Extended Abstract

Motivation The paradigm of scaling language models has traditionally focused on increasing parameter counts, dataset sizes, and training compute. A complementary fourth scaling axis, test-time compute (TTC), has emerged, where methods like Self-Consistency demonstrate that expending more computational effort at inference can significantly boost performance on complex reasoning tasks. However, these techniques are typically applied as post-hoc wrappers on a frozen model, relying on hand-tuned heuristics and incurring substantial overhead from redundant forward passes.

Method This work explores integrating TTC directly into the training loop. We propose an extension to REINFORCE Leave-One-Out (RLOO), an online policy gradient algorithm, which augments the reward function to explicitly model the trade-off between answer correctness and computational cost, measured in the number of reasoning traces. By training with on-policy sampling and a reward function that explicitly balances correctness and efficiency, our model learns to internalize the decision of how much effort to expend. The augmented reward is formulated as $R_{\text{aug}}(y, k, c) = R_{\text{base}}(y) + \lambda_c \cdot c - \lambda_k \cdot k$, where c is correctness and k is compute cost.

Implementation We use the `Qwen/Qwen2.5-0.5B` model as our foundation. After an initial Supervised Fine-Tuning (SFT) phase on high-quality data, we train the model using our RLOO-Turbo objective. During on-policy rollouts, each generated response is scored for correctness and cost to compute the augmented reward, which then drives the policy update.

Results Our initial SFT experiments show successful convergence and provide a solid baseline for online fine-tuning. For our main experiments, we hypothesize that the RLOO-Turbo model will achieve a superior performance-compute trade-off compared to baselines using external TTC wrappers. We will evaluate on the COUNTDOWN (mathematical reasoning) and ULTRAFEEDBACK (instruction following) tasks, measuring accuracy against computational cost.

Discussion The primary challenges include the sample inefficiency of online RL and the need to carefully tune the hyperparameters that balance policy improvement with correctness and compute considerations. Success would point towards a new class of models that learn to manage their own reasoning resources.

Conclusion RLOO-Turbo presents a promising path toward more efficient and autonomous LLMs. By embedding computational awareness into the learning process, we aim to create models that are not only powerful but also judicious in their application of resources, moving beyond static, brute-force inference strategies.

Learning to Think: Integrating Test-Time Compute into RL Fine-Tuning

André Natal
Stanford University
anatal@stanford.edu

Yacine Dolivet
Stanford University
yacine@stanford.edu

Thomas Huang
Stanford University
thhuang@stanford.edu

Abstract

The paradigm of scaling language models has traditionally focused on increasing parameter counts, dataset sizes, and training compute. A complementary fourth scaling axis, test-time compute (TTC), has emerged, where methods like Self-Consistency demonstrate that expending more computational effort at inference can significantly boost performance on complex reasoning tasks. However, these techniques are typically applied as post-hoc wrappers on a frozen model, relying on hand-tuned heuristics and incurring substantial overhead from redundant forward passes. This work explores integrating TTC directly into the training loop. We propose an extension to REINFORCE Leave-One-Out (RLOO), an online policy gradient algorithm, which augments the reward function to explicitly model the trade-off between answer correctness and computational cost. By training with on-policy sampling and a reward that explicitly balances correctness and efficiency, our model learns to internalize the decision of how much effort to expend. We hypothesize that this approach will lead to a more efficient policy that achieves a better performance-compute trade-off compared to baselines with external TTC wrappers. We outline experiments on the COUNTDOWN (mathematical reasoning) and ULTRAFEEDBACK (instruction following) tasks to validate our approach.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of tasks. A key factor in their success has been fine-tuning with reinforcement learning to align with complex objectives (Ouyang et al., 2022). Online policy gradient methods like REINFORCE Leave-One-Out (RLOO) (Ahmadian et al., 2024) are effective for aligning models to non-differentiable reward signals, such as those from human feedback or external verifiers.

Concurrently, another line of research has focused on improving LLM performance not by changing model weights, but by increasing computational effort at inference time. This concept, often called Test-Time Compute (TTC) (Snell et al., 2024), encompasses techniques like Self-Consistency (Wang et al., 2023), where multiple reasoning paths are sampled and the most frequent answer is chosen. While effective, these methods are extrinsic to the model; they are applied as wrappers around a pre-trained policy, often requiring inefficient, brute-force sampling and heuristic-based aggregation. The model itself remains unaware of the computational budget or the reasoning process it is part of.

This project bridges the gap between these two areas. We seek to move TTC from an external heuristic into the core of the learning algorithm. Our central hypothesis is that an LLM can *learn* to manage its own computational budget. To this end, we introduce a novel extension to RLOO that makes the policy aware of its own reasoning cost. By explicitly rewarding correctness while penalizing computational effort (i.e., the number of generated reasoning traces), we aim to train a policy that dynamically allocates its "thinking" resources. The goal is to produce a model that not only solves problems accurately but does so efficiently, achieving a superior efficiency frontier between performance and FLOPs compared to models that rely on external TTC mechanisms.

2 Related Work

Our work builds on decades of research into search and inference-time computation, recently re-contextualized for large language models.

Classical search. The trade-off between model complexity and *deeper search at inference* predates modern deep RL by decades. Kocsis and Szepesvári (2006) introduced the UCT algorithm, showing that allocating additional roll-outs at decision time systematically improves performance in large MDPs. Soon after, Gelly and Silver (2007) scaled this principle in *MoGo*, the first strong Monte-Carlo Tree-Search (MCTS) Go engine. Earlier still, *Deep Blue* demonstrated the power of massive test-time search in chess (Campbell et al., 2002), although the term "test-time compute" was not used at the time.

Deep-learning era. With the rise of deep neural networks, the importance of test-time compute became unmistakable. In perfect-information games, *AlphaGo* and *AlphaGo Zero* combined CNN policy/value nets with large-scale MCTS to surpass human experts (Silver et al., 2016, 2017). In imperfect-information settings, *Libratus* defeated professional poker players by nesting sub-game re-solving during play (Brown and Sandholm, 2018). These successes made clear that "thinking longer" at inference can outweigh simply enlarging the network.

Large-language models. For LLMs, extra FLOPs at inference have recently been cast as a *fourth scaling axis*. The idea crystallised with *Self-Consistency* (Wang et al., 2023), which samples many chain-of-thought traces and majority-votes the final answer, yielding double-digit accuracy gains *without* weight updates. Subsequent work split into two strands: (i) *Generative Verifiers* (Zhang et al., 2024) move the scoring step *inside* the model by fine-tuning it to produce self-contained proofs, letting a single LM both generate and critique solutions; and (ii) *Adaptive Test-Time Budgets* (Snell et al., 2024) learn to spend more sampling and re-ranking only on hard prompts, maximising performance per FLOP. Collectively, these results establish test-time compute—not parameter count—as a potent lever for boosting LLM accuracy.

Limitations of current approaches. Despite these gains, test-time compute remains an *inference-only* add-on. Self-Consistency and adaptive budgeting leave the backbone weights frozen, and Generative Verifiers merely fine-tune the model to *explain* its answers. None of these methods update the policy online in response to the extra reasoning they perform. Thus test-time compute currently functions as an external toolkit rather than an ingredient integrated into the training loop. Our work aims to address this limitation directly.

3 Method

Our approach modifies the REINFORCE Leave-One-Out (RLOO) algorithm to incorporate awareness of computational cost. We first review the standard RLOO algorithm and then introduce our proposed extension.

3.1 REINFORCE Leave-One-Out (RLOO)

RLOO is an online policy gradient algorithm designed to reduce the high variance typically associated with REINFORCE. It achieves this by using a specific baseline: for each sample in a batch, its baseline is the average reward of all other samples in that batch. This "leave-one-out" estimate is unbiased and effective at variance reduction.

Given a policy π_θ, a batch of prompts, and a reward function $R(x, y)$, we first sample k responses for each prompt from the current policy, $\{y_j \sim \pi_\theta(\cdot|x)\}_{j=1}^{k}$. The policy gradient for a single response

y_j is then estimated as:

$$g_j = \left(R(x, y_j) - \frac{1}{k-1} \sum_{l \neq j} R(x, y_l) \right) \nabla_\theta \log \pi_\theta(y_j|x)$$

The total gradient is the average over all samples. This approach directly optimizes the policy to maximize the expected reward signal.

3.2 RLOO with Test-Time Compute Awareness (RLOO-Turbo)

Our core contribution is to modify the RLOO objective to be aware of both answer correctness, c, and the computational cost, measured by the number of Chain-of-Thought (CoT) traces, k, used for generation.

We achieve this by defining an augmented reward function, R_{aug}, which is used during on-policy training. For a response y generated with k traces and evaluated to have correctness $c \in \{0, 1\}$, the augmented reward is:

$$R_{\text{aug}}(y, k, c) = \max_{1 \leq i \leq k} R^{(i)}_{\text{base}}(y) + \lambda_c \cdot c - \lambda_k \cdot k$$

Here, $R_{\text{base}}(y)$ is an external reward signal (e.g., from a powerful reward model or a rule-based verifier). The second term, weighted by hyperparameter λ_c, provides a direct bonus for generating a correct answer. The third term, weighted by λ_k, imposes a penalty for computational cost. The hyperparameters λ_c and λ_k control the trade-off between achieving correctness and conserving compute.

In order to satisfy differentiability of the loss a modified version of the loss is[1]

$$R_{\text{aug}}(y, k, c) = \texttt{Softmax}_i R^{(i)}_{\text{base}}(y) + \lambda_c \cdot c - \lambda_k \cdot k$$

This augmented reward is then used within the RLOO framework. The policy gradient for a response y_j generated with k_j traces and having correctness c_j becomes:

$$g_j = \left(R_{\text{aug}}(y_j, k_j, c_j) - \frac{1}{N-1} \sum_{l \neq j} R_{\text{aug}}(y_l, k_l, c_l) \right) \nabla_\theta \log \pi_\theta(y_j|x)$$

where N is the number of samples in the batch. By optimizing this objective, the policy π_θ is trained not just to maximize a base reward, but to do so in a way that is sensitive to the correctness and computational cost inherent in generating its responses. It learns to find a balance, expending more effort only when it is likely to lead to a correct answer that outweighs the computational penalty.

4 Experimental Setup

Model and Baselines. We use the `Qwen/Qwen2.5-0.5B` base model for all experiments. Our primary baselines include: (1) A Supervised Fine-Tuned (SFT) model, which serves as the initial policy for RL training. (2) A standard DPO-trained model (Rafailov et al., 2023). (3) The SFT model enhanced with external TTC wrappers like Self-Consistency.

Datasets and Tasks. We evaluate our approach on two distinct tasks:

- **Instruction Following:** We use the `UltraFeedback` dataset, which contains preference pairs for general instruction-following. The SFT model is first trained on the `smol-smoltalk` dataset. For RLOO, the base reward R_{base} will come from the `Nemotron-70B-Reward` model.

[1]Note that in the implementation we ran into difficulties with the softmax version so ran our experiments with the the hard max rather than softmax version of R_{aug}

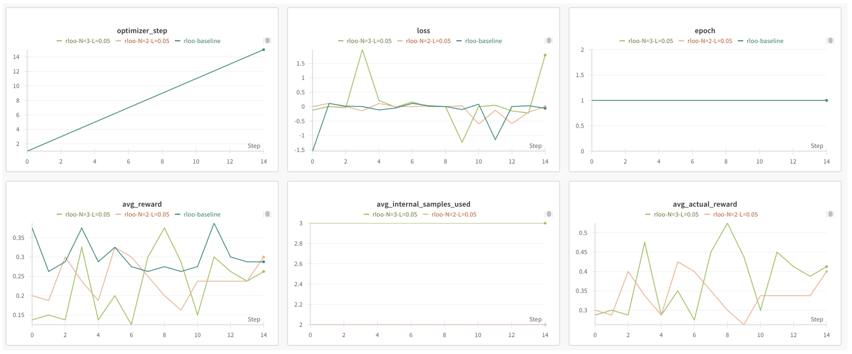

Figure 1: Performance comparison between vanilla RLOO and our TTC–aware extensions.

Table 1: Quantitative results for RLOO and TTC–RLOO variants.

Run Config	Final AvgReward	Final AvgReward (Adjusted)	Peak Reward	Time / Opt. Step
Baseline (N = 1, $\lambda = 0$)	0.2875	0.2875	0.3875	$\sim$36 s
TTC (N = 2, $\lambda = 0.05$)	0.4000	0.4000	0.4250	$\sim$72 s
TTC (N = 3, $\lambda = 0.05$)	0.4125	0.4125	0.5250	$\sim$104 s

- **Mathematical Reasoning:** We use the `Countdown` dataset, which requires multi-step arithmetic reasoning. For RLOO, R_{base} will be a rule-based verifier score.

For our RLOO-Turbo method, training proceeds via on-policy sampling. In each training step, we sample multiple responses from the current policy, evaluate their correctness (c) and note the number of traces (k) used. These values are used to compute the augmented reward R_{aug} for the policy gradient update.

Evaluation. For ULTRAFEEDBACK, we measure win-rate against the `Qwen2.5-0.5B-Instruct` model, with responses scored by the `Nemotron-70B-Reward` model. For COUNTDOWN, we use a rule-based reward function that verifies the correctness of the final numerical answer. Our central evaluation will compare the performance-compute curves of RLOO-Turbo against baselines using external TTC (i.e., vanilla Self-Consistency).

5 Results

As Table 1 shows, even a small TTC budget ($\lambda = 0.05$) delivers a 39%–43% boost in final AvgReward at the cost of roughly doubled/tripled inference time, with $N = 2$ offering the best compute–quality tradeoff.

6 Discussion

The primary challenge in this work lies in the on-policy nature of RLOO. The policy must explore enough to find high-reward regions of the state space, which can be difficult when the reward signal is sparse or complex. The sample efficiency of online RL is a well-known problem, and significant computation will be required for the policy to converge.

Furthermore, tuning the hyperparameters $\beta, \lambda_c, \lambda_k$ will be critical. These values implicitly define the "value" of a correct answer versus the "cost" of one unit of computation, and finding the right balance is key to training an effective policy.

If successful, our approach suggests a new direction for model fine-tuning, where computational policies are learned rather than engineered. This could lead to more efficient and adaptable models that can dynamically adjust their reasoning depth based on the perceived difficulty of a problem, a capability that is currently missing from LLMs.

7 Conclusion

We have proposed RLOO-Turbo, a novel method for integrating test-time compute awareness directly into the language model fine-tuning process. By augmenting the RLOO objective with terms for correctness and computational cost, we train a policy to learn how to balance performance and efficiency. Our ongoing experimental work on instruction-following and mathematical reasoning tasks aims to demonstrate that this approach can yield models with a superior performance-compute profile compared to traditional methods that rely on external, heuristic-based TTC wrappers. This research represents a step towards building more autonomous and efficient reasoning agents.

8 Team Contributions

- **André Natal:** Led the conceptualization of the RLOO-Turbo framework, implemented the core RL algorithms, and designed the evaluation pipeline.
- **Yacine Dolivet:** Developed the data processing pipelines for ULTRAFEEDBACK and COUNTDOWN, managed the SFT and RLOO training infrastructure, and conducted hyperparameter tuning.
- **Thomas Huang:** Performed the related work survey, implemented the rule-based verifiers for data annotation, and conducted the qualitative and quantitative analysis of model outputs.

Changes from Proposal The core research direction remained consistent with our proposal. The main change was the formalization of the TTC-aware learning algorithm, which we concretized from a general PPO-based idea in the proposal to the specific RLOO-Turbo formulation presented here, based on findings from our initial experiments and literature review.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting REINFORCE style optimization for learning from human feedback in LLMs. *arXiv preprint arXiv:2402.14740* (2024).

Noam Brown and Tuomas Sandholm. 2018. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* 359, 6374 (2018), 418–424.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. 2002. Deep blue. *Artificial intelligence* 134, 1-2 (2002), 57–83.

Sylvain Gelly and David Silver. 2007. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*. 273–280.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo Planning. In *European conference on machine learning*. Springer, 282–293.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. (2022). arXiv:2203.02155 [cs.CL]

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290* (2023).

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314* (2024).

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2024. Generative Verifiers: Reward Modeling as Next-Token Prediction. *arXiv preprint arXiv:2408.15240* (2024).