When Test-Time Training Fails: A Critical Analysis of Robustness and Hyperparameter Sensitivity

Anonymous authors
Paper under double-blind review

Abstract

Test-time training (TTT) through input perplexity minimization has emerged as a promising approach for enhancing language model performance during inference. However, questions remain about its practical robustness and applicability beyond popular benchmarks. This paper presents a preliminary analysis investigating two critical questions: whether TTT is effective on unseen tasks and how sensitive it is to hyperparameter choices. We evaluate TTT on three anti-memorization datasets—Memo-Trap, GSM-Symbolic, and Math-Perturb—using six models from the Qwen 2.5 and Llama 3 families. Our findings reveal that while TTT shows effectiveness on common benchmarks such as AIME 2024, it struggles with tasks designed to counter memorization, raising questions about whether the gains stem from domain adaptation or data contamination. We identify significant performance differences among optimizers, with SGD outperforming Adam despite slower convergence. Through extensive hyperparameter sweeps over learning rates, training steps, weight decay, momentum, and gradient normalization, we demonstrate that TTT is highly sensitive to these choices, with no universal recipe across tasks and models. Notably, gradient normalization emerges as an effective technique for improving robustness by mitigating catastrophic performance drops and reducing sensitivity to the learning rate. Our analysis also reveals that tuning feed-forward networks can achieve better peak performance than full model tuning, while attention-only tuning provides more stable worst-case performance. These findings highlight the need for continued research into making test-time training more practical and reliable for real-world deployment. Since this research only focuses on a specific algorithm of TTT: input perplexity minimization, our conclusion may not be applied to all TTT algorithms. We call on the community to pay closer attention to TTT's sensitivity to make it better suited for real-world applications.

1 Introduction

Advocated by OpenAI o1 (Jaech et al., 2024), test-time scaling has gained significant attention from researchers in language models (LMs) due to its effectiveness in enhancing the model's capabilities, ranging from everyday question-answering to specialized domain reasoning. Test-time scaling naturally covers a broad spectrum. For example, researchers can scale the quantity of generated tokens, such as triggering the model's self-reflection capabilities (Muennighoff et al., 2025). Alternatively, researchers may incorporate the training process into the testing phase (Zuo et al., 2025; Hu et al., 2025b;a). We refer to the latter approach as test-time training (TTT).

TTT encompasses a substantial body of research with diverse methodological approaches, spanning various algorithmic frameworks and modalities across multiple domains (Sun et al., 2020; Wang et al., 2025; Hardt & Sun, 2024; Sun et al., 2024; Dalal et al., 2025). However, they do share a similar workflow:

`Input` $x \rightarrow$ `Training model` θ `with TTT` $f(x) \rightarrow$ `Inference on target tasks` $g(x)$

$g(x)$ is the task we aim to test, which could be a classification problem or a generative problem. In the era of LMs, $g(x)$ is unified to a next-token prediction problem. The function $f(x)$, as the TTT task, is designed manually and must align with the function $g(x)$ (Sun et al., 2020). Therefore, the design of $f(x)$ could vary.

For example, $f(x)$ could be an image reconstruction task when $g(x)$ is image classification (Gandelsman et al., 2022); Zuo et al. (2025) use majority voting to build reward signals and apply reinforcement learning on top of it.

Language models unify tasks into next-token prediction given a sequence of tokens. To enable test-time training for LMs, we minimize the model's perplexity on the test input in a self-supervised manner, adapting it to the test distribution. This approach has been shown to be effective both theoretically (Hu et al., 2025a) and empirically (Hu et al., 2025b). Though exciting, two dark clouds hang over its practical applications.

First, the existing experiment focuses on the existing popular benchmarks. For example, Sun et al. (2020) tests on AIME 2024 (Mathematical Association, 2024), GSM8k (Cobbe et al., 2021), Math500 (Lightman et al., 2023) and GPQA (Rein et al., 2024); Hu et al. (2025a) test on GSM8k (Cobbe et al., 2021), MetaMATH (Yu et al., 2023), and Logiqa (Liu et al., 2020). Considering the data pollution of current LMs (Wu et al., 2025), it is questionable whether the gain comes from data contamination or distribution adaptation.

Unlike the conventional training workflow, TTT usually does not have a reliable validation set to choose appropriate hyperparameters. Existing works usually choose a fixed setting, which blurs the future of real deployment of TTT. For example, Hu et al. (2025a) uses Adam optimizer ((Kingma & Ba, 2014)) and different learning rates for different datasets; Hu et al. (2025b) uses Adam optimizer, a fixed learning rate, a fixed training step, and weight decay in the experiment. It is reasonable to consider ways to reduce the sensitivity of hyperparameters since we often do not have a validation set for us to choose the best hyperparameters for test tasks.

Therefore, this paper focuses on answering two questions for input perplexity minimization TTT:

- Is TTT effective for unseen tasks?
- Is TTT sensitive to hyperparameters? If so, how can we reduce its sensitivity?

For the first question, we select three representative tasks that specifically go against the model's memorization: Memo-Trap (McKenzie et al., 2024) that asks the model to generate an anti-common-sense quote; GSM-Symbolic (Mirzadeh et al., 2024) and Math-Perturb (Huang et al., 2025) that alter the problem of GSM8k (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021b). For the second question, we investigate different hyperparameters: learning rate, training steps, optimizers, weight decay, momentum, and gradient normalization. Results on six models from two model families (Qwen 2.5 (Qwen et al., 2025) and Llama 3 (Dubey et al., 2024)) lead to the following findings:

- **Effectiveness and Hyperparameters Sensitiveness**: Although TTT is effective on some popular benchmarks, we identify that TTT struggles to work when the dataset is made to anti-memorization, potentially suggesting that TTT is recalling memorization instead of doing domain adaptation. For tasks where TTT shows its effectiveness, different tasks require different best learning rates, and different training steps for best performance.
- **Optimizers**: Although Adam typically converges faster, the SGD optimizer is better suited for the loss landscape of this optimization problem. Specifically, SGD converges and overfits more slowly, and it has more stable and better performance than Adam. Even if we lower Adam's learning rate for slower convergence, its performance is still worse than SGD. The preliminary analysis suggests the reason is the more accurate gradient direction in SGD. Considering the large performance gap between the two optimizers, we choose to use SGD to analyze other factors.
- **Regularization**: The regularization is often achieved via weight decay; we experiment with a commonly used value 0.01 as well as a large value 0.5. Experiments show that different regularization factors do not obviously contribute to preventing overfitting.
- **Gradient Normalization**: The gradient normalization makes the model more robust to the choice of learning rate and training steps, significantly slows the pace of overfitting, and sometimes even improves the TTT performance.

- **Momentum**: Momentum serves a similar role to gradient normalization, but is less robust than gradient normalization, as sometimes it makes TTT performance even worse.
- **Different Parameters**: We specifically only tune part of the models: Feed-forward networks (FFNs) or attention modules. Attention modules have fewer parameters and show better robustness to performance drops; however, tuning FFNs can match or even exceed the best performance when compared with fully finetuning.

In brief, this paper presents a preliminary analysis of the instability issues associated with test-time training for language models and advocates for greater attention to make test-time training a more practical approach. We want to emphasize that our research focuses on a specific TTT algorithm: input perplexity minimization; therefore, our conclusion may not apply to all TTT algorithms. We call on the community to pay closer attention to make TTT more suitable for real-world applications.

2 Method and Settings

2.1 Test-time Training

We use $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=0}^{N}$ to denote training dataset, and $\mathcal{D}_{\text{test}} = \{(x_i)\}_{i=0}^{M}$ to denote the test dataset. A language model $P_\theta(\cdot|x)$ parameterized by θ, is trained by optimizing $\mathcal{L} = \sum_{(x,y)\sim\mathcal{D}_{\text{train}}} -\log(P_\theta(y|x))$, which is the typical pre-training and supervised fine-tuning process.

A test-time training approach first defines a new training objective that depends solely on the model and input, $\mathcal{L}_{\text{test}}(\theta, x)$. Then for each $x \sim \mathcal{D}_{\text{test}}$, a θ_x is obtained based on θ:

$$\theta_x \leftarrow \theta - \gamma \nabla_\theta \mathcal{L}_{\text{test}}(\theta, x)$$

The formula above uses gradient descent for one step, which can be replaced by other optimizers and applied over multiple steps. Then the prediction is conducted by $\hat{y} \sim P_{\theta_x}(\cdot|x)$. The θ_x is discarded when the prediction is finished.

$\mathcal{L}_{\text{test}}(\theta, x)$ has different designs in different literature, and we use perplexity minimization (Hu et al., 2025a;b), a representative method of TTT, due to the simplicity and effectiveness:

$$\theta_x \leftarrow \theta + \gamma \nabla_\theta \log(P_\theta(x))$$

2.2 Dataset

We aim to select the datasets that go against the model's memorization to better understand the functionality of TTT. Specifically, we choose:

Memo-Trap (McKenzie et al., 2024) aims to test language models' instruction following capabilities by asking the model to complete a famous quote with an uncommon word. For example, the input `Write a quote that ends in the word "heavy": Absence makes the heart grow` expects the output "heavy" instead of "fonder". The dataset contains 860 examples.

GSM-Symbolic (Mirzadeh et al., 2024) is a dataset that perturbs the GSM8k data such as changing nouns and numbers. It tests the model robustness to the problems and contains two subsets: the easier "p1" with 5,000 data points and the harder "p2" with 2,500 data points. We use the "p2" dataset in our experiment as it is more difficult.

Math-P (Huang et al., 2025) similarly perturbs the MATH data and consists of two subsets: a simple one with 279 datapoints and a hard one with 279 datapoints. We experiment on a "simple" subset, as another subset is too difficult for models to solve. Due to Llama's poor performance on this dataset (Huang et al., 2025), we only experimented with Qwen 2.5 series models on MATH-P.

Dataset	Model	Learning Rate	Training Steps 0	20	40	60	80	100
AIME 2024	Qwen 2.5 1.5B-Instruct	$1e-6$	3.33	3.33	3.33	**6.67**	**6.67**	**6.67**
		$1e-5$	3.33	**6.67**	**6.67**	**6.67**	**6.67**	3.33
		$5e-5$	3.33	3.33	0.00	3.33	3.33	**6.67**
	Qwen 2.5 3B-Instruct	$1e-6$	6.67	**10.00**	6.67	6.67	6.67	3.33
		$1e-5$	6.67	**13.33**	3.33	3.33	**6.67**	3.33
		$5e-5$	**6.67**	3.33	3.33	3.33	3.33	0.00
AIME 2025	Qwen 2.5 1.5B-Instruct	$1e-6$	3.33	3.33	3.33	3.33	3.33	3.33
		$1e-5$	3.33	3.33	0.00	0.00	0.00	0.00
		$5e-5$	3.33	0.00	0.00	0.00	0.00	0.00
	Qwen 2.5 3B-Instruct	$1e-6$	6.67	6.67	3.33	3.33	3.33	3.33
		$1e-5$	6.67	**10.00**	3.33	3.33	0.00	0.00
		$5e-5$	6.67	3.33	6.67	6.67	6.67	3.33

Table 1: TTT with SGD optimizer on AIME 2024 and AIME 2025. Qwen 2.5 works well in AIME 2024 but struggles to achieve better and stable performance on AIME 2025.

2.3 Model

We adopt Llama 3 (Dubey et al., 2024) and Qwen 2.5 (Qwen et al., 2025) series of models for the experiment. Specifically, we test with 1B/3B/8B sizes for Llama 3 and 1.5B/3B/7B sizes for Qwen 2.5. For the memo trap dataset, we use base models, and for the other two datasets, we use instruction-following models. For Llama 1B/3B models, we use the 3.2 version, and the 3.1 version when testing the 8B model.

2.4 Hyperparameters

As hyperparameter robustness is the core topic of this paper, we aim to sweep the hyperparameters as broadly as we can to find the best recipe to make the model robust. First, we sweep the learning rate and training steps in a wide range. The learning rate spans 5 orders of magnitude (from $1e-8$ to $1e-3$), and training steps span up to 100. Second, we only adopt common choices and values for optimizers, regularization, momentum, and gradient normalization to avoid our claims overfitting our tasks. For optimizers, we ablate SGD and Adam optimizers. For regularization, we use weight decays of 0.01 and 0.5. For momentum, we use a weight of 0.9. For gradient normalization, we use a value of 10 because the common value 1 converges extremely slowly.

3 Main Observation 1: TTT Is Not A Panacea

TTT works for AIME 2024, but struggles with 2025. The very first step is to replicate results in AIME to ensure TTT works in popular benchmarks. Table 1 shows the performance of Qwen 2.5 on AIME with TTT and SGD optimizer. Obviously, Qwen 2.5 achieves better and more stable performance with TTT than the original. This aligns with prior research observation (Hu et al., 2025b). However, models tend to struggle with AIME 2025 and can not get better results with TTT. It is noticeable that Qwen is released after AIME 2024 but before AIME 2025, which questions if the gain comes from data contamination.

TTT struggles with anti-memorization datasets. To further validate our hypothesis, we conduct experiments on three anti-memorization datasets with TTT and SGD optimizer. Figure 1 shows the accuracy and loss progression with different training steps and learning rates. In general, models struggle to get better performance with TTT. Specifically, Qwen 2.5 1B/7B does not achieve performance gains on the Memo-Trap dataset. Even when Qwen 2.5 models show certain performance gains, their results are unstable and sensitive to training steps. For example, Qwen 2.5 3B suffers from a quick performance drop on GSM-Symbolic when

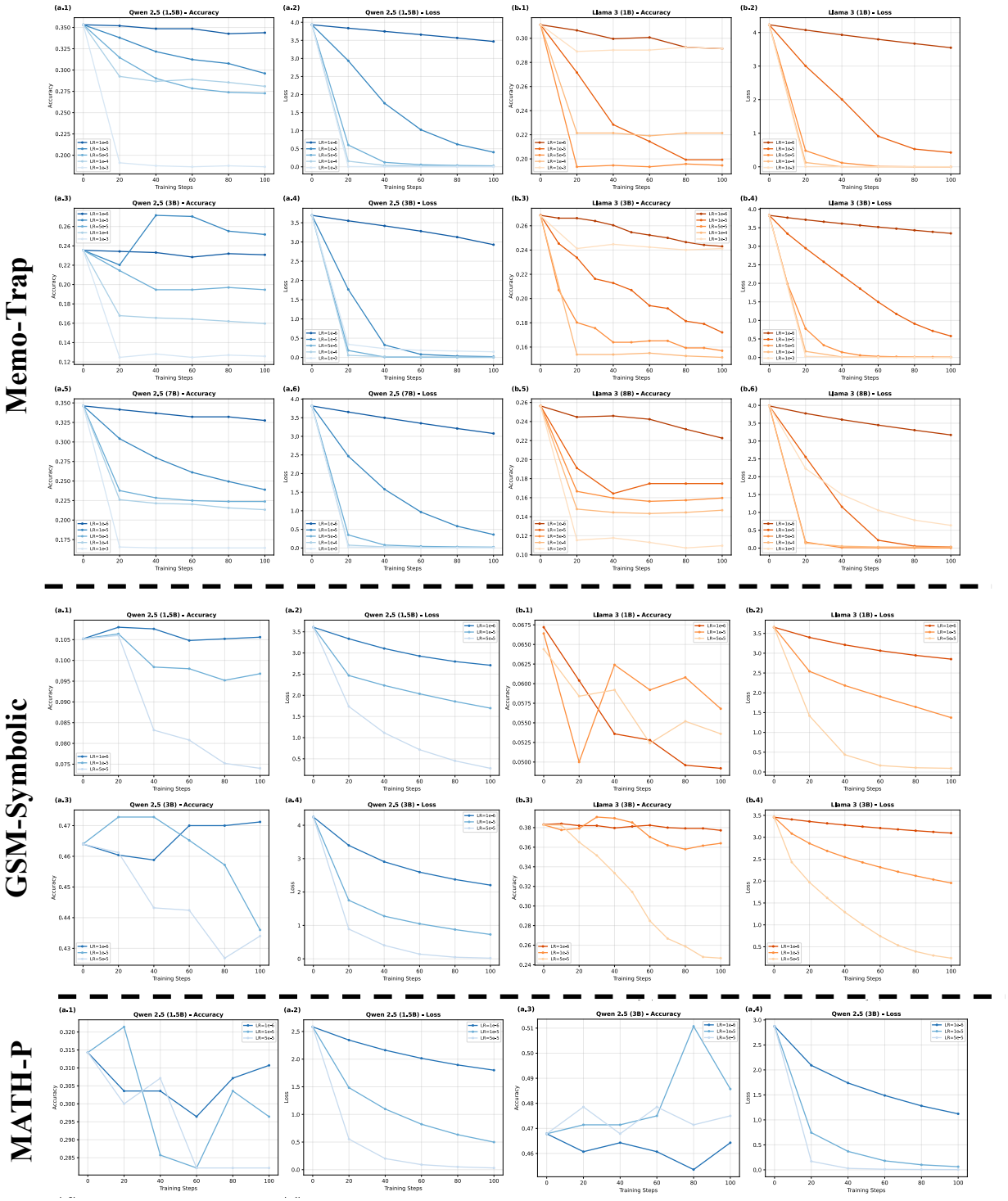

Figure 1: Comparison of Qwen 2.5 / Llama 3 models across different learning rates and training steps during TTT on three different datasets. x-axis represents training steps. **Color**: Blue indicates Qwen 2.5 model, and orange indicates Llama 3 models. Color depth indicates different learning rates. **Rows:** Different model sizes. **Columns:** The first and third column denotes accuracy, and the second and fourth column indicates losses. Compared with AIME results in Table 1, TTT on these datasets struggles to achieve better and more stable results. Besides, TTT learning dynamics differ with various models/sizes/tasks, learning rates, and training steps. There is no universal recipe.

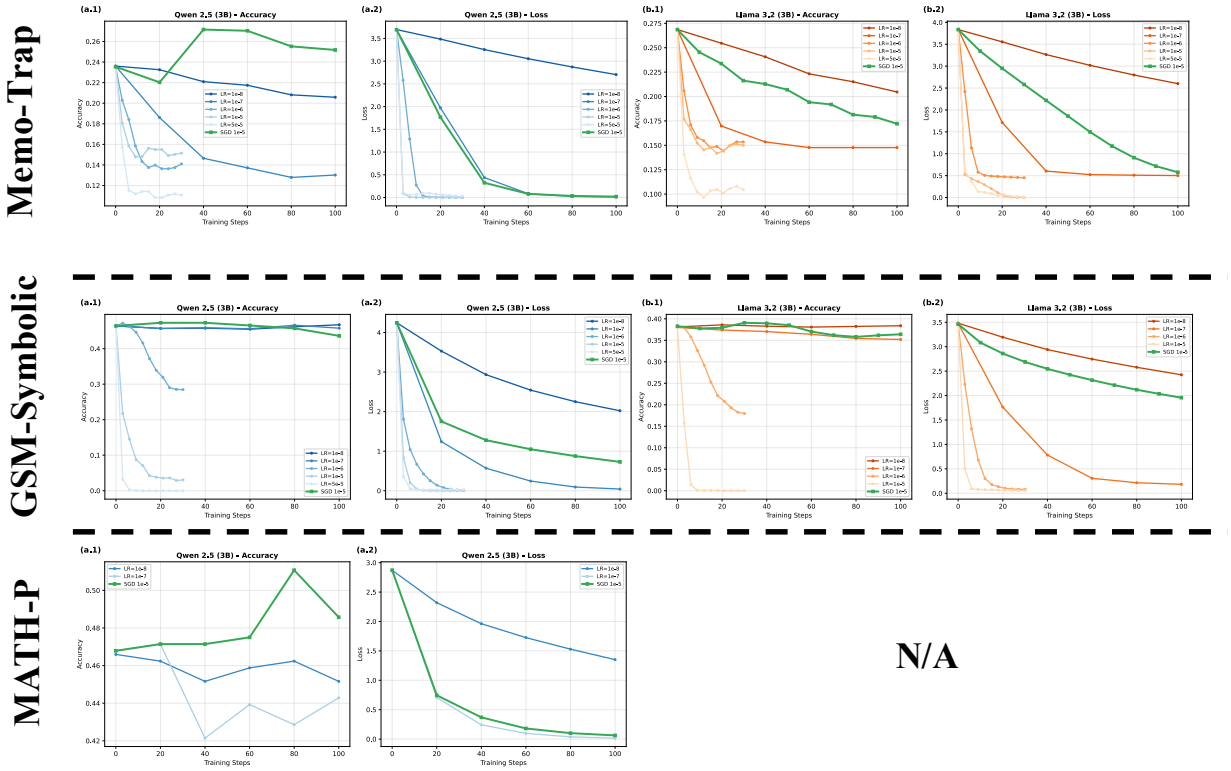

Figure 2: TTT results with Adam optimizers. Figures have the same structure and interpretation as Figure 1. The green curve is the TTT with the SGD optimizer. Although Adam converges quickly, the final performance may be disastrous. SGD converges slowly but performs much better with a similar loss in most cases, indicating the problematic TTT loss landscape in Adam.

training steps exceed 40 with learning rate $1e-5$; Similarly, Qwen 1.5B has a steep performance drops when training steps exceed 20 with learning rate $1e-5$. Most Llama models experience a performance drop.

No universal recipe. It will ease us a lot of burden if we can tell a generating rule of the hyperparameter choice. However, Figure 1 also shows this is not the case. On GSM-Symbolic, learning rate $1e-6$ gives the Qwen 2.5 models a stable improvement, whereas $1e-5$ quickly causes performance drops. On the contrary, $1e-5$ gives Qwen 2.5 3B model a stable improvement on MATH-P, whereas $1e-6$ steadily yields performance drops. Needless to say, different datasets and models have different best training steps. For example, 80 training steps works best for Qwen 2.5 3B on MATH-P but causes low performance on Qwen 2.5 1.5B.

4 Observation 2: Optimizers Have Dramatic Effect

SGD has better stability and performance than Adam. The conventional wisdom of LM optimization is to use the Adam optimizer. However, TTT typically contains only a single example, and we find that Adam often results in a problematic landscape. Figure 2 shows how TTT performs with the Adam optimizer. The green curve denotes TTT with the SGD optimizer. Although Adam optimizer yields a steep loss decay and quickly converges in a few steps, its performance is far worse than SGD with the same or higher loss in most cases. SGD, on the contrary, exhibits smooth loss decay and, more importantly, much better, more stable performance. For example, Qwen 2.5 3B has a very similar loss curve between Adam ($1e-7$) and SGD ($1e-5$) on Memo-Trap and MATH-P, but very different downstream performance. We attribute the catastrophic performance drop in the Adam optimizer to its adaptive estimation. For example, Algorithm 1 shows that the first step of Adam is degenerated to a sign method.

Apprantantly, the sign method has two intuitive negative effects in TTT:

Algorithm 1: First Step of Adam Optimizer - Sign Method Behavior

Require: Learning rate $\alpha > 0$, exponential decay rates $\beta_1, \beta_2 \in [0, 1)$, small constant $\epsilon > 0$
Require: Initial parameter θ_0, gradient $g_1 = \nabla f(\theta_0)$
Initialize: $m_0 = 0$, $v_0 = 0$ ▷ First and second moment estimates
Update biased first moment:
$m_1 = \beta_1 \cdot m_0 + (1 - \beta_1) \cdot g_1 = (1 - \beta_1)g_1$
Update biased second moment:
$v_1 = \beta_2 \cdot v_0 + (1 - \beta_2) \cdot g_1^2 = (1 - \beta_2)g_1^2$
Compute bias-corrected estimates:
$\hat{m}_1 = \frac{m_1}{1-\beta_1^1} = \frac{(1-\beta_1)g_1}{1-\beta_1} = g_1$
$\hat{v}_1 = \frac{v_1}{1-\beta_2^1} = \frac{(1-\beta_2)g_1^2}{1-\beta_2} = g_1^2$
Parameter update:
$\theta_1 = \theta_0 - \alpha \cdot \frac{\hat{m}_1}{\sqrt{\hat{v}_1}+\epsilon} = \theta_0 - \alpha \cdot \frac{g_1}{\sqrt{g_1^2}+\epsilon}$
When $|g_1| \gg \epsilon$**:** $\theta_1 \approx \theta_0 - \alpha \cdot \text{sign}(g_1)$

- Incorrect gradient directions: All gradients are scaled to either 1 or -1
- Large gradient updates: 1 and -1 are generally large gradients

Considering figure 2 reveals that Adam's performance is still low even with $1e-8$ learning rates, we believe the gradient directions may be a key factor that leads to Adam's performance collapse. We leave the concrete analysis as future work. We also noted that SGD becomes Gradient Descent (GD) in the context of TTT; however, we still refer to it as SGD in accordance with common naming conventions.

Another conventional wisdom is to reduce the size of the tunable parameter to slow the fast convergence, which potentially may improve Adam's performance. Unfortunately, Hu et al. (2025b) demonstrates that TTT performance deteriorates dramatically with just a few additional training steps with Adam, even if the training parameter size is small.

Loss is not a strong indicator of performance. Although Adam may achieves similar loss curves as SGD (e.g., Memo-Trap with Qwen 2.5 3B in Figure 2), their performance curve differs dramatically. Figure 1 also suggests that Qwen 2.5 3B with learning rate $1e-4$ and $5e-5$ shares a very similar loss curve but behaves differently in accuracy. Llama also has similar observations between learning rates $1e-3$ and $1e-4$. This reveals that we cannot simply observe loss to determine if the model is well-trained, although this is often the few metric available during a real deployment.

Some prior research (Sun et al., 2020) finds that Adam overfits quickly, but SGD continues to improve model performance in certain vision tasks. We show that Adam not only overfits quickly but also generally hurts model performance in modern LMs.

5 Observation 3: Gradient Norm Makes SGD More Robust

Although SGD shows impressive performance, it still notably overfits as the number of training steps increases. Simply lowering the learning rate to $1e-6$ is not overall beneficial, as model performance may remain the same due to slow convergence or sometimes even decrease (e.g., Qwen 2.5 3B + SGD + MATH-P), according to Figure 1.

Three common approaches to control overfitting are regularization (e.g., weight decay), gradient normalization, and momentum. To avoid overfitting hyperparameters to our test dataset, we select only commonly used values.

Regularization is not helpful to prevent overfitting. 3 shows how the weight decay TTT. We experiment with a value 0.01, which is the default weight decay used in PyTorch AdamW implementation, and a higher value 0.5. Interestingly, adding weight decay does not significantly affect the results, and the learning curves

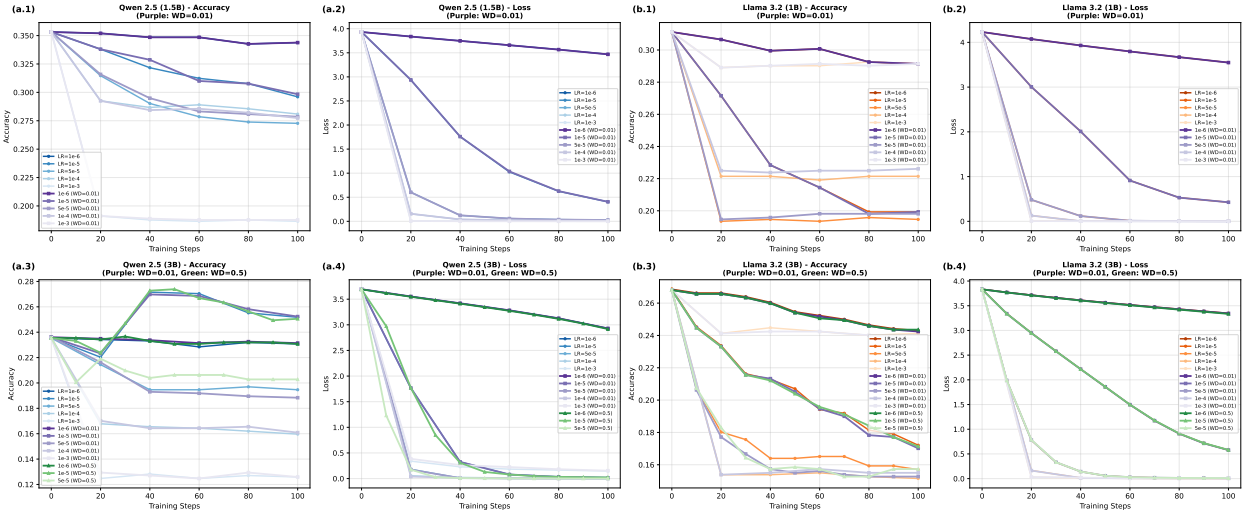

Figure 3: Analysis of weight decay effects on TTT on Memo-Trap dataset. The figure contains the same structure and interpretation as Figure 1. Within each model family, both sizes share the same color palette (blue or orange) to facilitate comparison across model scales. Darker colors indicate higher learning rates within each color scheme. Purple lines with square markers represent ablation experiments with weight decay 0.01, and green curves denote weight decay 0.5. Interestingly, purple and green curves almost overlap with blue curves (i.e., without weight decay).

nearly align with those of the original one without weight decay. This may suggest that TTT only changes a small quantity of parameters, making regularization less effective.

Gradient normalization slows the pace of overfitting. Figure 4 shows the effect brought by gradient normalization. We use the gradient normalization value 10 instead of 1 since the latter converges too slowly. Compared with weight decay, gradient normalization has an obvious effect: it generally mitigates the catastrophic performance drop and sometimes even brings extra gains to the peak performance. For example, Qwen 2.5 3B on Memo-Trap has the worst accuracy $\sim 13\%$, but gradient normalization only has the worst case $\sim 20\%$ accuracy; moreover, two different learning rates with gradient normalization achieve the same high scores, whereas only one learning rate leads to performance gain after TTT without gradient normalization. On GSM-Symbolic, Qwen 2.5 3B achieves even better performance with the help of gradient normalization and TTT. However, gradient normalization is not universally beneficial, as sometimes its peak performance is lower than the original TTT. As the core topic of this paper is to investigate the stability of TTT, the gradient norm is a relatively ideal choice.

Momentum plays a similar role to gradient normalization, but less effectively. Figure 5 shows the effect of momentum on TTT. It generally has the same effect as gradient normalization. However, since it sometimes leads to a more catastrophic performance (e.g., Qwen 2.5 1.5B with momentum), using gradient normalization is more favorable.

6 Observation 4: Tuning FFNs leads to better peak performance, whereas tuning attention leads to better worst performance

A straightforward question is to lower the tunable parameter size to mitigate overfitting. Prior works (Hu et al., 2025b) have shown that overfitting is common even if the tunable parameter $\in \mathbb{R}^{1\times d}$ only builds on top of the final representation, where d is the hidden dimension. In this section, we are more interested in understanding which parameters contribute more to TTT. Specifically, we only tune FFNs or attention during the TTT, and report results in Figure 6. Qwen 2.5 3B models roughly have $340M$ parameters for attention and $2.4B$ parameters for FFNs, which are all more than sufficient to fit the single test-time example. In Llama 3, the performance curves fit the conventional expectation that tuning FFNs leads to

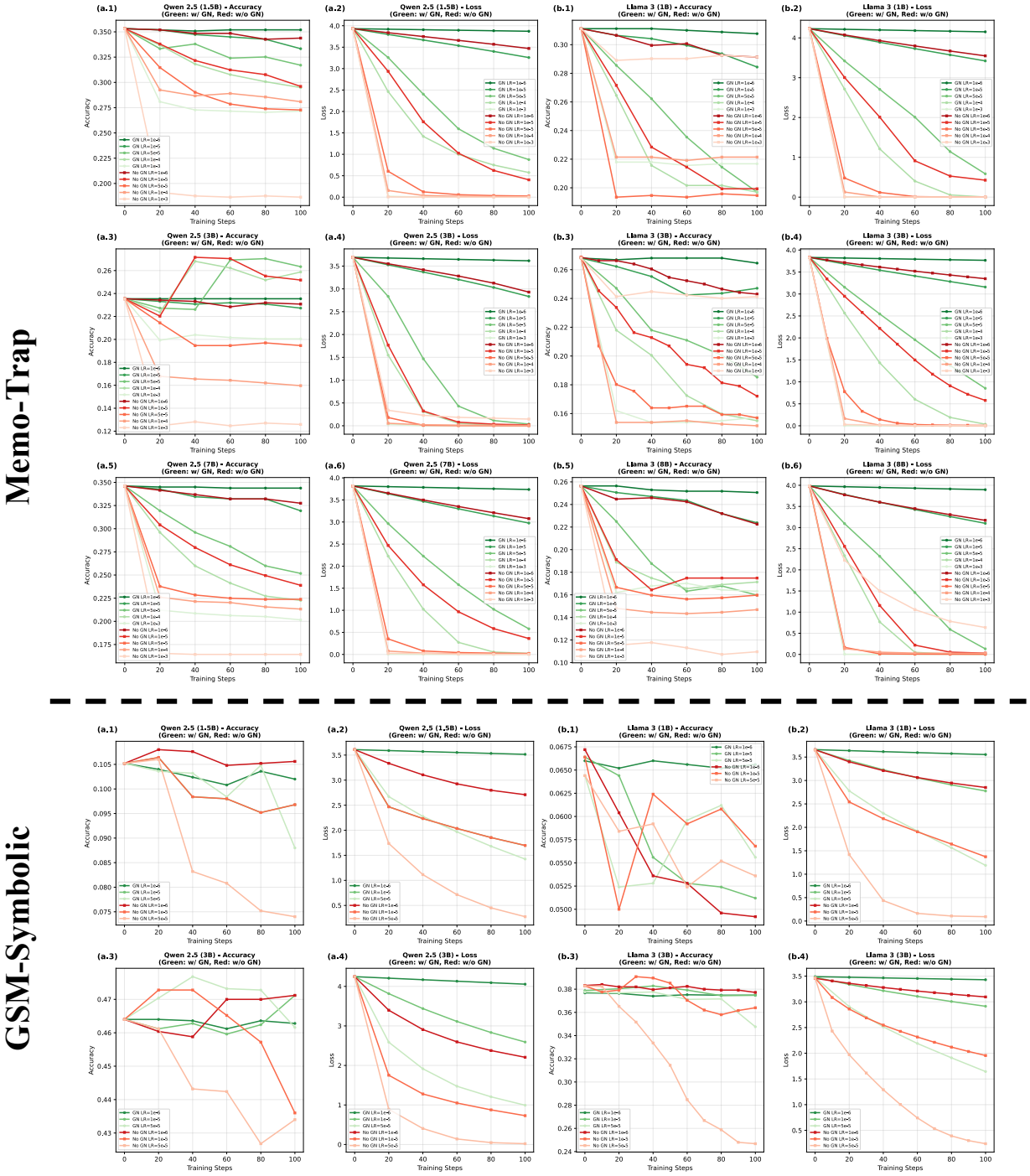

Figure 4: TTT with gradient normalization. Red/Green curves represent TTT without/with gradient normalization, respectively. The figure structure and interpretation are the same as Figure 1 elsewhere. Two core messages are conveyed in this figure: (1) Gradient normalization can generally mitigate catastrophic performance drop, and (2) sometimes yields better performance than without gradient normalization. Although gradient normalization sometimes also brings lower peak performance, its stability makes its a good choice for TTT.

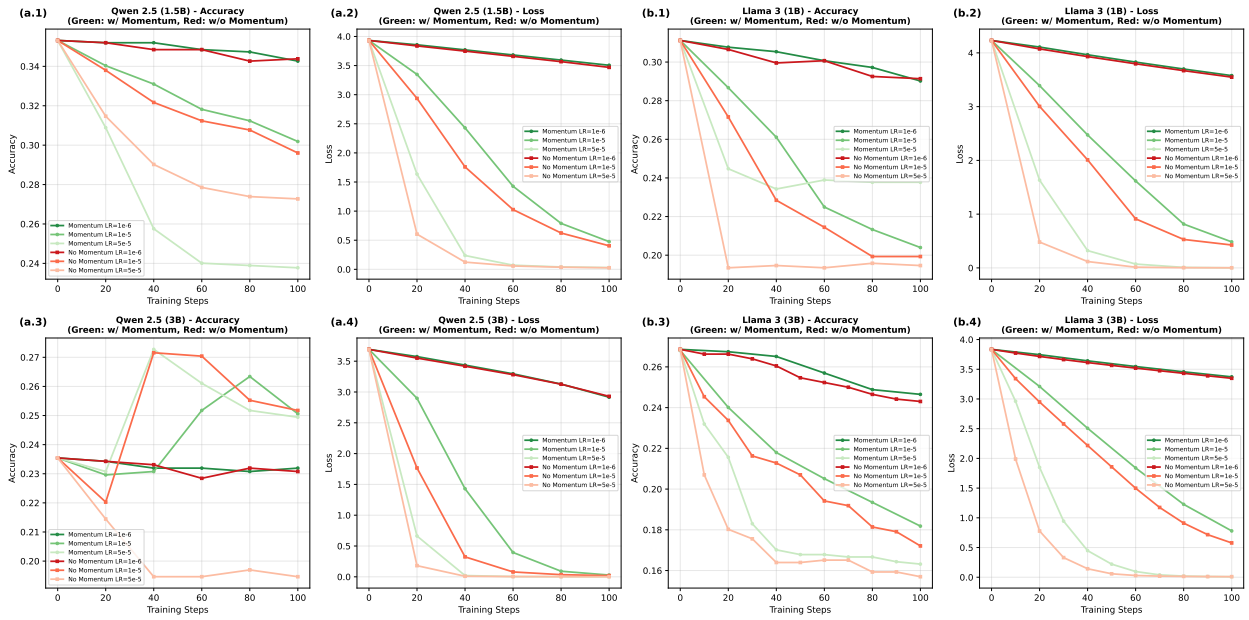

Figure 5: Results of TTT with Momentum on Memo-Trap. Red/Green curves represent TTT without/with momentum, respectively. The figure structure and interpretation are the same as Figure 1 elsewhere. Momentum plays a similar role to gradient normalization, mitigating catastrophic performance drops and providing a stabler TTT curve. However, unlike gradient normalization, it sometimes leads to even worse catastrophic performance drop (e.g., Qwen 2.5 1.5B with momentum).

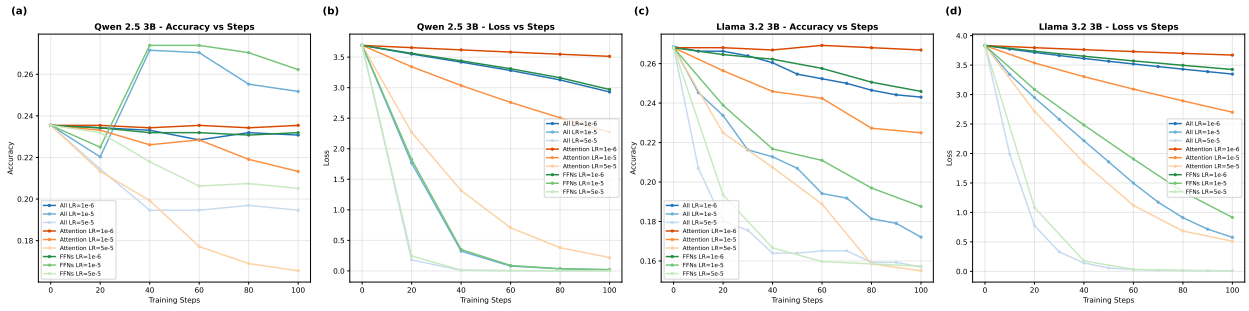

Figure 6: TTT performance when tuning FFNs or Attentions on Memo-Trap dataset. The red/green/blue curves denote training on attention/FFNs/all parameters, respectively. Although FFNs have many more parameters than attention modules, they both are far more than sufficient to fit a single test-time example. Nevertheless, training FFNs achieve much higher peak performance than training attention modules, suggesting the functionality differences between FFNs and attention modules.

easier overfitting. However, performance with Qwen 2.5 shows that training FFNs may yield a better peak performance even with much larger parameters, which suggests a potential difference in functionality between FFNs and attentions. One hypothesis is that FFNs serve as a memory bank (Geva et al., 2021), which is more suitable for TTT.

7 Related Work

Test-time Training. Originally introduced for domain generalization (Sun et al., 2020), TTT updates model parameters using a self-supervised loss on each test example. Subsequent work extended this idea to masked autoencoders (Gandelsman et al., 2022), video steams (Wang et al., 2025), and multimodal generation (Dalal et al., 2025). Recent studies adapted TTT to large language models, typically via input-perplexity minimization (Hu et al., 2025b;a) or RL-based reward optimization (Zuo et al., 2025), reporting gains on benchmarks such as GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021a) and AIME. Akyürek et al. (2024) find that input-perplexity minimization has a surprisingly good effect on the in-context learning task —— using few-show examples to train models before doing in-context learning yields an obvious lift than doing in-context learning directly. Despite the success of TTT, popular benchmarks are known to suffer from possible data contamination (Wu et al., 2025), leaving the true generalization capability of TTT unclear. More recently, Sun et al. (2024) proposes TTT-style layers whose hidden states are dynamically adapted by self-supervised learning on test sequences. They show that such layers reduce perplexity as more context is seen, opening a path toward sequence-adaptive architectures. Hardt & Sun (2024) use a retrieval-based scheme: for each test example, the model retrieves a small set of nearest neighbors and performs a single gradient update on them before inference. This work offers a practical baseline for TTT in LMs and highlights issues of index quality and overfitting risks under adaptation.

Optimization and stability. Existing TTT approaches for LLMs commonly employ the Adam optimizer (Kingma & Ba, 2014) with fixed hyperparameters inherited from fine-tuning practice (Hu et al., 2025b). Yet, TTT operates in a distinct single-example optimization regime where such settings may be suboptimal. Earlier work in vision tasks hinted that SGD could provide smoother convergence, but a systematic comparison for language models has been missing. Moreover, while prior studies explored regularization techniques such as weight decay or momentum to stabilize adaptation, their effectiveness for large-scale TTT remains limited (Sun et al., 2024).

Parameter-efficient adaptation. Prior arts often tune all parameters of a language model during TTT. A complementary line of work studies parameter-efficient fine-tuning (PEFT), which adapts models by updating only a small subset of parameters through mechanisms such as adapters (Houlsby et al., 2019), LoRA (Hu et al., 2022), and prompt or prefix tuning (Ding et al., 2023; Li & Liang, 2021). These methods improve stability and efficiency by decoupling adaptation capacity from full model updates, inspiring recent extensions of test-time adaptation that operate via adapter or prompt layers (Gao et al., 2022; Wang et al., 2022).

8 Conclusion and Discussion

This paper presents a preliminary but critical analysis of the robustness and practical applicability of entropy-based test-time training for language models. Through extensive experiments on anti-memorization datasets with multiple model families and sizes, we uncover several important findings that challenge the current understanding of TTT and point toward future research directions.

First, we demonstrate that TTT's effectiveness is not universal. While it shows promise on popular benchmarks like AIME 2024, it struggles significantly with tasks specifically designed to counter memorization, such as Memo-Trap, GSM-Symbolic, and Math-Perturb. This discrepancy raises important questions about the underlying mechanisms of TTT—whether it primarily performs domain adaptation or relies on recalling memorized patterns from training data.

Second, we identify a dramatic and previously underappreciated difference between optimizers in the TTT setting. Despite Adam's dominance in conventional language model training, SGD consistently outperforms

Adam by substantial margins in TTT scenarios. Our analysis suggests this stems from Adam's sign-method-like behavior in single-example optimization, which leads to incorrect gradient directions and unstable performance. This finding has immediate practical implications for practitioners implementing TTT systems.

Third, we reveal significant hyperparameter sensitivity across different models, model sizes, and tasks. There is no universal recipe for selecting learning rates and training steps—what works optimally for one task may cause catastrophic performance drops on another. This poses a fundamental challenge for real-world deployment, where validation sets are typically unavailable for selecting appropriate hyperparameters for test-time examples.

Fourth, among various regularization approaches, we find that gradient normalization is the most effective technique for improving TTT robustness. It successfully mitigates catastrophic performance drops, reduces sensitivity to learning rate choices, and sometimes even improves peak performance. Weight decay, despite its widespread use in standard training, shows minimal impact on preventing overfitting in TTT. Momentum provides similar benefits to gradient normalization but proves less reliable.

Finally, our analysis of training different model components reveals interesting patterns. Training only feed-forward networks can match or exceed full model fine-tuning performance, supporting the hypothesis that FFNs serve as memory banks. Meanwhile, training only attention modules, despite having fewer parameters, provides more stable worst-case performance, suggesting functional differences between these components that warrant further investigation.

Limitations and Future Work. This study represents a preliminary analysis with several limitations. First, our experiments focus on three specific anti-memorization tasks, which, while informative, may not capture the full spectrum of real-world applications. Second, we limit our hyperparameter choices to common values to avoid overfitting to our test scenarios, but a more comprehensive search might reveal additional insights. Third, our analysis of why SGD outperforms Adam remains incomplete—while we identify gradient direction as a key factor, a deeper theoretical understanding is needed. Fourth, we only experimented with the input-perplexity minimization method; therefore, our conclusion may not hold for all TTT tasks. Fifth, we focus only on single-example optimization and do not use additional examples, which is closer to domain adaptation. As test-time training becomes a new paradigm for scaling LMs, we believe specialized optimizers for TTT are one promising direction. As test-time training becomes a new paradigm for scaling LMs, we believe specialized optimizers for TTT is one promising direction.

References

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

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. *arXiv preprint arXiv:2110.14168*, 2021.

Karan Dalal, Daniel Koceja, Jiarui Xu, Yue Zhao, Shihao Han, Ka Chun Cheung, Jan Kautz, Yejin Choi, Yu Sun, and Xiaolong Wang. One-minute video generation with test-time training. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 17702–17711, 2025.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature machine intelligence*, 5(3):220–235, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Yossi Gandelsman, Yu Sun, Xinlei Chen, and Alexei Efros. Test-time training with masked autoencoders. *Advances in Neural Information Processing Systems*, 35:29374–29385, 2022.

Yunhe Gao, Xingjian Shi, Yi Zhu, Hao Wang, Zhiqiang Tang, Xiong Zhou, Mu Li, and Dimitris N Metaxas. Visual prompt tuning for test-time domain adaptation. *arXiv preprint arXiv:2210.04831*, 2022.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories, 2021. URL https://arxiv.org/abs/2012.14913.

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021a.

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. *NeurIPS*, 2021b.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

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.

Jinwu Hu, Zhitian Zhang, Guohao Chen, Xutao Wen, Chao Shuai, Wei Luo, Bin Xiao, Yuanqing Li, and Mingkui Tan. Test-time learning for large language models. *arXiv preprint arXiv:2505.20633*, 2025a.

Yang Hu, Xingyu Zhang, Xueji Fang, Zhiyang Chen, Xiao Wang, Huatian Zhang, and Guojun Qi. Slot: Sample-specific language model optimization at test-time. *arXiv preprint arXiv:2505.12392*, 2025b.

Kaixuan Huang, Jiacheng Guo, Zihao Li, Xiang Ji, Jiawei Ge, Wenzhe Li, Yingqing Guo, Tianle Cai, Hui Yuan, Runzhe Wang, Yue Wu, Ming Yin, Shange Tang, Yangsibo Huang, Chi Jin, Xinyun Chen, Chiyuan Zhang, and Mengdi Wang. MATH-Perturb: Benchmarking LLMs' math reasoning abilities against hard perturbations. *arXiv preprint arXiv:2502.06453*, 2025.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*, 2020.

America Mathematical Association. American invitational mathematics examination (AIME) [2024], 2024.

Ian R. McKenzie, Alexander Lyzhov, Michael Pieler, Alicia Parrish, Aaron Mueller, Ameya Prabhu, Euan McLean, Aaron Kirtland, Alexis Ross, Alisa Liu, Andrew Gritsevskiy, Daniel Wurgaft, Derik Kauffman, Gabriel Recchia, Jiacheng Liu, Joe Cavanagh, Max Weiss, Sicong Huang, The Floating Droid, Tom Tseng, Tomasz Korbak, Xudong Shen, Yuhui Zhang, Zhengping Zhou, Najoung Kim, Samuel R. Bowman, and Ethan Perez. Inverse scaling: When bigger isn't better, 2024. URL https://arxiv.org/abs/2306.09479.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candes, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Workshop on Reasoning and Planning for Large Language Models*, 2025.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

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. In *First Conference on Language Modeling*, 2024.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pp. 9229–9248. PMLR, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

Renhao Wang, Yu Sun, Arnuv Tandon, Yossi Gandelsman, Xinlei Chen, Alexei A Efros, and Xiaolong Wang. Test-time training on video streams. *Journal of Machine Learning Research*, 26(9):1–29, 2025.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. Adamix: Mixture-of-adaptations for parameter-efficient model tuning. *arXiv preprint arXiv:2205.12410*, 2022.

Mingqi Wu, Zhihao Zhang, Qiaole Dong, Zhiheng Xi, Jun Zhao, Senjie Jin, Xiaoran Fan, Yuhao Zhou, Huijie Lv, Ming Zhang, et al. Reasoning or memorization? unreliable results of reinforcement learning due to data contamination. *arXiv preprint arXiv:2507.10532*, 2025.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, et al. Ttrl: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*, 2025.