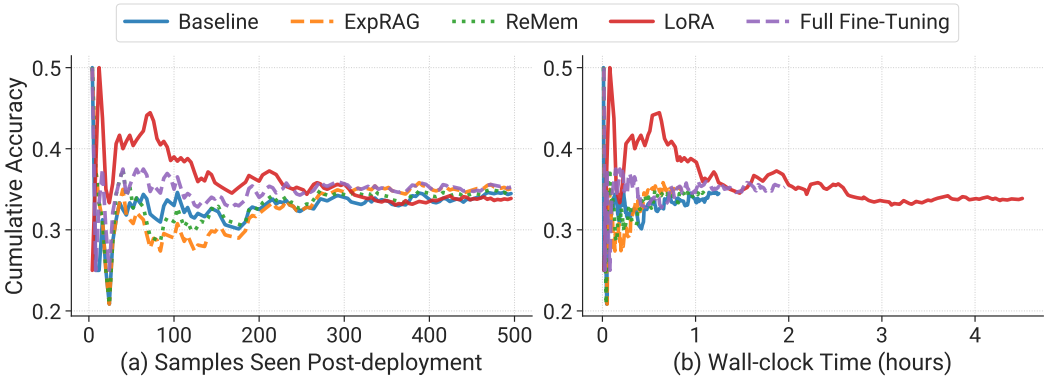

Figure 9: MMLU-PRO (Philo.) (Qwen3-8B Thinking)

B HYPERPARAMETERS

General. Across all methods, we limit the maximum generation length to 16,384 tokens and sample with a temperature of 0.6. Inference, evaluation, and adaptation updates are performed with a batch size of 4.

GRPO. We use a group size of 4, generating four candidate responses per prompt for advantage estimation. Advantages are computed via within-group reward normalization as described in Background.

LoRA. We apply low-rank adaptation with rank $r = 128$ and scaling factor $\alpha = 256$ to all linear layers, using a learning rate of 5×10^{-5}.

Full Fine-Tuning. All model parameters are updated using the same GRPO objective with a learning rate of 1×10^{-5}.

C INFRASTRUCTURE

All experiments run on a single node with 4 NVIDIA GH200 superchips, and we report both accuracy and end-to-end wall-clock time (evaluation + adaptation) to reflect end-to-end cost. We modified RL2 (Tan et al., 2025), an open-source library for RL, as the unified framework for both memory methods and GRPO fine-tuning. We use SGLang (v0.5.7) for model serving. Since the memory methods as described in Evo-Memory (Wei et al., 2025) are not open sourced yet, we implement them following the descriptions in the paper. Memory-only methods follow the same execution path as parameter-updating methods but skip the backward/optimizer step. A key design choice is that the same 4 GPUs are used for both inference and training, following a hybrid-engine approach. During the rollout phase, SGLang serves the model with tensor parallelism across all GPUs (with a tensor parallelism degree of 4), handling generation requests through an asynchronous router that distributes work across the tensor-parallel workers. When the rollout phase completes and training begins, the system first pauses generation and waits for all in-flight requests to drain from the scheduler. It then releases the KV cache and, for fine-tuning, also the serving copy of the model weights and adapters from GPU memory, freeing capacity for the training phase. The model is then loaded under FSDP (ZeRO-3) sharding across the same GPUs, and gradient updates are performed using GRPO with group-relative advantage normalization. After the backward pass, the updated parameters (PEFT adapters or full weights, depending on the method) are serialized and streamed back to the SGLang workers to replace serving state, the KV cache is reallocated, and generation resumes. This colocated design avoids the need for dedicated training and serving GPU pools, which is particularly relevant for the resource-constrained deployments where test-time adaptation is most practical, though it does mean that training and inference cannot overlap.