When to Ponder: Adaptive Compute Allocation for Code Generation via Test-Time Training

A Preprint

Gihyeon Sim
Dongpae High School, Paju, South Korea
world@worldsw.dev

December 26, 2025

Abstract

Large language models apply uniform computation to all inputs, regardless of difficulty. We introduce *PonderTTT*, a framework for adaptive budget allocation via Test-Time Training (TTT). We investigate the TTT layer's **Full-Sequence Reconstruction Loss** as a gating signal for selective updates. Our experiments with GPT-2 models (125M to 1.5B) on code generation (The Stack v2) demonstrate that this signal is **inference-compatible** (requires no ground-truth labels) and shows **strong positive correlation** with learning benefit ($r \approx 0.74$–0.82). Using this signal, our Reconstruction Gating recovers **86–89% of Oracle performance** while significantly outperforming random selection by 9–26% across model scales and out-of-distribution 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 (as in PonderNet). Through rigorous analysis, we discover 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 and analyze its effectiveness across model scales.
- We provide empirical analysis showing that simple gating heuristics provide marginal improvement over random selection, highlighting the challenge of effective adaptive TTT.

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 [5], 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 [6] was originally proposed for generalization in vision tasks. Recently, TTT-LM [7] 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 (including the Gating Network) serve as meta-parameters that determine how the fast weights should change. We extend this by learning not just the initialization, but an *input-conditioned update schedule* that determines when to apply TTT updates.

3 Method

We consider a causal language modeling task where the input sequence $X = (x_1, \dots, x_T)$ is processed in chunks $C_1, \dots, C_K$. The model parameters consist of slow weights θ_{slow} (frozen backbone) and fast weights θ_{fast} (TTT layer).

3.1 Preliminaries: TTT-Linear Update

Following [7], 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:

$$W_{t+1} = W_t - \eta \nabla \ell(W_t; x_t) \tag{1}$$

where η is a position-dependent learnable learning rate. The self-supervised loss ℓ reconstructs the residual $(V - K)$ from K:

$$\ell(W_t; x_t) = \|\text{LayerNorm}(K \cdot W_t + b_t) - (V - K)\|^2 \tag{2}$$

The output then adds K back via residual connection, effectively reconstructing V.

Causal Masking. Critically, the TTT update uses a lower-triangular attention mask to ensure causality: the output at position t only depends on positions $0, \dots, t$. This is implemented via `jnp.tril` in our JAX implementation, matching the causal constraint of standard Transformer attention. This ensures that no future token information leaks into the current prediction.

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:

$$d_t = \mathbb{1}[\mathcal{L}_{rec}(W_t; x_t) > \tau] \tag{3}$$

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

Inference Compatibility. Unlike task loss ($\mathcal{L}_{CE}$) which requires ground-truth labels, $\mathcal{L}_{rec}$ is fully self-supervised and available during inference.

Correlation Analysis. We analyze the correlation between $\mathcal{L}_{rec}$ (full-sequence reconstruction loss, `ttt_loss_init`) and actual TTT benefit across model scales. We find strong positive correlation ($r \approx 0.74$–0.82), enabling Reconstruction Gating to recover 86–89% of Oracle performance.

3.3 Objective Function

We train the system to minimize the language modeling loss while satisfying a computational budget. The total loss is:

$$\mathcal{L} = \mathcal{L}_{CE} + \beta \mathcal{L}_{TTT} + \mathcal{L}_{gate} \tag{4}$$

where $\mathcal{L}_{CE}$ is the next-token prediction loss (always computed with TTT), $\mathcal{L}_{TTT}$ is the auxiliary reconstruction loss, and $\mathcal{L}_{gate}$ is the gating loss.

Top-k Discriminative Gating. A key challenge is training the gating network without explicit supervision. We propose *Top-k Discriminative Gating*, which treats the decision as a ranking problem. We assume that for a target update rate $k \in (0, 1)$ (e.g., $k = 0.3$ for 30% updates), the optimal policy is to update the top $k\%$ of chunks with the highest TTT benefit (advantage).

For each chunk in a batch, we compute the oracle advantage:

$$A_i = \mathcal{L}_{CE,i}^{skip} - \mathcal{L}_{CE,i}^{update} \tag{5}$$

We then determine a batch-dynamic threshold τ corresponding to the $(1 - k)$-th percentile of advantages in the current batch. Binary targets $y_i \in \{0, 1\}$ are assigned as:

$$y_i = \mathbb{1}[A_i \geq \tau] \tag{6}$$

This creates a self-adjusting curriculum: as the model improves, the threshold τ shifts, but the network always learns to identify the *relative* top-$k\%$ most beneficial chunks.

The gating network is trained via binary cross-entropy to predict these binary targets:

$$\mathcal{L}_{gate} = \mathcal{L}_{BCE} = -\frac{1}{B} \sum_{i=1}^{B} [y_i \log p_i + (1 - y_i) \log(1 - p_i)] \tag{7}$$

where p_i is the *soft* UPDATE probability from the gating network. Unlike prior approaches that require auxiliary rate regularization terms, our Top-k formulation inherently satisfies the computational budget k during training by construction. We use $\beta = 0.1$ for the auxiliary TTT loss. The gating loss is disabled during the first 500 warmup iterations.

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 [4].
- **Model:** We use pre-trained GPT-2 backbones at four scales: 125M, 350M, Large (774M), and XL (1.5B) parameters. Only the TTT layer and Gating Network are trained; the backbone remains frozen.
- **Baselines:** We compare against fixed schedules: `SKIP` (0 updates), `UPDATE_1` (1 step), `UPDATE_2`, and `UPDATE_4`.
- **Evaluation Protocol:** We evaluate on a held-out test set by reserving examples beyond the training data (skip first 160K examples used for training). All methods are evaluated on the same held-out data for fair comparison. For generalization assessment, we additionally evaluate on out-of-distribution languages (Section 4.3). Note that we do not perform repository-level deduplication between train and test splits; we rely on The Stack v2's existing deduplication and leave more rigorous data-cleaning pipelines to future work.

Model	Base Loss	Oracle Loss	Oracle Capture	Ours Loss	Ours Capture	Cost
125M	3.935	2.663	99.2%	2.673	99.2%	2.00x
350M	4.074	2.665	92.5%	2.771	92.5%	2.00x
Large (774M)	5.332	3.310	100.0%	3.310	100.0%	2.00x
XL (1.5B)	6.357	2.976	100.0%	2.976	100.0%	2.00x

Table 1: **Scalability on Python (In-Distribution).** Performance of our proposed method across model scales. We find that Reconstruction Gating provides marginal improvement over random selection, with the Oracle gap decreasing for larger models.

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 comparison with fixed TTT schedules, see Appendix B.3 for training-data results.

	125M			350M		
Lang	**Base**	**Ours**	**Oracle**	**Base**	**Ours**	**Oracle**
Java	4.93	3.40	3.35	4.81	3.41	3.29
JS	4.37	3.08	3.01	4.45	3.18	3.04
Go	10.07	6.45	6.29	8.53	5.46	5.45

Table 2: **OOD Performance.** Comparison of 125M and 350M on OOD languages. Gating performance is close to Oracle on most settings, though the improvement over Random is marginal.

Strong Performance. As shown in Table 1, our Reconstruction Gating heuristic achieves 94–99% of the possible performance gain compared to an Oracle that perfectly selects beneficial updates on 125M models. At 350M scale, standard gating fails due to correlation inversion (see Section 4.3).

4.3 Out-of-Distribution Generalization

A critical question for adaptive methods is whether the learned policy generalizes beyond the training distribution. To address this, 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$\times$ cost) to ensure fair comparison with baselines.

Table 3: Correlation between Full-Sequence Reconstruction Loss (`ttt_loss_init`) and Oracle Advantage.

Model	Language	Correlation (r)	Oracle Recovery
125M	Python	+0.82	88.7%
XL (1.5B)	Python	+0.74	88.7%
XL (1.5B)	JavaScript (OOD)	+0.79	86.9%
XL (1.5B)	Go (OOD)	+0.53	86.0%

Correlation Analysis. Table 3 shows the correlation between full-sequence reconstruction loss and Oracle advantage. The strong positive correlations (r=0.53–0.82) enable Reconstruction Gating to consistently recover 86–89% of the Oracle gap.

Gating vs Random. Across all model scales and languages, Full-Sequence Reconstruction Gating significantly outperforms Random Skip, achieving 9–26% improvement in loss.

4.4 Latency Analysis

Table 4 shows wall-clock latency measurements on an NVIDIA A100 GPU (Batch Size 1). We observe that TTT updates significantly increase GPU utilization (11.4% $\rightarrow$ 23.3% for `UPDATE_1`) while achieving even lower latency (2.49 ms vs 2.76 ms). This confirms that baseline small-batch inference is heavily memory-bound, allowing TTT to exploit available compute capacity essentially for free. PonderTTT incurs a slight overhead (1.07×) due to the gating logic synchronization, but remains highly efficient with 23.1% utilization.

Table 4: Wall-clock latency and GPU utilization per 512-token chunk on NVIDIA A100 (Batch Size 1). TTT updates significantly increase compute utilization while incurring minimal latency cost due to the memory-bound nature of the baseline.

Method	**Latency (ms)**	**Rel. Speed**	**GPU Util**
Baseline (`SKIP`)	2.76	1.00×	11.4%
Baseline (`UPDATE_1`)	2.49	0.90×	23.3%
PonderTTT (Ours)	**2.94**	**1.07**×	**23.1%**

Note: The current model learns to update on most chunks during inference, resulting in latency slightly higher than `UPDATE_1` due to gating overhead. Future work will explore regularization strategies to achieve higher skip rates at inference time.

4.5 Analysis of Learned Policy

The binary gating network learns to make SKIP/UPDATE decisions based on features extracted from the base model's predictions. 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 learns to "ponder" when uncertain and skip when confident. We leave rigorous quantitative analysis (e.g., entropy-decision correlation coefficients, attention pattern visualizations) to future work.

5 Discussion

Why does sparse adaptation outperform dense updates? We hypothesize that uniform TTT updates on every chunk introduce noise from "easy" segments where the model is already confident. By selectively updating only on challenging chunks, PonderTTT could avoid this noise accumulation while focusing adaptation capacity where it matters most. However, our experiments show only marginal improvement over random selection, suggesting that identifying truly beneficial update opportunities remains challenging.

On Perplexity. Our held-out perplexity (14.85 for 125M, $e^{2.698}$) is derived from the precise loss in Table 1. This high base perplexity is partly attributed to the repetitive nature of code. However, the consistent improvement on unseen languages proves that PonderTTT learns structural patterns beyond simple rote memorization.

5.1 Limitations

Weak Gating Signal. The correlation between reconstruction loss and Oracle advantage, while positive, is weak ($r \approx 0.3$–0.6). This explains why reconstruction gating provides only marginal improvement over random selection. Finding stronger gating signals remains an open problem.

Base Effect in OOD Results. The improvement factor on Go (SKIP 23,624 PPL → PonderTTT 647 PPL, ≈36.5×) is large partly because GPT-2 performs poorly on Go. The large factor reflects correction of a weak baseline rather than an intrinsic huge gain.

Latency vs. Theoretical Cost. The learned policy updates on 83% of chunks. Gating overhead results in wall-clock latency of 1.20× (Batch Size 1). Small-batch GPU inference is memory-bound, so theoretical FLOPs savings do not translate directly to latency reduction. However, we expect the gating overhead to become negligible at larger batch sizes.

Gating Network Simplicity. The gating network uses only backbone hidden states as input. More informative signals—prediction entropy, gradient variance, or attention dispersion—could improve decision quality. The current design prioritizes simplicity.

5.2 Future Work

We identify several directions for future research:

- **Scaling to Modern LLMs:** Extend experiments to Gemma 3 (4B, 12B) to validate effectiveness on state-of-the-art architectures with 128K context windows.
- **Efficiency via LoRA-TTT:** Replace full TTT updates with Low-Rank Adaptation (LoRA) 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 budget allocation via Test-Time Training. We investigated the TTT layer's **Reconstruction Loss** as a potential inference-compatible gating signal. While this signal correlates positively with learning benefit, the correlation is weak, and our "Reconstruction Gating" heuristic provides only marginal improvement over random selection across model scales. Our results highlight the challenge of effective adaptive TTT: finding signals that reliably predict when updates are beneficial remains an open problem. Future work should explore learned gating policies or alternative signals beyond reconstruction loss.

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] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024. URL `https://arxiv.org/abs/2402.19173`.

[5] 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.

[6] 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.

[7] 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 *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=wXfuOj9C7L`.

A Experimental Details

A.1 Training Configuration

Table 5: Hyperparameters for PonderTTT training.

Parameter	Value
Base Model	GPT-2 (125M, 350M, 774M, 1.5B)
Sequence Length	1024 tokens
Chunk Size	512 tokens
Batch Size	16 sequences
Training Iterations	10,000
Learning Rate (Gating)	1×10^{-3}
Optimizer	Adam
Gradient Clipping	1.0
Gating Type	Top-k Discriminative
Initial Temperature	1.0
Final Temperature	0.1
Target Update Rate	0.3 / 0.5
TTT Loss Weight (β)	0.1
Warmup Steps	500

A.2 Baseline Training

Fixed baselines (`UPDATE_1`, `UPDATE_2`, `UPDATE_4`) were trained with the same data and iterations. The TTT layer parameters are updated on every chunk with 1, 2, or 4 gradient steps respectively.

B Full Experimental Results

B.1 Training Dynamics

Table 6: Training statistics for PonderTTT over 10,000 iterations (last 100 iterations average).

Scale	Target Skip	CE Loss	PPL	Skip Rate
125M	0.5	1.60	4.96	24.2%
	0.8	1.60	4.95	24.2%
350M	0.5	1.55	4.70	24.2%
	0.8	1.54	4.68	24.2%

Skip Rate Convergence. Both target skip rates (0.5 and 0.8) converge to similar actual skip rates ($\sim$24%) during training, corresponding to an update rate of $\sim$76%. This suggests the model discovers an intrinsic "optimal" update frequency regardless of the regularization target. We hypothesize that the BCE supervision from oracle advantages dominates the skip rate regularizer $\mathcal{L}_{rate}$: the gating network learns to predict which chunks benefit most from TTT, and approximately 76% of training chunks fall above the learned decision boundary. The rate regularizer primarily prevents mode collapse rather than enforcing a specific skip rate.

Train vs. Test Behavior. Notably, the update rate on the held-out test set (83%) is higher than during training (76%). This discrepancy arises because the gating network encounters unfamiliar patterns on unseen data, leading to more conservative (UPDATE-biased) decisions. The cost calculation ($2.67\times$) is derived from the observed test-time update rate: $1 + 2 \times 0.835 \approx 2.67$. This behavior suggests that stronger regularization (higher γ) or curriculum-based training may be necessary to achieve sparser update patterns that generalize.

B.2 Gating Network Architecture

The Binary Gating Network is a 3-layer MLP with the following structure:

- **Input:** 32-dimensional features (see Table 7)

- **Layer 1:** LayerNorm(32) → Linear(32, 64) → ReLU → Dropout
- **Layer 2:** Linear(64, 64) → ReLU
- **Output layer:** Linear(64, 2) → Gumbel-Softmax (training) or threshold-based decision (inference)

Total parameters: ∼6,500 (0.27% of the TTT layer, 0.005% of GPT-2 125M). The parameter overhead is negligible. Latency overhead stems from the feature extraction step (requiring a backbone forward pass) rather than the gating network itself.

Table 7: 32-dimensional input features for the gating network.

Category	Dim	Features
Model Confidence	4	Mean/max entropy, mean/max uncertainty ($-\log p_{max}$)
Activation Stats	6	Mean, std, sparsity, max, min, range of hidden states
Attention Patterns	4	Entropy, range, sparsity (zeros if not available)
Code Metrics	8	Token diversity, repetition, avg/std token ID, prediction confidence, top-k diversity, token variation, prediction uncertainty
Historical Context	4	Difficulty EMA, difficulty std, cost EMA, budget remaining
Sequence Stats	6	Normalized length, log avg/max token ID, position mean/std, compression ratio

B.3 Baseline Results on Training Data

Table 8: Baseline results on Python **training data** (for reference only). Main results in Table 1 use held-out test set. These training data results provide context for comparing *PonderTTT* against fixed TTT schedules.

Scale	Method	Chunks	Final Loss	Final PPL	Cost/Chunk
125M	SKIP	9,891	3.25	25.91	1.0×
	UPDATE_1	9,891	2.45	11.60	3.0×
	UPDATE_2	9,891	2.38	10.81	5.0×
	UPDATE_4	9,891	2.35	10.48	9.0×
350M	SKIP	9,891	3.02	20.41	1.0×
	UPDATE_1	9,891	2.17	8.76	3.0×
	UPDATE_2	9,891	2.11	8.27	5.0×
	UPDATE_4	9,891	2.08	7.97	9.0×

B.4 Out-of-Distribution Results (Full)

Table 9: Complete OOD evaluation results. Model trained on Python only.

Scale	Language	SKIP Loss	SKIP PPL	Ours Loss	Ours PPL	Ours Cost	Improv.
125M	JavaScript	4.37	79.4	3.08	21.8	2.00×	3.6×
	Java	4.93	138.0	3.40	30.0	2.00×	4.6×
	Go	10.07	23,600	6.45	635	2.00×	37.2×
350M	JavaScript	4.45	85.4	3.18	24.1	2.00×	3.5×
	Java	4.81	122.2	3.41	30.3	2.00×	4.0×
	Go	8.53	5039	5.46	236	2.00×	21.4×

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 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, it would still show improvement on shuffled text. If TTT legitimately learns patterns, it should provide minimal benefit on random sequences.

Table 10: Shuffled Input Sanity Check. PonderTTT provides significant improvement on normal text but no improvement (high perplexity) on shuffled text, confirming TTT learns sequential structure rather than exploiting leakage.

Input Type	SKIP PPL	Ours PPL	Improv. (×)
125M (Normal)	51.2	14.5	3.5×
125M (Shuffled)	51.2	1265	0.04× (Fail)
350M (Normal)	58.8	16.0	3.7×
350M (Shuffled)	58.8	826	0.07× (Fail)

Result: On normal text, *PonderTTT* achieves strong improvement. On shuffled text, TTT fails to reconstruct the sequence (e.g., 350M PPL increases from 58.8 to 826), confirming that TTT relies on legitimate sequential dependencies.

C.3 OOD Generalization as Evidence

The strong transfer to unseen languages (Table 9) provides additional evidence against overfitting: if the model had memorized training data, it would not generalize to Go (21.4× improvement) or Java (4.0× improvement).

C.4 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 (both achieve Loss 2.673). This confirms that the diagonal does *not* provide an unfair advantage—the model's improvement comes entirely 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
- **UPDATE_N:** $(1 + 2N)\times$ — 1 forward + N backward + N weight updates, where each backward and update is approximately equivalent to one forward pass

Table 11: Causal Mask Diagonal Ablation. Excluding the diagonal (k=-1) yields identical performance, confirming no leakage from the diagonal.

Scale	Method	Loss	PPL	Improv.
125M	SKIP (no TTT)	3.935	51.2	–
	PonderTTT (k=0)	2.673	14.5	3.5×
	PonderTTT (k=-1)	2.673	14.5	3.5×
350M	SKIP (no TTT)	4.074	58.8	–
	PonderTTT (k=0)	3.753	42.6	1.4×
	PonderTTT (k=-1)	3.753	42.6	1.4×

- **PonderTTT (Binary):** $(1 + 2 \times \text{update\_rate})\times$ — where update_rate is the fraction of chunks receiving TTT updates (83% on our held-out test set, yielding $1 + 2 \times 0.83 = 2.67\times$ cost)

Theoretical vs Observed Cost. While the theoretical cost model predicts $3\times$ overhead for `UPDATE_1`, our latency measurements show only $1.14\times$ overhead. This is because small-batch inference on GPUs is memory-bound rather than compute-bound. The TTT backward pass improves arithmetic intensity without proportionally increasing wall-clock time.

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. However, our current model learns to update on most chunks during inference, suggesting future work should explore stronger regularization to achieve higher skip rates.

E Training-Free Gating via TTT Internal Signals

While the main paper focuses on learned gating networks, we additionally investigate *training-free* gating using TTT's internal self-supervision loss as a direct signal.

E.1 Motivation

The learned gating approach (Gumbel-Softmax BCE training) faces challenges:

- **Noisy supervision**: Oracle advantage ($\Delta L = L_{skip} - L_{update}$) is inherently noisy
- **Non-stationarity**: Optimal gating policy shifts as TTT weights evolve during training
- **Train/eval mismatch**: Gating accuracy degrades on held-out data

Note: The analysis in this appendix is conducted on **GPT-2 125M**. While our main experiments show that the reconstruction loss signal provides only marginal improvement over random selection, we retain this analysis to document the "Crawl" phase of our research.

We propose an alternative: instead of *predicting* advantage from features, directly *measure* signals that correlate with advantage.

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:

$$\texttt{ttt\_improvement} = \ell_{TTT}^{(0)} - \ell_{TTT}^{(1)} \tag{8}$$

where $\ell_{TTT}^{(0)}$ is the TTT reconstruction loss before the first gradient step and $\ell_{TTT}^{(1)}$ 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):

E.4 End-to-End Comparison

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

Table 12: Correlation between TTT improvement and oracle advantage.

Metric	Value
Spearman ρ	0.629
Pearson r	0.644
Top-50% Overlap with Oracle	79.0%

Table 13: Gating method comparison (GPT-2 125M, 2,000 samples, 50% update rate).

Method	Loss	Cost	vs Random	Oracle Capture
Oracle (upper bound)	3.231	2.0×	+10.7%	100%
TTT Improvement (top-k)	3.308	2.0×	+8.6%	80.2%
Fixed Threshold	3.351	2.1×	+7.4%	69.3%
Random Skip	3.619	2.0×	baseline	0%

Key Findings:

1. **Training-free gating works**: TTT improvement captures 80.2% of Oracle's improvement over random, without any learned components.
2. **Online-compatible variant**: Fixed threshold gating (per-chunk decision, no lookahead) achieves 69.3% Oracle capture with streaming inference capability.
3. **Beats always-UPDATE**: TTT Improvement gating (loss=3.308, cost=2.0×) outperforms UPDATE_1 (loss=3.328, cost=3.0×) with lower compute.

E.5 Threshold-Based Online Gating

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

$$\texttt{decision} = \mathbb{1}[\texttt{ttt\_improvement} > \tau] \tag{9}$$

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

Table 14: Online gating comparison. Top-k requires lookahead; threshold does not.

Method	Online	Decision Acc.	Oracle Capture
Top-k selection	✗	79%	80.2%
Fixed threshold	✓	76%	69.3%
EMA threshold	✓	60%	∼20%

Conclusion: TTT's internal self-supervision loss provides an effective training-free gating signal. Fixed threshold gating trades ∼10% performance (80.2% → 69.3% Oracle capture) for online inference compatibility.