Test-Time Training for Efficient RL Sequence Modeling

Alexander Waitz
Department of Computer Science
Stanford University
waitz@cs.stanford.edu

Vrushank Gunjur
Department of Computer Science
Stanford University
vrushank@cs.stanford.edu

Kenny Dao
Department of Computer Science
Stanford University
kdao@cs.stanford.edu

Abstract

Modern Reinforcement Learning (RL) in partially observable settings requires agents to perform sequence modeling over a history of states. While Transformers are highly effective, their quadratic $O(n^2)$ computational complexity is often prohibitive for resource-constrained applications like robotics. In this paper, we investigate Test-Time Training (TTT), an efficient $O(n)$ RNN-style architecture that adapts its weights online via backpropagation at inference time. Our goals are twofold: (1) apply TTT to a sequential RL task, and (2) introduce a "gradient stopping" method to dynamically trade inference speed for accuracy. To overcome the inherent training instability of TTT , we develop a robust imitation learning pipeline that uses a privileged expert to train the TTT agent with Behavioral Cloning (BC) and Dataset Aggregation (DAgger). We evaluate our method on position-only variants of `CartPole-v1` and `AcroBot-v1`, which requires temporal reasoning to solve. On `AcroBot-v1`, TTT achieves an average reward of -65.93, which is comparable to the Transformer's score of -64.48. On `CartPole-v1`, our stabilized TTT agent achieves an average episode length of 266.64, underperforming the Transformer's baseline score of 434.9. Moreover, our gradient stopping heuristic proved to not be effective, as performance degraded faster than inference speed improved. We conclude that TTT, when paired with a robust imitation learning strategy, is a powerful and computationally efficient alternative to Transformers for sequential RL, though further experimentation with criteria for when to pause gradient updates is required. These findings suggest that TTT may be a viable alternative to Transformers in real-time RL applications, particularly for deployment on low-resource systems such as drones or embedded devices where accuracy can sometimes be traded for inference speed.

1 Introduction

Many real-world Reinforcement Learning (RL) tasks, particularly in fields like robotics, take place in partially observable environments. In these settings, the agent cannot rely on the current observation alone and must reason over a history of past interactions to infer the true state of the world. This necessity introduces the challenge of sequence modeling into the RL paradigm.

Transformers (Parisotto et al., 2019) have emerged as the dominant architecture for sequence modeling tasks, delivering state-of-the-art performance by using an attention mechanism to capture long-range

dependencies. However, the self-attention mechanism's computational and memory complexity scales quadratically with the sequence length ($O(n^2)$), posing a significant bottleneck for applications requiring low latency and efficient use of resources. More efficient architectures like Recurrent Neural Networks (RNNs) (Bakker, 2001) and State Space Models (SSMs) (Gu et al., 2022) offer $O(n)$ or near-linear complexity but often struggle to match the performance of Transformers, especially on tasks with long-term dependencies (Arora et al., 2023).

In this paper, we investigate Test-Time Training (TTT) (Sun et al., 2025), a novel RNN-style architecture that seeks to bridge this gap. TTT models adapt online by performing gradient updates on their own weights (which serve as the recurrent hidden state) at each step during inference. This allows the model to continuously learn from incoming data streams, making it robust to distributional shifts. Our work explores the application of TTT to sequential RL problems. We are motivated by the potential of TTT to match the performance of Transformers while retaining the $O(n)$ efficiency of recurrent models. We also propose an interpolation between the fixed, offline adaptation of models like Model-Agnostic Meta-Learning (MAML) (Finn et al., 2017) and the continuous online adaptation of TTT. We hypothesize that by selectively skipping gradient updates during periods of observational stability, we can further improve TTT's efficiency with minimal performance loss.

Our contributions are twofold: (1) We present a robust training pipeline using imitation learning that overcomes some of the inherent instabilities of training TTT-based policies but nevertheless underperforms transformers. (2) We introduce and evaluate a "gradient stopping" method for dynamically managing the computational cost of test-time updates, analyzing the resulting trade-off between inference speed and policy accuracy.

2 Related Work

2.1 Online Learning

Our work is closely related to the field of online or streaming learning in reinforcement learning, where an agent must learn continuously from incoming data without relying on large replay buffers or batch updates. This paradigm is particularly challenging due to the non-stationarity of the data stream, which can lead to catastrophic forgetting or training instability. A significant challenge in this area is the "stream barrier," a phenomenon where deep RL algorithms that are stable in batch settings often fail when adapted to a streaming context (Elsayed et al., 2024).

The work by Elsayed et al. (2024) introduces techniques to overcome this barrier, enabling deep RL agents to learn stably and efficiently from a single data stream. Their approach shares a similar motivation with our work on TTT: enabling robust, moment-by-moment adaptation in resource-constrained settings. While their focus is on eliminating replay buffers for standard RL algorithms, the problem of maintaining stability during continuous updates is directly analogous to the challenge of stabilizing the test-time gradient steps in TTT. Our use of imitation learning with DAgger can be seen as one strategy to provide a stable, supervised signal to guide the online adaptation process, addressing a similar instability problem from a different perspective.

2.2 Training at Test-Time

Training at Test-Time (TTT), as presented by Sun et al. (2025), is formulated as a Recurrent Neural Network (RNN) where the hidden state is a weight tensor which parametrizes an "inner" model. The update rule is defined by standard gradient backpropogation. There is also an "outer" model which defines the labels for which the inner model's backpropagation is defined by. In sum, one can think of the outer model as defining a supervised, online learning task for the inner model. This scheme is illustrated in Figure 1.

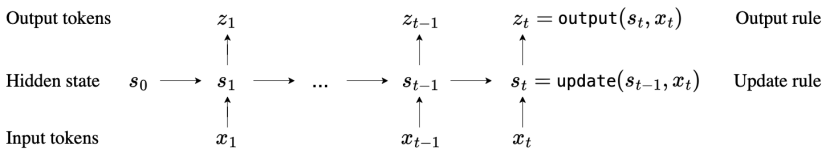


Figure 1: High-level depiction of the TTT pipeline.

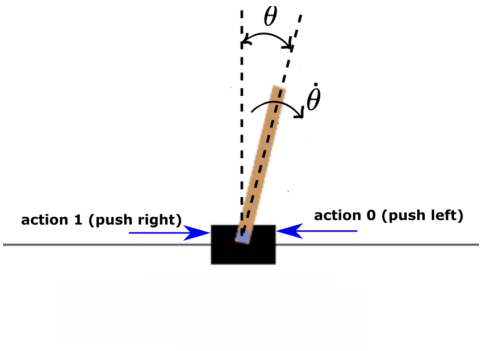
2.3 Model-Agnostic Meta-Learning

Model-Agnostic Meta-Learning (MAML), as presented by Finn et al. (2017), is a meta-learning methodology for pre-training a single RL policy which can be cheaply adapated to a variety of downstream tasks. In the MAML training procedure, we pretrain our model on a pre-defined set of tasks simultaneously using a meta-learning scheme. Then, during inference time, we can adapt our model for one of the specific tasks using a small number of fine-tuning steps on labeled data from that task. This enables us to maintain a single pretrained model which can be cheaply deployed to a variety of pre-defined tasks.

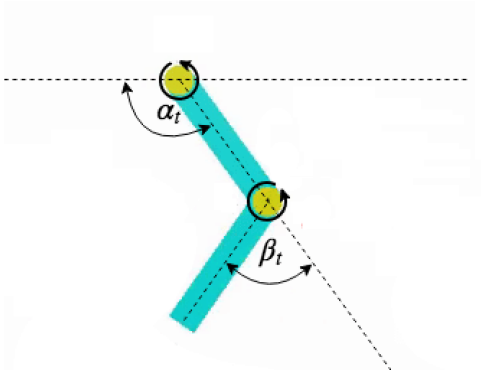
To clarify distinctions: MAML focuses on fast adaptation across tasks via meta-learning but lacks online updates; TTT adapts at every step during inference; and streaming RL emphasizes continual learning from non-stationary data without replay buffers. Our method intersects all three by stabilizing TTT through imitation learning in a streaming context.

3 Experimental Setup & Method

3.1 Environment



(a) `CartPole-v1` environment: The agent applies discrete left or right forces to balance an upright pole on a moving cart.



(b) `AcroBot-v1` environment: The agent applies torque at the joint between two connected links to swing the outer link above a target height.

Figure 2: Environments used for training and evaluation.

We train and evaluate our policies on a vanilla and masked variant of `CartPole-v1` (Barto et al., 1983), which is shown in Figure 2a. The action space for `CartPole-v1` is $\{0, 1\}$, where 0 corresponds to pushing the cart left and 1 corresponds to pushing the cart right. Moreover, the observation space is $\{o = (x, v, , \omega) | o \in \mathbb{R}^4, x \in [-4.8, 4.8], z \in [-0.418, 0.418]\}$. The first and third features correspond to position and angle, whereas the second and fourth features correspond to velocity and angular velocity.

In our masked variant, we modify the observation space to be $\{o = (x, z) | o \in \mathbb{R}^2, x \in [-4.8, 4.8], z \in [-0.418, 0.418]\}$. In other words, only position and angle are visible and the velocity features have been masked to simulate the environment known as "Position-only CartPole". Masking out the velocity information requires the model to create a memory of the previous states to

reconstruct the velocity information, effectively introducing a temporal dependency and converting the problem to a Partially Observable Markov Decision Process (POMDP).

An episode terminates if the agent is no longer remaining upright or is out of bounds. These bounds are defined as the limits on x and z in the definition of the observation space. The reward function is relatively simple, defined as $R(s) = 1$. Thus, the episode length is equal to the episodic reward. The max possible episodic reward is 500.

To ensure that our methods generalized, we also tested the implementations on the `AcroBot-v1` environment (Sutton, 1995), which is shown in Figure 2b. The `AcroBot-v1` environment is comprised of a chain with two links (the angle of each joint being θ_1 and θ_2, which are α and β in the diagram). The action space here is the torque applied on the actuated joint between two links, and the goal is to swing the chain above a given height after starting hanging downwards. For the fully observable variant, we give the models access to the rotational joint angles as well as the angular velocities. The masked version only has the angles $\cos$ and $\sin$ of θ_1 and θ_2, and the angular velocities are masked out. Reward is defined as -1 for each timestep the goal is not reached (incentivizing high swing height as early as possible) and the target heigh results in termination with a reward of 0. The episode ends if the episode length is greater than 500, or if $\cos(\theta_1) - \cos(\theta_2 + \theta_1) > 1$. The episodic reward is upper bounded by 0 (this can not be achieved in practice, since it would require completion in 0 steps).

3.2 Algorithms

In this paper, we present two novel developments: a reliable training methodology for TTT-parameterized RL policies and a method for reducing the computational footprint of TTT by interpolating between TTT and MAML. To apply both TTT and Transformers to an RL domain, we trained them with an action head to project the final layer into the action space for the environment. When training with PPO, we also introduced a critic head for value estimation. With the addition of these projections and the development of methods used to produce trajectories and initialize the models, we developed both a TransformerActor and TTTActor. The TransformerActor was trained with Proximal Policy Optimization (PPO) Schulman et al. (2017). We also used PPO to train a simple MLPActor as a baseline. Both the MLP and Transformer baseline training implementations build upon the code by Huang et al. (2022).

3.2.1 Training the TTT Actor

Our first hurdle was to devise a reliable method for training TTT-parameterized RL policies. RNNs notoriously exhibit unstable training dynamics due to gradients being backpropagated through the entire recurrent network call stack. Moreover, due to the meta-learning mechanism in TTT, these instabilities are compounded. We observe this in practice: while PPO (Schulman et al., 2017) is sufficient for training MLP and Transformer policies, TTT struggles to learn (depicted in Figure 3).

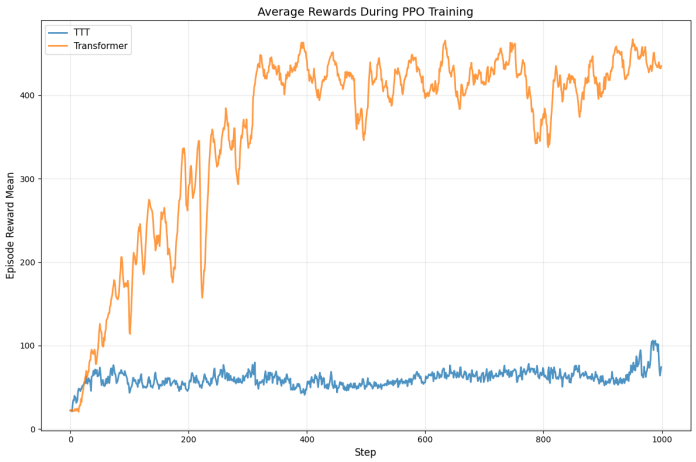


Figure 3: Comparison of training dynamics between a policy parameterized by a Transformer versus TTT using PPO on masked `CartPole-v1`, showing that PPO alone is ineffective to train the TTTActor.

In order to improve TTT performance, we applied a method called "Learning by Cheating" by Chen et al. (2019), where a privileged model has access to information about the environment which the target model lacks. The "Learning by Cheating" approach assumes that we can train a near-optimal expert policy by giving it access to the fully-observable state, and that its expertise can be effectively transferred to the TTT agent, which only experiences partial observability.

We first apply this insight by training an MLPActor for each of the fully-observable versions of the environments (which an MLP is sufficient to solve) and use these "experts" to train TTTActors on the partially observable versions of each environment. We initially perform behavior cloning followed by PPO, but discovered highly unstable training dynamics. We then incorporated a mix of both behavioral cloning and dataset aggregation (DAgger) (Ross et al., 2011) to learn from the privileged expert MLPActors, which was sufficient to produce stable training dynamics. We build a custom implementation for this training approach, as shown in Figure 4. This was the final architecture we used to train TTT on both environments, and the one which produced the best results.

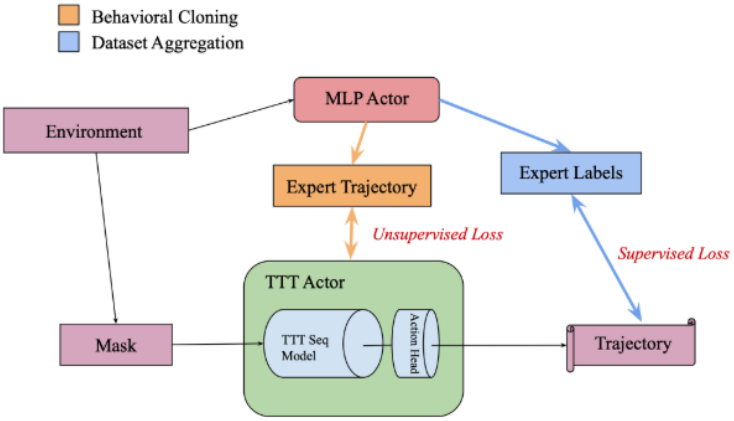


Figure 4: Approach used to train the TTT Actor.

Note: We also use this pipeline to train the Transformer in `Acrobot-v1`, since PPO fails to learn in this environment. Conversely, our BC + DAgger pipeline fails to train the Transformer in `Cartpole-v1`, thus we use PPO in this case.

3.2.2 Gradient Stopping Heuristic for TTT Efficiency

By viewing TTT as an adaptation of MAML, and to take advantage of an interpolation between these two methods, we aimed to improve the usability of TTT for RL by providing a mechanism through which to make a tradeoff between model accuracy and inference speed/computational burden.

We do so by selectively performing gradient updates for TTT during inference time by noticing that all backpropogations result in a similar number of FLOPs at this step, but they have varying degrees of usefulness based on the state of the environment and the actor.

To take advantage of this, we must find a signal to use in order to determine the right steps to stop backpropogation. We investigate two main metrics: the effective learning rate (η) as well as the norm of the gradient with regard to the output token. To determine the cutoffs, we first collected data on their distributions, as presented in Figure 5.

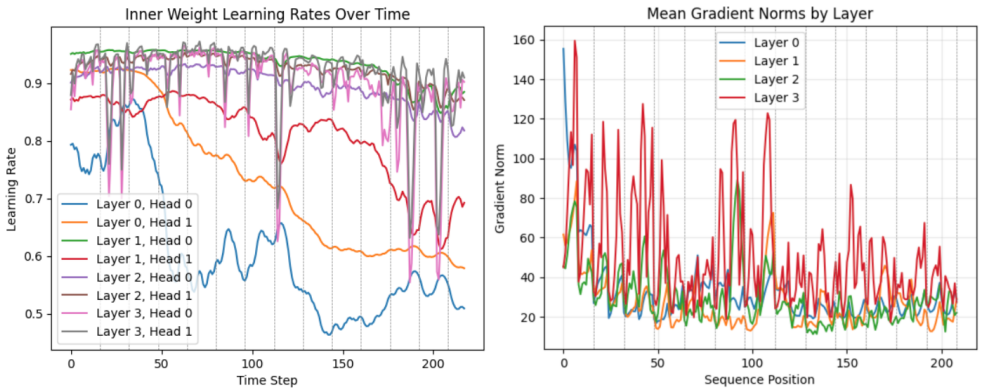


Figure 5: Prevalence of various learning rates and gradient norms during inference.

Note that there's an η value for each head and each layer of the network. We noticed that learning rates and gradient norms tended to rise in later layers, but it seemed like early layers' learning rates and gradient norms tended to stabilize and drop after a certain number of environment steps. We suspect that early layers are learning low-frequency policy features while later layers are learning high-frequency features.

Based on the relative predictability and clustering of the gradient norms in particular, we decided to base the cutoff on the size of the gradient norm. Namely, if the gradient norm is below a threshold, we choose not to do backpropagation. We refer to this method as "gradient stopping." We experiment with many different thresholds – the more gradient updates we stop, the worse the expected performance but the lower the expected FLOPs.

Since backpropoagation is performed over batched matrices across all layers and heads, selectively stopping updates for individual parts would lead to significant overhead. Thus, we compute the gradient stopping threshold by averaging across all layers and heads.

4 Quantitative & Qualitative Results

4.1 Fully Observable Environment

First, we perform some baseline experiments. To validate that our implementations were correct, we train an MLP, Transformer, and TTT policy on the fully observable `CartPole-v1` environment. All three architectures are able to learn optimal policies, as expected. We include a sample training log in Figure 6. Note that although all three policies were able to achieve max performance, TTT suffered from large training variance. After learning an optimal policy, the model suddenly collapses to near 0 performance before repeating another equally large fluctuation. This further motivates the TTT training pipeline with increased stability described in methods.

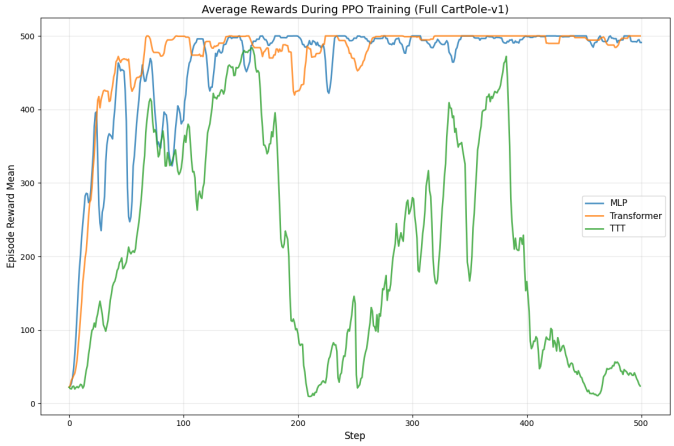


Figure 6: MLP, Transformer, and TTT obtaining optimal reward on fully observable `CartPole-v1` using PPO.

4.2 Partially Observable Environment

Architecture + Algorithm	**Avg. Ep Length (masked `CartPole-v1`)**
MLP w/ PPO	45.08
Transformer w/ PPO	434.90
TTT w/ PPO	43.78
TTT w/ PPO + BC	180.00
TTT w/ BC + DAgger	266.64

Figure 7: Average episode length of MLP, Transformer, and TTT agents on the partially observable `CartPole-v1` task. Higher values indicate better policy performance.

Next, we applied our TTTActor, MLPActor, and TransformerActor to the partially observable `CartPole-v1` environment to evaluate their sequence modeling capabilities. The performance of each architecture and training permutation is summarized in Figure 7.

As shown in the figure, the MLPActor was not able to learn more than an essentially random policy, achieving an average episode length of only 45.08. This was expected, as the MLP has no mechanism to capture the temporal dependencies required to solve this problem. Qualitatively, the MLPActor oscillates left-and-right since it can tell if it is falling one direction or the other by position only. However, since it can not calculate velocity, it overcorrects in the opposite direction and ends up oscillating wider until falling. We attempt to depict this in Figure 8 although it is difficult to fully perceive the trajectory without embedding a video. The TransformerActor, serving as our state-of-the-art baseline, successfully learned an effective policy and achieved a high average episode length of 434.90. This demonstrates that the task is solvable with a powerful sequence model.

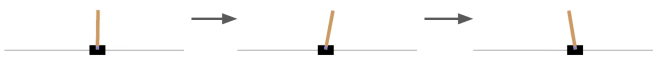


Figure 8: Failure trajectory under MLP. Begins in balanced state but progressively overcorrects, leading to loss of control and pole collapse.

The TTTActor's performance was highly dependent on the training methodology, highlighting the instability of recurrent meta-learning systems: (1) When trained with PPO alone, the TTTActor struggled with instability and performed on par with the MLP, reaching an average episode length of only 43.78. (2) Adding Behavioral Cloning (BC) pre-training before PPO fine-tuning offered

a significant improvement by providing a better initialization, raising the score to 180.00. (3) Our final proposed method, training the TTTActor with a combination of BC and DAgger from a privileged expert, yielded the best result at 266.64. This demonstrates the necessity of a stabilized, imitation-based learning pipeline for making TTT effective and competitive in a sequential RL setting.

Qualitatively, we noticed that the TTTActor was able to stay within the angular bounds of the environment quite successfully. However, it would often smoothly drift either left or right during the trajectory, eventually going out of bounds as depicted in Figure 9:

Figure 9: Failure trajectory of TTT. Begins in balanced state but continually corrects left, leading to loss of control and pole moving out-of-bounds.

We believe this illuminated the failure mode of TTT; while it could successfully temporally local behavior, it was unable to address drift across the entire sequence. This is a type of sequence modeling weakness that is characteristic of RNNs.

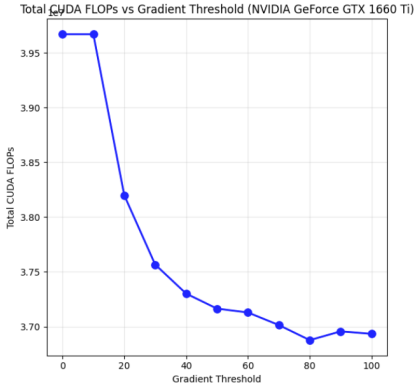
Architecture + Algorithm	**Avg. Ep Reward** (`AcroBot-v1`)
MLP w/ PPO	-496.42
Transformer w/ BC + DAgger	-64.48
TTT w/ BC + DAgger	-65.93

Figure 10: Average episode length of Transformer and TTT agents on the partially observable `AcroBot-v1` task. Higher values indicate better policy performance.

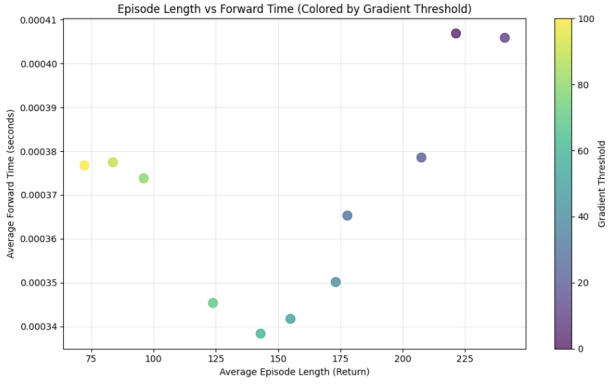
For generality, we also test our training pipeline on `Acrobot-v1`, which is depicted in 10. TTT and transformer achieve essentially equivalent episodic rewards on this slightly simpler task.

Ultimately, the success of the DAgger-based pipeline stems from its ability to provide a stable, supervised learning signal. Standard RL methods like PPO rely on noisy, often delayed reward signals for credit assignment. This challenging exploration problem is compounded by TTT's recurrent meta-learning dynamic, leading to training instability. In contrast, DAgger provides immediate, expert-labeled actions for every state the agent visits, converting the problem into a more stable supervised learning task that is better suited to TTT's online weight updates.

4.3 Gradient Stopping



(a) Effect of removing gradient steps based on various thresholds on GPU FLOPs.



(b) Effect of stopping gradients during test-time on inference speed.

Figure 11: Impact of gradient stopping on both computational cost and inference speed.

For our exploration of gradient stopping to improve speed / FLOPs at the cost of accuracy, we performed a sweep of the gradient norm of the loss with regard to the output token (using the masked `CartPole-v1` environment). This gradient is calculated prior to any backpropagation, so we avoid incurring the cost of calculating and propogating gradients for the whole inner network. As shown in Figure 11a, skipping backpropagation based on low gradient norms led to a clear reduction in GPU FLOPs, confirming that the heuristic can lower inference-time computational overhead. We discovered a clear correlation between increased gradient stopping and decreased episodic reward. Qualitatively, we also observed that increased gradient stopping was correlated with less smooth and more erratic policy behavior.

However, this gain in efficiency came at a cost. As shown in Figure 11b, once the threshold became too aggressive (e.g., stopping all gradients with norms below 70), the average forward time unexpectedly increased. We speculate that this might be happening due to two compounding effects: First, performance suffers significantly when most gradient updates are skipped, leading to shorter episode lengths, more frequent environment resets, and lower cache utilization/hit rates. While individual resets are inexpensive, their cumulative overhead may start to dominate. Second, the control flow behind the frequent gradient skipping (e.g. conditional execution, memory reuse, etc) may inhibit efficient batching and caching within the deep learning framework itself.

5 Discussion

Our results highlight two key findings: (1) Test-Time Training, despite its inherent training complexities, can be made into a effective sequence model for reinforcement learning. The failure of direct PPO training underscores the challenge of stabilizing recurrent meta-learning systems. However, the success of our imitation learning pipeline, which uses a privileged expert and DAgger, provides a clear and effective recipe for training these models. With our pipeline, we are able to train a TTT model that performs equally to a Transformer model on `AcroBot-v1`, while there remains a performance gap in `CartPole-v1`. Nevertheless, TTT remains theoretically far more efficient than the attention mechanism in Transformers ($O(n)$ vs. $O(n^2)$). We also believe that our training methods leave significant room for improvement. A major challenge we faced was deciding whether a model's under-performance was due to inherent limitations or poor hyperparameter choice. For instance, is there an inherent reason TTT cannot learn an effective policy with PPO, or did we simply not discover an appropriate hyperparameter configuration? Our success in training a TTT model on `AcroBot-v1` lends credence to our belief that TTT could also achieve competitive performance on `CartPole-v1` given the correct setup and more time performing hyperparameter sweeps. We hope that our results act as groundwork for future investigations into operationalizing TTT as a compelling architecture for resource-constrained, long-context RL applications. We believe it could push the frontier of sophistication for policies served on smaller, less powerful hardware, such as autonomous robots.

(2) Our exploration of dynamic gradient stopping reveals the subtleties of online adaptation. The simple heuristic of skipping updates based on gradient norm was not as effective as we hoped; it created an undesirable accuracy/efficiency trade-off. This suggests that the magnitude of the gradient alone is not a sufficient signal to determine the utility of a learning step. One point of interest is that removing only the very smallest gradients (i.e., using a very small τ) sometimes led to a minor improvement in performance. This may indicate that these small updates are effectively noise, and pruning them can stabilize the online learning process. However, we were unable to devise a reliable formula for producing this effect. This points to a potential avenue for improving the core TTT framework itself, perhaps through gradient regularization rather than a hard cutoff (which could also be implemented more cleanly and effectively in a kernel).

The inconsistent nature of the gradient norms, as seen in our analysis, likely makes it difficult for a single, global threshold to be effective. A more sophisticated mechanism, perhaps one that considers both the learning rate and the gradient norm, or even a learned model that decides when to update, may be required to find a more favorable point on the efficiency-performance curve. However, we still believe that the ability of TTT to provide this tradeoff is a key and underexplored capability of the model architecture which still shows significant promise.

Finally, we also note that, out of necessity, we train our baseline Transformer policy using our BC + DAgger pipeline in `Acrobot-v1` and with PPO in `Cartpole-v1`. This choice is motivated

by the failure of PPO to learn a successful policy in `Acrobot-v1` and failure of BC + DAgger in `Cartpole-v1`. We suspect that BC + DAgger performs better than PPO in `Acrobot-v1` since its reward environment is sparser and thus an expert is necessary to boost reward signal. We do not have a good explanation for why PPO works better for the Transformer in `Cartpole-v1`.

6 Conclusion

In this work, we successfully adapted the Test-Time Training (TTT) architecture for use in sequential reinforcement learning. We demonstrated that by using a robust imitation learning strategy based on DAgger, BC, and a privileged expert, we could overcome the training instabilities inherent to TTT and achieve performance nearly on par with a state-of-the-art Transformer baseline. This result validates TTT as a computationally efficient and powerful alternative for RL problems requiring temporal reasoning; though it requires some extra setup and modification to the environment. Our secondary contribution, a method for dynamically skipping test-time gradient updates, failed to provide a favorable tradeoff but validated the notion of providing an inference-time lever for resource usage. This highlights that while TTT is powerful, naively reducing its computational load can significantly harm its adaptive capabilities.

7 Future Work

Our findings open up several avenues for future research:

- **Bridging the Performance Gap:** Further work is needed to close the remaining performance gap between TTT and Transformers in all environments. This could involve exploring more advanced TTT architectures, more sophisticated self-supervised learning objectives, or more extensive hyperparameter sweeps.
- **Intelligent Gradient Stepping:** A key area for improvement is the gradient stopping mechanism. Future work could explore learning a parametric function to decide when to perform a gradient step, potentially taking into account state novelty, policy entropy, or other signals in addition to the gradient norm.
- **Gradient Regularization:** Instead of a hard cutoff, incorporating a regularization term into the training objective that penalizes frequent or large updates could encourage the model to learn to make more efficient use of its test-time updates inherently.

We are also broadly interested in better understanding how RL performance scales with compute resource allocation. In our results, we discover that, gradient stopping degrades performance at a faster rate than it saves compute by overall percentage. In future work, we would like to explore how this axis of compute-performance scaling compares to that of other architectures and algorithms. For instance, we wonder whether simply decreasing parameter count would yield a similar relationship between computational resources and policy performance. If not, is TTT's scaling under or over-performing alternative scaling axes? Moreover, we are interested in analyzing TTT's efficiency in an embedded hardware setting, specifically when being served on a robot or other autonomous piece of hardware.

8 Team Contributions

- **Alex Waitz:** Integrated TTT with episodic PPO and implemented DAgger from scratch. Setup and ran experiments. Tuned hyperparameters for improved learning. Set up data collection for TTT and analyzed internal gradient behavior.
- **Vrushank Gunjur:** Set up testing environment and baselines such as transformer on fully and partially observable environments. Did timing and performance benchmarking for gradient stopping. Built TransformerActor and MLPActor. Implemented TTT verbosity for analysis of gradient norms, learning rates etc.
- **Kenny Dao:** Set up environment, implemented BC, ran experiments, explored expansion into other environments, etc.

Changes from Proposal Our initial proposal centered on developing a reinforcement learning framework to actively guide the adaptation process of a TTT agent, where an "outer" RL agent would learn a policy for how and when to update the "inner" TTT model. However, we discovered that stabilizing the training dynamics of the TTT-parameterized policy within an RL context was a significant and foundational challenge. Consequently, our project's focus shifted. We pivoted from developing an RL meta-controller to first establishing a robust training methodology for the TTT agent itself. This led us to employ a "learning by cheating" paradigm with Dataset Aggregation (DAgger), a technique not outlined in the original proposal. Subsequently, we replaced the proposed learned adaptation policy with a simpler, heuristic-based method, "gradient stopping," to explore the efficiency trade-offs. In essence, the project evolved from investigating a learned adaptation policy to first making TTT viable in RL and then evaluating a simpler heuristic for its efficiency.

References

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. 2023. Zoology: Measuring and Improving Recall in Efficient Language Models. arXiv:2312.04927 [cs.CL] https://arxiv.org/abs/2312.04927

Bram Bakker. 2001. Reinforcement Learning with Long Short-Term Memory. In *Advances in Neural Information Processing Systems*, T. Dietterich, S. Becker, and Z. Ghahramani (Eds.), Vol. 14. MIT Press. https://proceedings.neurips.cc/paper_files/paper/2001/file/a38b16173474ba8b1a95bcbc30d3b8a5-Paper.pdf

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13, 5 (1983), 834–846. https://doi.org/10.1109/TSMC.1983.6313077

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. 2019. Learning by Cheating. arXiv:1912.12294 [cs.RO] https://arxiv.org/abs/1912.12294

Mohamed Elsayed, Gautham Vasan, and A. Rupam Mahmood. 2024. Streaming Deep Reinforcement Learning Finally Works. arXiv:2410.14606 [cs.LG] https://arxiv.org/abs/2410.14606

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. arXiv:1703.03400 [cs.LG] https://arxiv.org/abs/1703.03400

Albert Gu, Karan Goel, and Christopher Ré. 2022. Efficiently Modeling Long Sequences with Structured State Spaces. arXiv:2111.00396 [cs.LG] https://arxiv.org/abs/2111.00396

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. 2022. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research* 23, 274 (2022), 1–18. http://jmlr.org/papers/v23/21-1342.html

Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M. Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. 2019. Stabilizing Transformers for Reinforcement Learning. arXiv:1910.06764 [cs.LG] https://arxiv.org/abs/1910.06764

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. 2011. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. arXiv:1011.0686 [cs.LG] https://arxiv.org/abs/1011.0686

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] https://arxiv.org/abs/1707.06347

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. 2025. Learning to (Learn at Test Time): RNNs with Expressive Hidden States. arXiv:2407.04620 [cs.LG] https://arxiv.org/abs/2407.04620

Richard S Sutton. 1995. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems*, D. Touretzky, M.C. Mozer, and M. Hasselmo (Eds.), Vol. 8. MIT Press. https://proceedings.neurips.cc/paper_files/paper/1995/file/8f1d43620bc6bb580df6e80b0dc05c48-Paper.pdf

A Experiment Details

Our implementation can be found here: https://github.com/ajwaitz/ttt-rl