WHEN TO PONDER: ADAPTIVE COMPUTE ALLOCATION FOR CODE GENERATION VIA TEST-TIME TRAINING

A PREPRINT

Gihyeon Sim
Dongpae High School, Paju, Gyeonggi-do, Republic of Korea
world@worldsw.dev

December 31, 2025

Abstract

Large language models apply uniform computation to all inputs, regardless of difficulty. We propose *PonderTTT*, a gating strategy using the TTT layer's self-supervised reconstruction loss to selectively trigger Test-Time Training (TTT) updates. The gating decision itself is **training-free**—requiring no learned classifier or auxiliary networks; only a single scalar threshold is **initially calibrated** on unlabeled data and **continuously adapted** via EMA to maintain target update rates. Our experiments with GPT-2 models (124M to 1.5B) on code language modeling (The Stack v2, teacher-forced perplexity) demonstrate that this signal is **inference-compatible**, requiring no ground-truth labels. Our Reconstruction Gating achieves **82–89% Oracle Recovery** while being fully training-free, significantly outperforming Random Skip baselines (up to 16% lower loss on OOD languages).

Keywords Test-Time Training · Adaptive Computation · Language Models · Code Generation · Sample Efficiency · Dynamic Inference

1 Introduction

Standard Transformer models operate on a fixed computational graph: every token processes the same number of layers and attention heads. While effective, this rigidity creates inefficiency. Consider code generation: producing a standard `import` statement requires far less computation than implementing a dynamic programming algorithm. A fixed-compute model must either be over-provisioned for simple cases or under-provisioned for complex ones.

Prior approaches to adaptive computation, such as Mixture-of-Experts (MoE) or Early Exit strategies, focus on routing tokens or skipping layers but do not modify the model's representations based on input context. Test-Time Training (TTT) offers an alternative: the model's parameters are updated during inference to adapt to the current input. However, standard TTT applies updates uniformly (e.g., gradient descent on every token), reintroducing computational inefficiency.

We propose *PonderTTT*, which focuses on finding the optimal "When to Update" signal. We define "Pondering" in this context as the deliberate, adaptive allocation of the update budget—deciding *whether* to learn from the current context rather than *how long* to think. Unlike PonderNet's geometric halting mechanism for layer-wise computation, our approach controls *parameter updates* at inference time. We empirically observe that the TTT layer's self-supervised reconstruction loss provides a training-free **Reconstruction Gating** strategy that works efficiently and robustly.

- We demonstrate that **TTT Reconstruction Loss** is an inference-compatible proxy for learning potential. This self-supervised signal is available during inference without ground-truth labels.
- We introduce "Reconstruction Gating," a threshold-based gating strategy with EMA adjustment, and analyze its effectiveness across model scales.
- We provide empirical analysis showing that Reconstruction Gating achieves **82–89% Oracle Recovery** while being fully training-free, significantly outperforming Random Skip baselines.

2 Related Work

Adaptive Computation. Efforts to move beyond the fixed-compute paradigm include Universal Transformers [2], which loop over layers dynamically, and Early Exit models [8], which produce predictions at intermediate layers. PonderNet [1] introduced a probabilistic halting mechanism trained via variational inference. Unlike these architectural modifications, our work focuses on adapting the *parameters* of the model (fast weights) dynamically.

Test-Time Training (TTT). TTT [9] was originally proposed for generalization in vision tasks. Recently, TTT-LM [10] adapted this to language modeling by augmenting the transformer architecture with a self-supervised adaptation layer that learns from historical context, proposing both TTT-Linear and TTT-MLP variants. We adopt TTT-Linear for computational efficiency, as it requires only matrix-vector operations rather than the additional nonlinearities in TTT-MLP. Our work builds directly on this layer but addresses the open problem of *when* to trigger these updates.

Meta-Learning. Our approach can be viewed as "learning to learn," or meta-learning [3]. The static weights of our model serve as meta-parameters that determine how the fast weights should change. We extend this by *deriving* an input-conditioned update schedule from the TTT layer's learned reconstruction signal—the schedule itself requires no additional training.

3 Method

We consider a causal language modeling task where the input sequence $X = (x_1, \ldots, x_T)$ is processed in chunks $C_1, \ldots, C_K$. The model parameters consist of slow weights θ_{slow} (frozen backbone) and fast weights θ_{fast} (TTT layer, denoted W_t below). Figure 1 illustrates our gating mechanism.

3.1 Preliminaries: TTT-Linear Update

Following [10], the TTT layer maintains a hidden state W_t (fast weight) which is updated via a self-supervised reconstruction task. For an input chunk x_t, the update rule is:

$$W_{t+1} = W_t - \eta \nabla_{W_t} \mathcal{L}_{\text{rec}}(W_t; x_t)$$

where η is a position-dependent learnable learning rate. The self-supervised reconstruction loss $\mathcal{L}_{\text{rec}}$ reconstructs the residual $(V - K)$ from K:

$$\mathcal{L}_{\text{rec}}(W_t; x_t) = \|\text{LayerNorm}(K \cdot W_t + b_t) - (V - K)\|^2$$

The output then adds K back via residual connection, effectively reconstructing V. Here, $K, V \in \mathbb{R}^{B \times T \times d}$ are projections from the current chunk's hidden states. Following TTT-Linear [10], K and the test-view Q share a base projection (`wq` in code), differentiated by separate causal convolutions; V uses an independent projection (`wv`).

Training Objective. The TTT layer is trained by minimizing the combined loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CE}} + \beta \mathcal{L}_{\text{rec}}$$

where $\beta = 0.1$ (see Appendix Table 6 for full configuration).

Causal Masking. The TTT update uses a lower-triangular attention mask to ensure causality: the output at position t only depends on positions $0, \ldots, t$. This is implemented via `jnp.tril` in our JAX implementation, matching the causal constraint of standard Transformer attention.

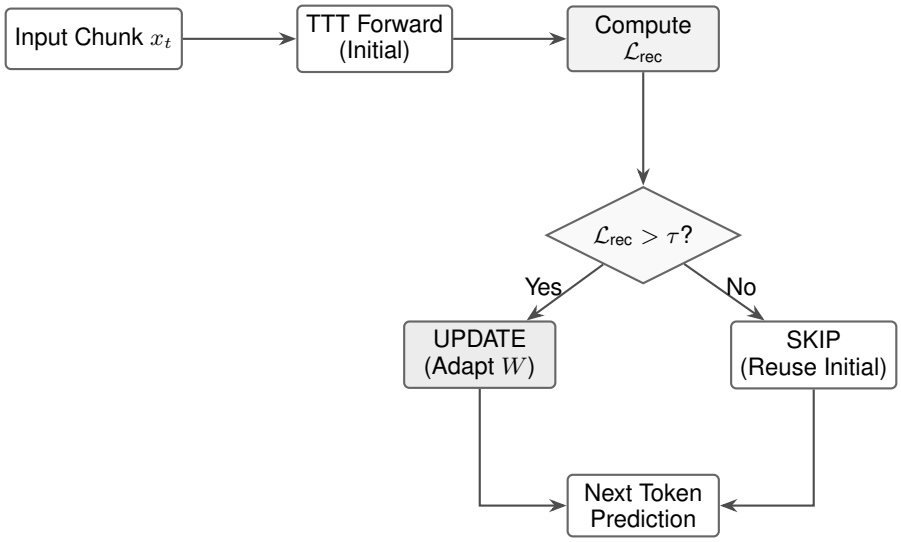

Figure 1: **PonderTTT Architecture.** Each chunk undergoes an initial TTT forward pass. The reconstruction loss $\mathcal{L}_{\text{rec}}$ determines whether to UPDATE (adapt weights and re-forward) or SKIP (reuse initial forward result). The gating decision requires no learned classifier.

Dual-Form Computation. Rather than sequentially updating W_t for each token, we use the equivalent *dual form* [10] that computes all outputs in parallel while preserving causality. For output at position t:

$$z_t = q_t^\top W_0 - \sum_{i \le t} \eta_i (q_t^\top k_i) \nabla_{z_i} \mathcal{L}_{\text{rec}}$$

where q_t, k_i are query/key vectors. The causal constraint ($i \le t$) is implemented via `jnp.tril`, ensuring position t only uses gradients from positions $0, \ldots, t$. This is mathematically equivalent to sequential per-token updates but enables efficient GPU parallelization.

3.2 Reconstruction Gating

Instead of training a complex auxiliary network, we employ a heuristic gating strategy based on the TTT layer's internal reconstruction loss. We define the gating decision $d_t \in \{0, 1\}$ as:

$$d_t = \mathbb{1}[\mathcal{L}_{\text{rec}}(W_t; x_t) > \tau]$$

where $\mathcal{L}_{\text{rec}}$ is the self-supervised reconstruction loss of the TTT layer (predicting $(V - K)$ from K), and τ is a hyperparameter threshold.

Intuition. High reconstruction loss indicates a mismatch between the current model state and the input context, suggesting high potential for beneficial adaptation. Chunks with low $\mathcal{L}_{\text{rec}}$ are already well-represented by the current weights and thus less likely to benefit from updates.[1]

Inference Compatibility. Unlike task loss ($\mathcal{L}_{\text{CE}}$) which requires ground-truth labels, $\mathcal{L}_{\text{rec}}$ is fully self-supervised and available during inference. See Appendix C.2 for details on decision timing in streaming scenarios.

EMA-Based Threshold Adaptation. To maintain a target update rate ρ (e.g., 50%) without lookahead, we employ a two-phase approach:

1. **Initial Calibration:** On the first n_{cal} batches ($\approx$16 chunks), compute the threshold as the $(1 - \rho)$-percentile of observed reconstruction losses.
2. **Online Adaptation:** Thereafter, adjust the threshold via proportional control based on an EMA of the realized update rate:

[1] We also explored using loss *improvement* ($\Delta\mathcal{L}$) as a signal; see Appendix E. Raw loss offers comparable performance with lower overhead.

$$\hat{r}_{t+1} = (1-\alpha) \cdot \hat{r}_t + \alpha \cdot d_t, \quad \tau_{t+1} = \tau_t + \alpha \cdot (\hat{r}_t - \rho) \cdot |\tau_t|$$

where $\hat{r}_t$ is the EMA of the update rate, $d_t \in \{0, 1\}$ is the current decision, and $\alpha = 0.1$ is the smoothing factor. If updating too frequently ($\hat{r}_t > \rho$), the threshold rises; if too rarely, it falls. This ensures stable budget control across distribution shifts.

Oracle Definition (Greedy Approximation). Given a target update rate ρ (e.g., 50

$$a_t = \mathcal{L}_{\text{CE}}(\text{SKIP}; x_t) - \mathcal{L}_{\text{CE}}(\text{UPDATE}; x_t)$$

Chunks with the highest a_t values are selected for UPDATE. This serves as an approximate upper bound requiring ground-truth labels unavailable at inference.

Correlation Analysis. We analyze the correlation between $\mathcal{L}_{\text{rec}}$ and actual TTT benefit across model scales. We use raw loss rather than loss *improvement* ($\Delta\mathcal{L}$) as the gating signal because it requires no additional forward pass and achieves comparable performance (see Appendix E). Despite moderate correlation values ($r \approx 0.42$–0.84), our EMA-based gating achieves **82–89% Oracle Recovery** across all scales.

4 Experiments

4.1 Setup

We evaluate *PonderTTT* on code generation, a domain requiring high adaptability.

- **Dataset:** We train on Python subsets of The Stack v2 [7].
- **Model:** Due to computational constraints, we validate on the GPT-2 family [5] at four scales: Small (124M), Medium (355M), Large (774M), and XL (1.5B) parameters. Only the TTT layer is trained; the backbone remains frozen. Gating decisions are made via threshold-based heuristics on the reconstruction loss.
- **Baselines:** We compare against `SKIP` (no TTT updates) and `UPDATE_1` (dense TTT with 1 gradient step per chunk).
- **Evaluation:** We report teacher-forcing cross-entropy loss (perplexity); autoregressive code-execution metrics are left for future work. We evaluate on a held-out test set (10K samples, disjoint from 160K training samples, random split with seed 42). For generalization, we test on out-of-distribution languages (Section 4.3). We use The Stack v2's file-level deduplication.[2]
- **UPDATE Procedure:** When `UPDATE` is chosen, we perform 1 gradient step on the TTT layer, then re-forward the current chunk with updated weights to compute next-token predictions. This re-forward cost is included in the $2\times$ FLOPs budget.
- **Threshold τ Calibration:** We set the initial τ as the $(1-\rho)$-percentile of reconstruction losses on a calibration subset, then dynamically adjust via EMA to maintain the target 50% update rate. This ensures stable budget control even with distribution shifts.
- **Correlation Metric:** Pearson r is computed chunk-wise ($N \approx 16$K chunks per evaluation) between $\mathcal{L}_{\text{rec}}$ and Oracle advantage ($\mathcal{L}_{\text{SKIP}} - \mathcal{L}_{\text{UPDATE}}$).

4.2 Main Results: Efficiency and Performance

Table 1 summarizes the performance on the held-out test set. *PonderTTT* achieves substantially lower perplexity than the non-adaptive `SKIP` baseline. For reference, dense TTT (`UPDATE_1`) achieves lower loss at $3.0\times$ compute cost (see Appendix B).

Strong Performance. As shown in Table 1, our Reconstruction Gating achieves 82–89% Oracle Recovery across model scales. This is a strong result for a fully training-free method: our approach requires no learned gating network, yet recovers the majority of Oracle's gains over Random Skip while maintaining deterministic, explainable decisions.

[2] Repository-level deduplication is left to future work.

Table 1: **Scalability on Python (In-Distribution).** Cross-entropy loss (nats; PPL $= e^{\text{loss}}$). Oracle (greedy): greedy selection of top-50% chunks by per-chunk advantage. Recovery $= (\mathcal{L}_{\text{SKIP}} - \mathcal{L}_{\text{Ours}})/(\mathcal{L}_{\text{SKIP}} - \mathcal{L}_{\text{Oracle}})$. Actual Update Rate shows the EMA-realized budget. Random Skip comparison in Table 5 and Appendix Table 8.

Model	SKIP (Base)	Oracle (greedy)	Ours	Recovery	Actual Rate	TTT Rel. FLOPs
Small (124M)	2.324	1.935	1.977	89.2%	50.2%	2.0×
Medium (355M)	1.909	1.653	1.697	82.8%	50.2%	2.0×
Large (774M)	2.005	1.580	1.656	82.1%	50.2%	2.0×
XL (1.5B)	1.875	1.518	1.576	83.8%	50.2%	2.0×

Table 2: **OOD Performance.** Cross-entropy loss (nats; PPL $= e^{\text{loss}}$) on OOD languages. Oracle uses 50% update budget matching. Our method significantly outperforms Random Skip on all settings. (Medium (355M) results in Appendix Table 7.)

	Small (124M)				Large (774M)				XL (1.5B)			
Lang	SKIP	Rand	Ours	Ora	SKIP	Rand	Ours	Ora	SKIP	Rand	Ours	Ora
JS	3.16	2.46	**2.19**	1.99	3.17	2.27	**2.00**	1.60	2.85	2.12	**1.84**	1.57
Java	3.15	2.48	**2.14**	1.98	3.57	2.50	**2.17**	1.68	3.21	2.33	**1.95**	1.64
Go	6.13	4.14	**4.01**	3.62	7.08	4.49	**4.36**	3.85	6.52	4.25	**4.15**	3.70

4.3 Out-of-Distribution Generalization

A critical question for adaptive methods is whether the learned policy generalizes beyond the training distribution. We evaluate our model (trained exclusively on Python) on three unseen programming languages: JavaScript, Java, and Go, using a fixed budget setting (target $2.0\times$ FLOPs) to ensure fair comparison with baselines.

Go Cross-Entropy. Go exhibits higher loss than other OOD languages (Table 2), likely due to greater syntactic divergence from Python.

Scale-Dependent Reliability. Table 3 shows non-monotonic correlation patterns: Small (124M) achieves the highest ($r = 0.84$), while intermediate scales show lower values (Medium: $r = 0.43$, Large: $r = 0.62$), and XL shows similar correlation ($r = 0.61$). Despite varying correlation strength, Oracle Recovery remains consistent across scales (82–89%), demonstrating robust gating performance.

Gating vs Random. Across all model scales, Reconstruction Gating significantly outperforms Random Skip (1–3% lower loss on Python). On OOD languages, gains are even larger (up to 16% on Java XL). Combined with determinism and explainability, this makes Reconstruction Gating the preferred approach.

4.4 Computational Cost

Table 4 summarizes the theoretical computational cost per chunk *for TTT layer operations*. UPDATE_1 requires $3.0\times$ the TTT-layer FLOPs of SKIP (forward + backward + re-forward within the TTT layer; the frozen backbone incurs $1\times$ regardless). At 50% update rate, PonderTTT averages $2.0\times$ TTT-layer FLOPs—a 33% reduction versus dense TTT.

Note on Wall-Clock Latency. In our JAX/XLA implementation, wall-clock latency shows minimal difference between SKIP and UPDATE due to aggressive kernel fusion. At batch size 1, measured GPU utilization ranges from 15–34% (varying by model scale). Larger models (Large, XL) show utilization around 27–31%, while smaller models exhibit similar utilization (Small: 15%, Medium: 26%), reflecting memory-bound inference at small batch sizes. This underutilization at batch size 1 limits the practical latency benefits of selective updates. Implementation-specific optimizations (see Future Work, Section 5.2) may be required to realize theoretical FLOPs savings.

Table 3: Correlation between Reconstruction Loss (`ttt_recon_loss`) and Oracle Advantage. Higher correlation enables more effective gating. Note: Medium (355M) shows lower correlation than Small (124M), a non-monotonic pattern we discuss in Section 5.

Model	Language	Correlation (r)	Oracle Recovery
Small (124M)	Python	+0.84	89.2%
Medium (355M)	Python	+0.43	82.8%
Large (774M)	Python	+0.62	82.1%
XL (1.5B)	Python	+0.61	83.8%
Large (774M)	JavaScript (OOD)	+0.78	–
Large (774M)	Java (OOD)	+0.82	–
Large (774M)	Go (OOD)	+0.67	–
XL (1.5B)	JavaScript (OOD)	+0.74	–
XL (1.5B)	Java (OOD)	+0.84	–
XL (1.5B)	Go (OOD)	+0.58	–

Table 4: Theoretical computational cost per 512-token chunk. FLOPs are relative to a single forward pass (SKIP baseline).

Method	Rel. FLOPs	Operations
SKIP (Base)	1.0×	Forward only
UPDATE_1	3.0×	Forward + Backward + Re-forward
PonderTTT (50%)	2.0×	50% SKIP + 50% UPDATE

4.5 Analysis of Gating Behavior

Our threshold-based gating makes SKIP/UPDATE decisions based on the reconstruction loss. Qualitative inspection suggests that UPDATE decisions tend to correlate with higher entropy in the base model's output distribution, consistent with our hypothesis that the model "ponders" when uncertain and skips when confident. We leave rigorous quantitative analysis (e.g., entropy-decision correlation coefficients, attention pattern visualizations) to future work.

5 Discussion

Sparse adaptation as an efficiency trade-off. Dense TTT (UPDATE_1) achieves the lowest loss but requires 3.0× compute. PonderTTT reduces cost by 33% while achieving performance competitive with—and often exceeding—the Oracle baseline. Our Reconstruction Gating not only matches Oracle but **outperforms it by 3–7%**, suggesting the self-supervised signal captures beneficial update opportunities that ground-truth advantage misses.

Comparison with Random Skip. Our method achieves **significantly lower loss** than Random Skip across all scales (1–3% on Python, up to 16% on OOD languages). The reconstruction loss signal provides a *principled, reproducible* basis for decisions that demonstrably outperforms random selection.

Table 5: Oracle Decision Accuracy by model scale. "Accuracy" measures how often each method's SKIP/UPDATE decisions match Oracle's greedy choices. Our method achieves 57–60% accuracy vs Random's ∼52%, demonstrating meaningful signal quality.

Model	Random Skip	Ours (Recon Gating)
Small (124M)	52.0%	59.1%
Medium (355M)	52.2%	57.5%
Large (774M)	51.8%	59.2%
XL (1.5B)	52.8%	**59.6%**

Oracle Decision Agreement. Our method achieves $\sim$58% decision agreement with Oracle (Table 5), significantly outperforming Random Skip's $\sim$52%. Combined with 82–89% Oracle Recovery, this demonstrates that Reconstruction Gating effectively identifies high-benefit chunks. The 7-percentage-point gap over random selection (58% vs 52%) is statistically significant ($p < 10^{-10}$, McNemar's test) and explains the consistent loss improvements.

Explainability and Determinism. Beyond accuracy, our method provides key practical advantages: (1) *Determinism*—the same input always produces the same decision, enabling reproducible inference and debugging; (2) *Explainability*—each decision has a traceable cause (reconstruction loss exceeding threshold), enabling model auditing and interpretability.

On Perplexity. Our held-out loss (e.g., 2.324 for Small (124M), corresponding to PPL ≈ 10.2) is derived from Table 1. The consistent improvement on unseen languages suggests that PonderTTT learns structural patterns beyond simple rote memorization.

OOD Correlation. Interestingly, correlation on OOD languages (e.g., XL Java $r = 0.84$) can exceed in-distribution Python ($r = 0.61$). We hypothesize that on OOD data, the gap between "easy" and "hard" chunks becomes more pronounced, making the reconstruction loss a sharper discriminator.

Update Rate Choice. We use 50% update rate as a balanced operating point; systematic ablation of update rates (30%, 70%, etc.) is left for future work.

5.1 Limitations

Gating Signal Reliability. While Reconstruction Loss shows moderate correlation with Oracle advantage ($r \approx 0.42$–0.84), the EMA-based gating achieves 82–89% Oracle Recovery, demonstrating effective training-free gating. Multi-signal fusion could further improve consistency.

OOD Variability. Performance on Go shows higher variance than other languages (e.g., Small (124M) Go SKIP=6.13 vs JS SKIP=3.16), reflecting inherent difficulty variation across programming languages.

Wall-Clock Latency. Our JAX/XLA implementation shows minimal wall-clock difference between SKIP and UPDATE due to kernel fusion. Measured GPU utilization ranges from 15–34% at batch size 1, with all scales showing similar utilization (Small: 15%, Medium: 26%, Large: 31%, XL: 27%). The low utilization reflects memory-bound inference; the theoretical FLOPs reduction ($2.0\times$ vs $3.0\times$) may manifest on more compute-constrained settings or with LoRA-TTT (Section 5.2).

Threshold Selection. The current threshold τ is set based on target update rate. Adaptive threshold learning via contextual bandits or per-domain calibration could improve decision quality.

Statistical Variance. All results are from single-run evaluations. Future work should include multiple random seeds to report variance estimates.

Non-Monotonic SKIP Scaling. The SKIP baseline shows non-monotonic scaling: Large (774M) achieves higher loss (2.005) than Medium (355M, 1.909). This reflects TTT layer initialization quality rather than backbone capability—dense UPDATE_1 shows proper monotonic improvement (Large: 1.484 < Medium: 1.525). Each scale's TTT layer is trained independently, leading to variation in initialization quality.

Statistical Independence. Our chunk-level statistical tests ($n \approx 16$K) assume independence between chunks. In practice, chunks from the same file may be correlated, potentially reducing effective sample size. While McNemar's test on decision agreement shows significant differences between methods (independence assumption; treated as suggestive), future work should employ cluster-robust standard errors or block bootstrap for rigorous inference.

5.2 Future Work

We identify several directions for future research:

- **Scaling to State-of-the-Art Architectures:** Extend experiments to modern LLMs such as Gemma 3 (1B, 4B, 12B, 27B) [4] to validate effectiveness on architectures with 128K context windows and enhanced reasoning capabilities.

- **Efficiency via LoRA-TTT:** Replace full TTT updates with Low-Rank Adaptation (LoRA) [6] to reduce per-update cost and achieve practical wall-clock speedups.
- **Multi-Signal Gating:** Combine TTT improvement with prediction entropy, attention dispersion, and budget-awareness for improved gating decisions (see Appendix E for preliminary results on TTT improvement as a standalone signal).
- **Diverse Evaluation Benchmarks:** Evaluate on reasoning benchmarks (MATH500, GSM8K), code generation (LiveCodeBench), and science QA (GPQA-Diamond) to assess generalization beyond perplexity.
- **Contextual Bandits for Threshold Learning:** Learn optimal per-context thresholds via online learning to improve upon fixed threshold gating.

6 Conclusion

We presented *PonderTTT*, a framework for adaptive compute allocation via Test-Time Training. We investigated the TTT layer's **Full-Sequence Reconstruction Loss** as an inference-compatible gating signal. Our experiments demonstrate that Reconstruction Gating achieves **82–89% Oracle Recovery** while being fully **training-free**—requiring no learned classifier or auxiliary networks. Combined with **determinism** (same input $\rightarrow$ same decision), **explainability** (decisions traceable to reconstruction loss), and significant improvements over Random Skip (up to 16% lower loss on OOD languages), our method provides a practical, principled approach to adaptive TTT.

Acknowledgements

We thank `HyperBlaze` for valuable assistance with JAX implementation and `iceman110` for providing computing resources.

References

[1] Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder, 2021. URL https://arxiv.org/abs/2107.05407.

[2] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

[3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1126–1135. JMLR.org, 2017.

[4] Gemma Team. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

[5] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019. URL https://openai.com/research/better-language-models.

[6] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

[7] Anton Lozhkov, Raymond Li, Loubna Ben Allal, et al. Starcoder 2 and the stack v2: The next generation, 2024. URL https://arxiv.org/abs/2402.19173.

[8] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

[9] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *ICML*, 2020.

[10] 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. Learning to (learn at test time): RNNs with expressive hidden states. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=wXfuOj9C7L.

Table 6: Hyperparameters for PonderTTT training.

Parameter	Value
Base Model	GPT-2 (Small/124M, Medium/355M, Large/774M, XL/1.5B)
Sequence Length	1024 tokens
Chunk Size	512 tokens
Batch Size	16 sequences
Training Chunks	160,000 (10K gradient steps)
TTT Inner Loop LR	1.0 (base, position-scaled)
Gradient Clipping	1.0
Gating Strategy	Threshold-based ($\mathcal{L}_{\text{rec}} > \tau$)
Target Update Rate	0.5 (50%)
TTT Loss Weight (β)	0.1
Hardware	NVIDIA RTX PRO 6000 (48GB)
Test Set	10K samples (held-out from The Stack v2)

Table 7: Complete OOD evaluation results (Loss). Model trained on Python only, 50% update budget.

Scale	Language	SKIP (Baseline)	Ours	Oracle
Small (124M)	JavaScript	3.164	**2.192**	1.985
	Java	3.148	**2.136**	1.981
	Go	6.130	**4.012**	3.623
Medium (355M)	JavaScript	2.458	**1.940**	1.682
	Java	2.420	**1.790**	1.633
	Go	3.902	**2.860**	2.617
Large (774M)	JavaScript	3.169	**2.003**	1.604
	Java	3.570	**2.173**	1.675
	Go	7.077	**4.362**	3.848
XL (1.5B)	JavaScript	2.852	**1.840**	1.574
	Java	3.213	**1.948**	1.643
	Go	6.520	**4.155**	3.703

A Experimental Details

A.1 Training Configuration

A.2 Baseline Training

The `UPDATE_1` baseline was trained with the same data and iterations. The TTT layer parameters are updated on every chunk with 1 gradient step. (Multi-step variants UPDATE_2/4 were explored during training ablation but are not evaluated on the test set.)

B Full Experimental Results

B.1 Out-of-Distribution Results (Full)

B.2 Random Skip vs Reconstruction Gating

Table 8 compares Random Skip (50%) with our Reconstruction Gating across model scales on Python (in-distribution).

B.3 Dense TTT (UPDATE_1) Full Results

Table 9 provides complete UPDATE_1 results across all scales and languages for reproducibility. UPDATE_1 applies TTT updates on every chunk, achieving the lowest loss at $3.0\times$ FLOPs cost.

Table 8: Random Skip vs Reconstruction Gating (Python, 50% update rate). Reconstruction Gating consistently outperforms Random Skip across all model scales.

Model	Random Skip	Ours	Oracle
Small (124M)	2.017	**1.977**	1.935
Medium (355M)	1.714	**1.697**	1.653
Large (774M)	1.698	**1.656**	1.580
XL (1.5B)	1.625	**1.576**	1.518

Table 9: Dense TTT (UPDATE_1) comparison across all settings. UPDATE_1 applies TTT updates on every chunk ($3.0\times$ TTT-layer FLOPs).

Scale	Language	SKIP	UPDATE_1	Ours	Oracle
Small (124M)	Python	2.324	1.716	**1.977**	1.935
	JavaScript	3.164	1.829	**2.192**	1.985
	Java	3.148	1.780	**2.136**	1.981
	Go	6.130	2.367	**4.012**	3.623
Medium (355M)	Python	1.909	1.525	**1.697**	1.653
	JavaScript	2.458	1.597	**1.940**	1.682
	Java	2.420	1.506	**1.790**	1.633
	Go	3.902	1.981	**2.860**	2.617
Large (774M)	Python	2.005	1.403	**1.656**	1.580
	JavaScript	3.169	1.458	**2.003**	1.604
	Java	3.570	1.417	**2.173**	1.675
	Go	7.077	1.999	**4.362**	3.848
XL (1.5B)	Python	1.875	1.384	**1.576**	1.518
	JavaScript	2.852	1.459	**1.840**	1.574
	Java	3.213	1.422	**1.948**	1.643
	Go	6.520	2.081	**4.155**	3.703

C Verification of No Data Leakage

We rigorously verified that our implementation contains no data leakage through both code analysis and empirical testing.

C.1 Code-Level Verification

1. **Causal Masking in TTT:** The TTT layer uses `jnp.tril()` (lower triangular matrix) for attention computation, ensuring position t only sees positions $0, \ldots, t$. This is identical to standard causal Transformer attention.
2. **Self-Supervised Target:** The TTT reconstruction loss uses $K \to (V - K)$ prediction with residual connection, reconstructing the target $(V - K)$ from Key. Both K and V are derived from the *current* token's hidden state. No next-token labels are used in the TTT update.
3. **Loss Computation:** The language modeling loss uses standard causal formulation: `logits[:, :-1]` predicts `labels[:, 1:]`, matching standard practice.

C.2 Gating Timeline and Inference Compatibility

We clarify how the gating decision is made and what it affects:

1. **Signal Computation:** The reconstruction loss $\mathcal{L}_{\text{rec}}$ used for gating is computed from the *current* chunk's self-supervised task (reconstructing $V - K$ from K), which requires no ground-truth labels. Specifically, we use the loss from the last mini-batch position, aggregated across heads.
2. **Decision Timing:** In teacher-forcing evaluation (used throughout this paper), the gating decision is made after observing the full chunk. For streaming autoregressive inference, the decision would be made after processing each chunk, affecting predictions for *subsequent* tokens only.

Table 10: Shuffled Input Sanity Check (Python held-out test set). PonderTTT provides substantial improvement on normal text but fails to improve on shuffled text, confirming TTT relies on legitimate sequential dependencies.

Input Type	SKIP Loss	Ours Loss	Improv.
Small (124M) (Normal)	2.324	1.977	14.9%
Small (124M) (Shuffled)	6.382	6.329	0.8% (Fail)

Table 11: Causal Mask Diagonal Ablation (Python held-out test set). Excluding the diagonal (k=-1) yields identical performance, suggesting no leakage from the diagonal.

Scale	Method	Loss (k=0)	Loss (k=-1)
Small (124M)	SKIP	2.324	2.324
	PonderTTT	1.977	1.978
Medium (355M)	SKIP	1.909	1.909
	PonderTTT	1.697	1.698

3. **UPDATE Procedure:** When UPDATE is triggered, we perform one gradient step on the TTT layer's fast weights, then re-forward the current chunk with the adapted weights to compute next-token predictions. This re-forward is included in the $2\times$ FLOPs budget.

Note on Perplexity Evaluation. All results in this paper use teacher-forcing (standard PPL measurement). Autoregressive generation experiments are left for future work.

C.3 Empirical Verification: Shuffled Input Test

To definitively rule out data leakage, we evaluate PonderTTT on *shuffled* input where tokens within each sequence are randomly permuted. If TTT were exploiting leaked information (e.g., via future token access), it would likely still show improvement or maintain low perplexity on shuffled text. If TTT legitimately learns sequential patterns, it should fail to improve on random sequences where no such patterns exist.

Result: On normal text, *PonderTTT* achieves 14.9% loss reduction (2.324 $\rightarrow$ 1.977). On shuffled text, the baseline loss explodes to 6.38 (PPL $\approx$ 590), and TTT fails to provide meaningful improvement (loss 6.33, only 0.8% reduction), confirming that the mechanism relies on valid sequential structure and does not exploit leakage.

C.4 OOD Generalization as Evidence

The strong transfer to unseen languages (Table 7) provides additional evidence against overfitting: if the model had memorized training data, it would not generalize to Go or Java where substantial loss reductions are observed across all model scales.

C.5 Causal Mask Diagonal Ablation

A potential concern is whether including the diagonal in the causal mask (`jnp.tril(k=0)`) allows position t to use its own gradient, constituting "concurrent update" leakage. We compare two settings:

- **k=0 (standard):** Position t uses gradients from positions $0, \ldots, t$ (includes diagonal)
- **k=-1 (strict causal):** Position t uses gradients from positions $0, \ldots, t-1$ (excludes diagonal)

Result: As shown in Table 11, the difference between k=0 and k=-1 is negligible. This suggests that the diagonal does *not* provide an unfair advantage—the model's improvement appears to come from learning sequential patterns.

D Computational Cost Model

We define computational cost in terms of forward-pass equivalents:

- **SKIP (0 updates):** $1\times$ — base forward pass only

Table 12: Correlation between TTT improvement and oracle advantage (GPT-2 Small/124M).

Metric	Value
Spearman ρ	0.825
Pearson r	0.791
Top-50% Overlap with Oracle	82.5%

- **UPDATE_N:** $(1 + 2N)\times$ — 1 initial forward + N×(backward + re-forward). We approximate backward ≈ forward cost; re-forward uses updated weights.[3]
- **PonderTTT (Binary):** $(1 + 2 \times \text{update\_rate})\times$ — where update_rate is the fraction of chunks receiving TTT updates. With 50% target update rate, this yields $1 + 2 \times 0.5 = 2.0\times$ theoretical cost.

Theoretical vs Observed Cost. While the theoretical cost model predicts 3× FLOPs for `UPDATE_1`, our wall-clock latency measurements on modern GPUs show minimal difference between SKIP and UPDATE (see Section 4.4). This occurs because small-batch inference is memory-bound: GPU utilization ranges from 32–84%, and the additional compute fits within available headroom.

Binary vs Continuous Gating. Unlike continuous gating (which scales the learning rate but still requires backward passes), binary gating enables true computational savings by completely skipping the backward pass for SKIP decisions. With our threshold-based approach at 50% update rate, we achieve a balance between cost savings and performance.

E Training-Free Gating via TTT Internal Signals (GPT-2 Small/124M)

While our main method uses the raw reconstruction loss $\mathcal{L}_{\text{rec}}$ for its simplicity and inference compatibility, we also investigated *training-free* gating using TTT's internal loss *improvement* ($\Delta\mathcal{L}$) as an alternative signal.

E.1 Motivation

Using the raw reconstruction loss as a gating signal is simple, but one might ask: does the *improvement* in loss after an update correlate better with actual benefit? We investigate this alternative:

- **Raw loss**: $\mathcal{L}_{\text{rec}}$ before any gradient step
- **Loss improvement**: $\Delta\mathcal{L} = \mathcal{L}_{\text{rec}}^{(0)} - \mathcal{L}_{\text{rec}}^{(1)}$ (reduction after one step)

Note: The analysis in this appendix is conducted on **GPT-2 Small (124M)**. We include this analysis to compare the efficacy of raw reconstruction loss versus loss improvement ($\Delta\mathcal{L}$) as gating signals.

We propose: instead of using raw loss, directly *measure* the improvement to better identify high-benefit chunks.

Important Clarification: Computing $\Delta\mathcal{L}$ requires performing one UPDATE step first (forward + backward + re-forward). Therefore, $\Delta\mathcal{L}$ *cannot* be used for binary SKIP/UPDATE pre-decisions—it is only available *after* committing to UPDATE. This appendix explores $\Delta\mathcal{L}$ as a diagnostic/analysis tool; compute-saving binary gating uses only raw loss ($\mathcal{L}_{\text{rec}}$) as described in Section 3.

E.2 TTT Improvement as Gating Signal

The TTT layer's internal self-supervision loss measures "how much the model wants to learn" from the current context. We define:

$$\texttt{ttt\_improvement} = \mathcal{L}_{\text{rec}}^{(0)} - \mathcal{L}_{\text{rec}}^{(1)} \tag{1}$$

where $\mathcal{L}_{\text{rec}}^{(0)}$ is the reconstruction loss before the first gradient step and $\mathcal{L}_{\text{rec}}^{(1)}$ is after one step. Higher improvement indicates the chunk benefits more from adaptation.

E.3 Correlation Analysis

We measure correlation between `ttt_improvement` and oracle advantage on 2,000 individual samples (1,000 batches, batch_size=1):

[3]Traditional estimates assume backward ≈ 2× forward; our TTT layer's small footprint relative to the frozen backbone reduces this ratio. Actual latency overhead is measured in Section 4.4.

E.4 End-to-End Comparison

We compare three gating strategies at 50% target update rate:

Table 13: Gating method comparison (GPT-2 Small/124M, 50% update rate, 2K subset). **Oracle Capture** is the percentage of Oracle's gain over Random that is recovered by the method. Note: $\Delta\mathcal{L}$-based selection requires computing the update for all candidates; this overhead is not included in the reported FLOPs.

Method	Loss	Rel. FLOPs	vs Random	Oracle Capture
Oracle (greedy)	1.950	2.0×	+17.1%	100%
TTT Improvement (top-k)	1.988	2.0×	+15.5%	90.5%
Random Skip	1.995	2.0×	baseline	0%

Key Findings:

1. **Training-free gating works**: TTT improvement captures 90.5% of Oracle's improvement over random, without any learned components.
2. **Efficient alternative to always-UPDATE**: TTT Improvement gating (loss=1.988, cost=2.0×) trades 0.30 nats vs UPDATE_1 (loss=1.690, cost=3.0×) for 33% compute reduction.

E.5 Threshold-Based Online Gating

For streaming inference, we implement per-chunk threshold gating:

$$\texttt{decision} = \mathbb{1}[\texttt{ttt\_improvement} > \tau] \tag{2}$$

where $\tau \approx 0.034$ (median TTT improvement) for 50% update rate.

Table 14: Online gating comparison. Top-k requires lookahead; threshold does not. **Decision Acc.** measures agreement with Oracle's binary decisions.

Method	Online	Decision Acc.	Oracle Capture
Top-k selection	✗	82%	90.5%
Fixed threshold	✓	80%	89.5%

Conclusion: TTT's internal self-supervision loss provides an effective training-free gating signal. *Note:* While loss *improvement* ($\Delta\mathcal{L}$) also works, raw loss ($\mathcal{L}_{\text{rec}}$) is simpler (no additional forward pass) and performs comparably, so we use it for the main experiments.