GAP-AWARE PREDICTIVE MEMORY FOR TEST TIME TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-Time Training (TTT) expands recurrent sequence modeling by treating hidden states or KV-Binding as input-adaptive fast weights that continuously learn during inference. Existing TTT frameworks typically optimize these weights using self-supervised next-token prediction or full-state reconstruction. We argue that fitting fast memory directly to raw future representations is fundamentally redundant: a substantial portion of the target state is already explained by the frozen model's pre-trained predictive prior. Under bounded memory budgets, this redundancy forces fast weights to expend limited capacity on static, task-invariant dynamics rather than novel, context-specific interactions. To address this limitation, we introduce **GAP-TTT** (**G**ap-**A**ware **P**redictive Memory), a TTT framework that explicitly decomposes sequence prediction into a pretrained slow prior and an adaptive fast correction. Instead of modeling full target representations, **GAP-TTT** trains fast memory exclusively on the predictive gap, the semantic residual unaccounted for by the frozen backbone. Specifically, a lightweight semantic predictor exposes the slow model's future latent expectations, and temporal consistency maps the resulting prediction error into both the memory write target and transition-aware key features. This unifies what to memorize and how to index it without requiring auxiliary inner-loop objectives. GAP-TTT updates these weights via online causal ridge regression, enabling hardware-efficient parallel prefix computation during inference, eliminating test-time backpropagation and iterative gradient descent. In the RULER and LongBench-v2 tests, GAP-TTT consistently outperforms other TTT baseline models while requiring only 25% of their training parameters, validating our argument that fast memory should focus on local contextual residuals.

1 INTRODUCTION

Large language models (LLMs) exhibit remarkable in-context reasoning capabilities, yet their underlying parameterization remains static during inference. To accommodate contextual dynamics, standard architectures process context through transient activation-based states: either retaining history within memory-intensive key-value (KV) caches Pope et al. (2023) or compressing it into fixed-capacity recurrent states Gu & Dao (2023). This paradigm exhibits severe limitations in extended contexts, non-stationary streams, and domain-shifted inputs, where effective reasoning demands dynamic structural adaptation rather than static retrieval over frozen representations. Test-Time Training (TTT) (Sun et al., 2020; Gandelsman et al., 2022; Liu et al., 2021; Ma, 2024; Ranzato et al., 2015; Yang et al., 2025b; Liang et al., 2025) addresses this fundamental bottleneck by reformulating sequence modeling as an online optimization problem: rather than passively storing tokens, the model continuously adapts a set of input-dependent **fast weights** on the incoming context via self-supervised inner-loop objectives. By compiling contextual history directly into parametric state updates at test time, TTT bridges the gap between static in-context conditioning and active parameter adaptation, offering a scalable path toward truly adaptive sequence models.

To operationalize this paradigm, recent advances in TTT have focused extensively on the architecture of fast weights, their adaptation objectives, and runtime efficiency. Foundational frameworks formulate recurrent states as inner-loop learners, such as linear models or shallow MLPs, trained online via self-supervised reconstruction (Sun et al., 2025a;b; Zuo et al., 2026). Subsequent efforts have sought to better align inner-loop objectives with autoregressive language modeling, incorporating next-token prediction signals, multi-step future representations, or nearest-neighbor context (Ouyang

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

et al., 2026; von Oswald et al., 2026; Hardt & Sun, 2024; Yuksekgonul et al., 2026). In parallel, a growing line of work addresses the computational burden of test-time optimization: architectures such as InPlaceTTT, S-TTT, and QTTT (Feng et al., 2026; Zhu et al., 2026; Bansal et al., 2025) minimize inner-loop memory and gradient overhead, while neural memory models like Titans (Behrouz et al., 2024) explore scalable test-time retention. Together, these developments demonstrate the broad potential of inference-time adaptation for extended-context modeling (Tandon et al., 2025; Zhu et al., 2026; Yuksekgonul et al., 2026; Sun et al., 2025a). Despite these empirical strides, existing research has focused disproportionately on how fast memory should be updated and scaled, leaving a more fundamental question unaddressed: **what information should fast memory actually store?** Prevailing methods uniformly train fast weights to reconstruct absolute future tokens or complete latent representations, typically through one-step token-level prediction. We argue that this formulation is fundamentally redundant: a pretrained language model already encapsulates powerful predictive priors that account for standard syntactic, semantic, and domain invariants. Forcing capacity-bounded fast weights to predict full representations needlessly expends memory on dynamics the frozen backbone already explains, diluting the capacity available for novel, context-specific interactions. Instead, fast memory should play a strictly complementary role. Rather than re-learning what is already predictable, test-time adaptation should model the **predictive gap**: the semantic residual between the frozen model's prior expectation and the observed contextual reality.

To realize this principle, we introduce **GAP-TTT** (**G**ap-**A**ware **P**redictive Memory), a test-time training framework that re-defines the objective of fast weights. GAP-TTT formalizes sequence processing as an explicit dual-memory system: the frozen base LLM operates as a slow predictive prior, while a lightweight associative memory dynamically captures only the residual information that escapes this prior. Specifically, rather than predicting a single future token or representation, GAP exposes the slow model's prospective state within an output-aligned semantic latent space and compares it against a multi-token future semantic trajectory. By enforcing temporal consistency over this trajectory, GAP-TTT jointly derives both the predictive-gap target (what to write) and its corresponding transition-aware key features (how to index). The resulting context-specific memory is compiled via online causal ridge regression and subsequently queried to inject localized semantic corrections into the frozen model's forward stream. By storing predictive residuals rather than monolithic sequence representations, GAP-TTT eliminates representational redundancy, shielding fast-weight capacity from premature saturation under strict memory budgets. Crucially, formulating memory synthesis as causal ridge regression resolves the primary computational bottlenecks of prior TTT architectures: it yields an exact, closed-form solution that requires neither iterative inner-loop gradient descent nor heuristic auxiliary losses, and eliminates the need for a fixed chunksize to hard-control the granularity of memory updates. Furthermore, this mathematical formulation naturally exhibits a dual execution profile: it admits parallel prefix computation via hardware-efficient associative scans during prompt prefilling, while seamlessly transitioning into recursive streaming updates. Extensive experiments on RULER and LongBench-v2 demonstrate the effectiveness of GAP-TTT across different backbone models and context lengths. When training only lightweight TTT-adapter parameters, GAP-TTT improves the average RULER performance of LLaMA3.1-8B from 86.94 to 87.63 with only 25.24M additional trainable parameters, substantially reducing the training overhead compared with full-model continued pre-training. Further ablations validate the predictive-gap target, transition-aware memory writing, and selective correction behavior of GAP-TTT, supporting its effectiveness as a parameter-efficient framework for test-time adaptation.

In summary, our core contributions are:

- **Predictive Residual Memory for TTT:** We introduce GAP-TTT, which models sequence-specific predictive residuals over a multi-horizon semantic trajectory rather than reconstructing complete future representations. Temporal consistency jointly derives the predictive-gap target and its transition-aware write feature.

- **Closed-Form Causal Memory Construction:** We derive a causal ridge regression formulation that eliminates test-time backpropagation and a fixed chunksize, enabling parallel prefix scans during prefill and context-length-independent recursive updates during decoding.

- **Efficient Long-Context Adaptation:** Across RULER and LongBench-v2, GAP-TTT consistently outperforms competitive TTT baselines while requiring only 25.2M trainable adapter parameters on LLaMA-3.1-8B.

2 Related Work

Test-Time Training for Language Models Test-time training (TTT) adapts pretrained models on unlabeled test inputs using self-supervised signals, and was initially developed to improve robustness under distribution shifts Sun et al. (2020). In sequence modeling, fast weights act as sequence-specific states that evolve with the current context. Learning to (Learn at Test Time) instantiates this idea with model-valued hidden states: TTT-Linear Sun et al. (2025a) and TTT-MLP use linear and nonlinear predictors optimized through self-supervised objectives Hardt & Sun (2024); Sun et al. (2025b;a). End-to-End TTT Tandon et al. (2025) instead continues next-token training over the observed context and meta-learns an initialization suitable for inference-time adaptation. Subsequent work improves objective alignment and scalability. TTT-NTP uses next-position contextual hidden states as next-token-aligned fast-memory targets Ouyang et al. (2026), while retrieval-based and longer-horizon methods exploit nearest-neighbor data or future sequence signals during inference Hardt & Sun (2024); Hwang et al. (2026). In-Place TTT Feng et al. (2026), S-TTT Zhu et al. (2026), and QTTT Jian et al. (2024) further reduce the computational and memory overhead of fast-weight updates. Long-context studies also show that context-specific training can improve the usable context of pretrained LLMs beyond simply placing more tokens in the prompt Bansal et al. (2025). These methods primarily investigate how fast memory should be optimized and updated, whereas GAP focuses on what information it should preserve.

Memory Targets for Test-Time Adaptation A central design choice in TTT is the supervision written into fast memory. Existing methods encode context through key–value associations, local representation reconstruction, or model-level context objectives Sun et al. (2025b); Behrouz et al. (2024); Tandon et al. (2025). TTT-NTP improves alignment with language modeling by using projected future contextual hidden states as write targets Ouyang et al. (2026). Recent analysis further shows that a broad class of KV-binding TTT mechanisms can be interpreted as learned linear attention Liu et al. (2026), while GradMem writes context into compact memory through per-sample test-time gradient descent Kuratov et al. (2026). Related long-context and self-guided methods explore alternative predictive signals and learned update objectives Zhu et al. (2026); Yuksekgonul et al. (2026). Test-Time Context Distillation uses the hidden-state discrepancy between a long-window teacher and a short-window student to identify remote information useful for future prediction Wang et al. (2026). GAP differs in both its target and memory geometry: it extracts an output-aligned semantic prediction from the pretrained model itself and uses temporal consistency to jointly derive the predictive gap and its transition-aware write feature. Consequently, GAP stores only the request-specific information unexplained by the pretrained predictive prior, rather than complete hidden representations or teacher–student representation differences.

Neural Memory and Fast Weight Models Fast weights have long been studied as temporary associative memories that preserve recent information without retaining all previous activations Ba et al. (2016). Linear attention can likewise be interpreted as recurrent key–value accumulation or fast-weight programming Schlag et al. (2021); Katharopoulos et al. (2020). More recent architectures introduce bounded compressive or neural long-term memories for scalable context modeling, including Infini-attention, Titans, and ATLAS Munkhdalai et al. (2024); Behrouz et al. (2024; 2025). These works mainly improve memory architecture, capacity, update rules, and retrieval efficiency. GAP is complementary to this line of research: rather than proposing a larger memory architecture, it redefines the content of adaptive memory as Bellman-consistent predictive residuals and constructs the resulting request-local memory through causal ridge regression.

3 Method

GAP-TTT reformulates fast memory as a predictive corrector, aiming to capture information unexplained by the pretrained model rather than complete future representations. Given an input sequence $\{x_t\}_{t=0}^{T-1}$, GAP-TTT augments selected Transformer MLP blocks with lightweight temporary memory following InPlace-TTT Feng et al. (2026). For an MLP layer, the intermediate activation and original output are computed as $u_t = (\mathbf{W}_{\text{up}} h_t) \odot \sigma(\mathbf{W}_{\text{gate}} h_t)$ and $o_t^0 = \mathbf{W}_{\text{down}} u_t$, where $\odot$ denotes element-wise multiplication. Instead of modifying pretrained parameters, GAP-TTT introduces a lightweight adapter that generates context-dependent corrections. Specifically, the memory key is obtained as $k_t = \boldsymbol{\mathcal{K}}_l(u_t) \in \mathbb{R}^R$, where $\boldsymbol{\mathcal{K}}_l$ is a low-rank addressing projection. The correction code generated by the memory is mapped back to the hidden dimension through a lightweight decoder $\boldsymbol{\mathcal{U}}_l : \mathbb{R}^S \to \mathbb{R}^D$. Different from previous TTT methods that directly reconstruct future representations, GAP-TTT views the pretrained LLM as the slow predictor and aims to store only the information

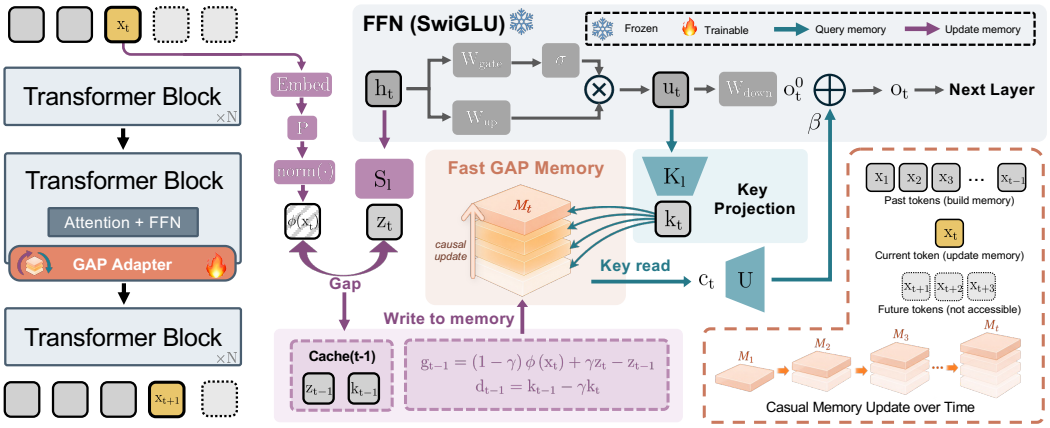


Figure 1: **Overview of GAP-TTT.** GAP-TTT decomposes sequence modeling into a frozen slow prior and a fast corrective memory. At each step, a lightweight predictor estimates the expected future semantic state $\hat{z}_t$, which is contrasted against the observed semantic representation $\phi(x_t)$. The resulting predictive residuals, capturing contextual dynamics unpredicted by the base model, are compiled into fast memory $\mathbf{M}$ using a Bellman-consistent writing strategy solved via causal ridge regression. At inference time, the model queries $\mathbf{M}$ with transition-aware features to retrieve context-specific semantic corrections, rectifying the base model's forward representations without requiring test-time gradient descent.

beyond its intrinsic prediction. To obtain this predictive state, we introduce a lightweight semantic predictor $\boldsymbol{\mathcal{S}}_l(\cdot)$ at each adapted layer, which projects hidden representations into an output-aligned semantic space:

$$z_t = \boldsymbol{\mathcal{S}}_l(h_t), \qquad z_t \in \mathbb{R}^S. \tag{1}$$

Here, z_t represents the semantic prediction extracted from the pretrained representations. The semantic space is defined as $\phi(x) = \mathrm{Norm}(\boldsymbol{\mathcal{P}}[\mathrm{Embed}(x)])$, where $\mathrm{Embed}(x)$ denotes the token embedding and $\boldsymbol{\mathcal{P}}$ is a lightweight semantic projection. GAP-TTT focuses on the missing predictive information. For a given observed prefix, GAP-TTT models the future semantic trajectory through a predictable component and a context-dependent residual:

$$\widehat{V}_t(\mathbf{M}) = z_t + \mathbf{M}k_t, \qquad \widehat{V}_t(\mathbf{M}) \approx V_t^*. \tag{2}$$

where V_t^* denotes the target future semantic trajectory constructed from subsequent tokens and $\mathbf{M} \in \mathbb{R}^{S\times R}$ is a shared candidate memory matrix used to fit the observed transitions, with their semantic predictions and keys treated as given within this fitting problem. At read position q, its estimate is denoted by $\mathbf{M}_q$ and is constructed using only completed transitions $t < q$, yielding the correction $\mathbf{M}_q k_q$. Sharing the candidate parameter $\mathbf{M}$ within each prefixwise fit does not require the estimates for different prefixes to be identical. GAP-TTT therefore aims to avoid repeatedly encoding predictable information and concentrates adaptation on predictive residuals revealed by the current context.

3.1 Bellman-consistent Gap Derivation

A central question in GAP-TTT is what information should be stored in fast memory. Since the pretrained model already captures part of the future semantics, storing complete future states would introduce redundant information. GAP-TTT therefore focuses on the predictive discrepancy between the model's current semantic prediction and the subsequently observed future semantics. To derive this residual causally, we exploit the temporal consistency of future semantic trajectories. Unlike conventional per-token-level prediction, GAP-TTT models a multi-horizon semantic trajectory that aggregates information across multiple future positions: $V_t^* = (1-\gamma)\sum_{j=1}^{\infty}\gamma^{j-1}\phi(x_{t+j})$, where $\gamma \in [0, 1)$ controls the prediction horizon. This trajectory satisfies the temporal consistency relation $V_t^* = (1-\gamma)\phi(x_{t+1}) + \gamma V_{t+1}^*$.

The temporal consistency relation provides a natural decomposition of future semantic evolution: the future state consists of the immediate semantic observation and the remaining future trajectory. Therefore, instead of storing the entire future representation, GAP-TTT seeks to capture

the discrepancy between the current semantic prediction and the newly observed future information. For each completed transition $t \to t+1$, we evaluate both $\widehat{V}_t(\mathbf{M}) = z_t + \mathbf{M}k_t$ and $\widehat{V}_{t+1}(\mathbf{M}) = z_{t+1} + \mathbf{M}k_{t+1}$ using the same candidate memory matrix $\mathbf{M}$. Imposing the temporal consistency condition $\widehat{V}_t(\mathbf{M}) = (1-\gamma)\phi(x_{t+1}) + \gamma\widehat{V}_{t+1}(\mathbf{M})$ and rearranging yields:

$$\mathbf{M}(k_t - \gamma k_{t+1}) = (1-\gamma)\phi(x_{t+1}) + \gamma z_{t+1} - z_t. \tag{3}$$

Therefore, GAP-TTT naturally derives the transition-aware write feature $d_t = k_t - \gamma k_{t+1}$ and the Bellman-consistent gap target $g_t = (1-\gamma)\phi(x_{t+1}) + \gamma z_{t+1} - z_t$. Both the memory target and its corresponding write feature follow from the same temporal consistency constraint, rather than being specified independently. The resulting target measures the temporal prediction discrepancy in the semantic space, allowing fast memory to model predictive corrections rather than reconstruct complete future representations. In practice, GAP-TTT estimates $\mathbf{M}_q$ by fitting the residuals $\mathbf{M}d_t - g_t$ over completed transitions $t < q$ using causal ridge regression, as described in the next subsection. Thus, the shared candidate $\mathbf{M}$ defines the regression problem, while its prefix-dependent solution $\mathbf{M}_q$ evolves as new observations arrive.

3.2 Causal Gap Memory Construction

Given write features $\{d_t\}$ and contextual gap targets $\{g_t\}$, GAP-TTT constructs a temporary memory through causal ridge regression. For each position q, the memory state is estimated only from previous transitions:

$$\mathbf{M}_q = \arg\min_{\mathbf{M}} \sum_{i<q} \|\mathbf{M}d_i - g_i\|_2^2 + \lambda\|\mathbf{M}\|_F^2. \tag{4}$$

The closed-form solution is $\mathbf{M}_q = \mathbf{B}_q\mathbf{A}_q^{-1}$, where $\mathbf{A}_q = \lambda I + \sum_{i<q} d_i d_i^\top$ and $\mathbf{B}_q = \sum_{i<q} g_i d_i^\top$. This causal formulation guarantees that each token only accesses previously observed information and prevents future leakage. Unlike gradient-based test-time adaptation, the ridge formulation estimates the memory state analytically from observed transitions without iterative optimization. This design enables efficient adaptation through both parallel prefix computation and streaming recursive updates. For more details, please refer Appendix C. During prompt prefill, GAP-TTT constructs memory states through causal prefix accumulation over the available context. During autoregressive generation, when a new token arrives, the previous transition is completed, the memory is updated through recursive least squares, and the current correction is retrieved before generating the next token. Therefore, training-time and inference-time adaptation follow the same causal memory construction. To improve parameter efficiency, GAP-TTT parameterizes the addressing projection and correction decoder using low-rank adapters. After retrieving the correction code $c_t = \mathbf{M}_t k_t$, the correction is projected back to the hidden dimension and injected into the original MLP output:

$$o_t = o_t^0 + \beta \boldsymbol{\mathcal{U}}_l c_t. \tag{5}$$

When the pretrained model already explains the observed context, the gap remains small and introduces minimal modification. Conversely, when newly observed information cannot be predicted by the pretrained model, GAP-TTT provides targeted corrections through the temporary memory. We also conducted dedicated experiments to verify this; please refer to Table 6 for details.

3.3 Training Objective

GAP-TTT is optimized with the causal language modeling objective together with an auxiliary semantic prediction alignment objective. Different from previous TTT methods that optimize an independent value prediction objective, GAP-TTT does not require learning a separate future-state predictor. Instead, the semantic predictor only provides a stable predictive space, while the fast memory is responsible for modeling the remaining contextual residual. Given the future semantic trajectory V_t^* and the predicted semantic state z_t, we introduce a semantic alignment loss in Eq. 6. The final training objective combines the standard causal language modeling loss with the gap alignment loss in Eq. 7:

$$\mathcal{L}_{\text{gap}} = \sum_t \|z_t - \text{sg}(V_t^*)\|_2^2 \tag{6}$$

$$\mathcal{L} = \mathcal{L}_{\text{lm}} + \lambda_{\text{gap}}\mathcal{L}_{\text{gap}} \tag{7}$$

where $\text{sg}(\cdot)$ denotes stop-gradient to stabilize the semantic target. The language modeling objective remains the primary optimization signal, while the gap loss in Eq. 6 provides auxiliary supervision

Table 1: Performance on RULER across different context lengths with different base models. Best results are in **bold**; second-best results are underlined. For Qwen3-4B, we extend its context window using YaRN for efficient training and inference.

Methods	4K	8K	16K	32K	64K	**Avg**
Qwen3-4B						
Base Yang et al. (2025a)	91.46	88.75	87.06	82.38	72.45	84.42
CPT	91.49	88.68	88.14	81.21	72.25	84.35
qTTT Bansal et al. (2025)	92.53	88.41	88.37	82.94	71.38	84.73
In-Place TTT Feng et al. (2026)	92.15	88.39	88.67	81.29	73.22	84.74
TTT-NTP Ouyang et al. (2026)	91.77	88.96	88.48	82.19	73.55	84.99
GAP-TTT (Ours)	**93.14**	**89.65**	**89.78**	**84.20**	**74.75**	**86.30**
LLaMA3.1-8B						
Base Grattafiori et al. (2024)	90.84	88.59	88.17	80.18	78.89	85.33
CPT	90.92	89.43	88.03	83.49	81.25	86.62
qTTT Bansal et al. (2025)	90.77	88.66	88.34	81.64	79.14	85.71
In-Place TTT Feng et al. (2026)	89.45	88.13	88.44	82.78	78.65	85.49
TTT-NTP Ouyang et al. (2026)	88.58	87.62	88.15	83.06	78.22	85.13
GAP-TTT (Ours)	**92.83**	**90.74**	**89.81**	**84.98**	**81.41**	**87.95**

for learning a meaningful semantic prediction space. Consequently, the learned GAP-TTT adapters acquire the ability to identify and correct context-specific prediction failures rather than memorizing redundant information already encoded by the pretrained model.

4 EXPERIMENTS

Implementation Details. We attach GAP-TTT adapters to the MLP down-projection layers of the Transformer and insert them every four layers by default. We train the GAP-TTT in two settings: (1) freezing the backbone parameters and training only the TTT-adapter; (2) performing full-parameter joint continuous pre-training of the GAP-TTT and the backbone. Specifically, we perform continuous pre-training on the Prolong-64K Gao et al. (2024) dataset on a 16-card H200. Training consists of two phases: 32K length with 15B tokens, followed by 64K length with 10B tokens. For more details, please refer to Appendix A and B.

Baselines and Evaluation Settings We use Qwen3-4B and LLaMA3.1-8B-Base as the backbone to compare GAP-TTT with several robust open-source TTT baselines, including InPlaceTTT Feng et al. (2026), TTT-NTP Ouyang et al. (2026), CPT, and qTTT Bansal et al. (2025), covering representative methods based on efficient and fast weight adaptation, predictive memory targets, continuous pre-training, and parameter-efficient test-time learning. All methods are evaluated under consistent settings to ensure fair comparison. For long context adaptation, we use RULER Hsieh et al. (2024) and LongBench-v2 Bai et al. (2025). RULER comprehensively evaluates long context understanding across four synthetic tasks (retrieval, multi-hop inference, aggregation, and question answering), comprising 13 sub-tasks with context lengths ranging from 4K to 64K; we report the average performance. We further evaluate GAP on LongBench-v2. Specifically, we use a moderately challenging test environment covering six domains with context lengths ranging from 32K to 128K. In addition to the long-text evaluation, we also measured GAP-TTT's general capability retention across standard downstream tasks, including MMLU, ARC-C, ARC-E, PIQA, and HellaSwag.

4.1 MAIN RESULTS.

Results on RULER. To verify that GAP-TTT can serve as an effective TTT enhancement method for the base model, we conducted experiments using two open-source models: Qwen3-4B and LLaMA3.1-8B . The former has a context window of 32K, which was expanded using YaRN during training. The latter has a context window of 128K, so no further expansion was needed. We first tested on RULER, specifically using two training paradigms: (1) training only the TTT-adapter parameters while freezing the base model; (2) unfreezing the base model and jointly training the TTT-adapter. This helps to explore whether GAP-TTT can serve as a parameter-efficient scheme to enhance the performance of the base model. The results are shown in Table 1.

The results in this table are obtained under Paradigm 2, strictly adhering to the original evaluation settings of the baseline methods. In the implementation, we set the rank of the low-rank projection to 128. The results show that under the same experimental scheme, GAP-TTT consistently outperforms all baseline methods in both backbone models, demonstrating its effectiveness in long context adaptation. Particularly noteworthy is the increasing performance advantage of GAP-TTT with increasing context length. At context lengths of 32K and 64K, GAP-TTT performs best among all comparison methods, highlighting its superior ability to handle increasingly complex long context scenarios.

Results on LongBench-v2. Since RULER mainly consists of synthetic long-context tasks, we further evaluate GAP-TTT on LongBench-v2 to measure its effectiveness in realistic long-document scenarios. We focus on the medium-difficulty split, which contains 215 questions with context lengths ranging from 32K to 128K tokens across six task domains. As shown in Table 2, GAP-TTT achieves the best overall performance, outperforming existing TTT methods and CPT. In particular, GAP obtains consistent improvements on challenging tasks such as multi-document understanding, in-context learning, and structured reasoning, demonstrating the effectiveness of storing contextual prediction gaps rather than complete representations.

Table 2: LongBench-v2 medium-difficulty splitting results, categorized by task domain.

Method	Sin-doc	Mul-doc	ICL	Dialogue	Code	Struct	Avg
LLaMA3.1-8B	26.0	25.0	27.9	15.8	33.3	26.1	25.7
CPT	29.3	27.6	24.8	21.7	**35.3**	26.6	27.6
In-Place TTT	29.7	28.4	26.9	**22.5**	33.6	26.1	27.9
TTT-NTP	**31.4**	26.3	27.7	17.6	32.9	28.8	27.5
GAP-TTT (ours)	**30.9**	**28.6**	**29.4**	19.7	34.4	**30.3**	**28.9**

General Capability Evaluation. To verify whether GAP-TTT would impair other general capabilities of the model, such as commonsense reasoning and semantic understanding, we conducted experiments on multiple datasets using Qwen3-4B and LLaMA3.1-8B trained with GAP-TTT, as well as the native base model. The results are shown in Table 3. The results show that GAP does not significantly impair the general capabilities of the original model; although performance on some tasks is inferior to the base model, the overall performance of the model is still improved.

Table 3: An evaluation of the GAP-TTT trained backbone on standard common sense and knowledge benchmark tests was conducted to examine whether general capabilities were impaired.

Backbone	Method	HellaSwag	ARC-E	ARC-C	PIQA	MMLU	Avg
LLaMA3.1-8B	Base	71.15	80.40	58.02	83.40	**63.45**	71.28
	GAP-TTT (ours)	**72.04**	**81.23**	**59.88**	**84.57**	62.68	**72.08**
Qwen3-4B	Base	65.24	74.35	**50.28**	79.77	73.88	68.70
	GAP-TTT (ours)	**66.43**	**75.16**	48.74	**80.21**	**74.66**	**69.04**

Parameter Efficiency. To investigate whether lightweight test-time adaptation can effectively improve the long-context capabilities of pre-trained LLMs, we froze all backbone network parameters and optimized only the parameters of the TTT-adapter. We evaluated different TTT methods on the RULER dataset using the LLaMA3.1-8B model under the same adaptation settings. For fair comparison, GAP-TTT and In-Place-TTT were inserted into the same layers. Since GAP-TTT only stores context-specific information that the pre-trained model cannot interpret, rather than reconstructing complete future representations, the number of trainable parameters introduced by GAP-TTT is significantly reduced when combined with low-rank parameterization: our implementation requires only 25.2M additional parameters, compared to 100.6M for In-Place TTT, reducing the trainable parameter scale by approximately 4×. The results are shown in Table 4. GAP-TTT achieves the best overall performance with

Table 4: Performance of training only TTT-adapter parameters on RULER.

Methods	Param.	4K	8K	16K	32K	Avg
LLaMA3.1-8B	0	90.84	88.59	88.17	80.18	86.94
In-Place TTT	100.66	88.92	87.33	85.67	77.60	84.88
TTT-NTP	100.66	88.75	85.96	86.43	80.29	85.36
GAP-TTT (Ours)	**25.24**	**89.88**	**89.47**	**88.54**	**82.61**	**87.63**

an average score of 87.63 while significantly reducing the number of trainable parameters. These results demonstrate that storing prediction residuals is more effective than directly memorizing future representations, enabling more efficient parameter utilization and more effective test-time adaptation.

4.2 Ablations.

Memory Targets. To validate our hypothesis that predictive residuals are a more effective memory target than directly storing future representations, we conduct a controlled analysis by replacing the memory target with alternative memory objectives while keeping the backbone, memory architecture, and training configuration unchanged. Specifically, we compare three memory formulations: (1) **Hidden-State Memory**, which follows TTT-NTP Ouyang et al. (2026) and trains the fast memory to reconstruct future hidden representations; (2) **KV-Binding Memory**, which stores future key-value representations and represents another common strategy for context memorization; and (3) **Predictive Residual Memory (GAP-TTT)**, which stores only the discrepancy between the pretrained model prediction and the observed future semantic trajectory. All variants used the same LLaMA3.1-8B backbone network and the same TTT config, training only the TTT-adapter. The only difference was the information written to the fast memory. For Hidden-State Memory, the writing target is the predicted future hidden state h_{t+1}; for Key-Value Binding Memory, the writing target is the future key-value representation; while GAP uses the Bellman-consistent semantic gap $g_t = (1-\gamma)\phi(x_{t+1}) + \gamma z_{t+1} - z_t$ together with the transition-aware write feature $d_t = k_t - \gamma k_{t+1}$. The results are summarized in Table 5. GAP-TTT achieves the best overall performance among all memory targets. These results demonstrate that directly storing future representations introduces redundant information, while predictive residuals provide a more effective memory target by selectively preserving information that cannot be explained by the pretrained model.

Table 5: Effect of different memory targets on RULER. All variants use the same backbone and TTT adapter configuration.

Memory Target	4K	8K	16K	32K	**Avg**
Base	90.84	88.59	88.17	80.18	86.94
Hidden-State	88.75	85.96	86.43	80.29	85.36
KV-Binding	88.71	86.96	84.53	79.82	85.01
GAP-TTT	**89.88**	**89.47**	**88.54**	**82.61**	**87.63**

Adaptive Corrections To examine whether GAP-TTT produces different correction strengths for contexts of varying difficulty, we use the Arxiv dataset and distinguish "known" and "unknown" contexts based on the prediction confidence of the pre-trained LLM. Specifically, samples with low perplexity and high next-token prediction confidence are regarded as known contexts that can already be well explained by the pre-trained model, while those with high perplexity and low confidence are treated as unknown contexts that are less well covered by its pre-trained knowledge. We select 500 samples for each category and measure the correction magnitude $\|UM_t k_t\|_2$. A smaller magnitude indicates that little additional correction is required, whereas a larger magnitude suggests that the fast memory provides stronger compensation. As shown in Table 6, GAP-TTT produces substantially smaller corrections for known contexts and stronger corrections for unknown ones. This suggests that GAP-TTT avoids redundantly memorizing information already captured by the pre-trained model and instead allocates larger corrections to less predictable information. Overall, these results support the view that GAP-TTT functions as a prediction-correction memory rather than a redundant context-storage mechanism.

Table 6: Analysis of GAP-TTT memory behavior on different levels of predictability.

Context Type	$\|\|UM_t k_t\|\|_2$
Known Context	0.11
Unknown Context	**0.31**

Gap Formulation. To investigate the importance of the consistent gap formulation Eq. 3, we compare GAP-TTT with two variants using different memory targets while keeping all other components unchanged. Specifically, (1) **Direct Residual** removes the temporal consistency modeling and directly uses the immediate semantic residual $g_t = \phi(x_{t+1}) - z_t$. This is equivalent to setting γ to 0. (2) **Hidden Residual** further replaces the semantic residual with hidden-state differences, i.e., $g_t = h_{t+1} - h_t$. The results are shown in Table 7. GAP-TTT achieves the best performance, demonstrating that modeling future semantic evolution through Bellman consistency provides a more effective memory target than single-step residuals or hidden-state differences.

Table 7: Gap formulation ablation on RULER.

Target	4K	8K	16K	32K	Avg
Hidden	89.42	87.31	86.62	80.47	85.96
Direct	89.24	88.15	87.02	81.36	86.44
GAP-TTT	**89.88**	**89.47**	**88.54**	**82.61**	**87.63**

Table 8: Write feature ablation on RULER.

Feature	4K	8K	16K	32K	Avg
k_t	89.32	88.62	87.71	81.74	86.85
$k_t - \gamma k_{t+1}$	**89.88**	**89.47**	**88.54**	**82.61**	**87.63**

Transition write. GAP-TTT derives the write feature $d_t = k_t - \gamma k_{t+1}$ from the temporal consistency constraint, rather than directly using the current key as the memory address. To verify the effectiveness of this design, we compare GAP-TTT with a variant that replaces the $d_t = k_t - \gamma k_{t+1}$ with $d_t = k_t$ while keeping the memory target unchanged. As shown in Table 8, the design can improve performance, indicating that modeling key evolution provides a better addressing mechanism for storing predictive residuals.

Efficiency Analysis. GAP-TTT is designed to be a lightweight and efficient adaptation mechanism. While its parameter efficiency has been discussed in Section 4.1, we further evaluate its inference efficiency in this section. We conduct experiments on the RULER using an H200, covering context lengths from 4K to 32K with a maximum generation length of 128. For each context length, we randomly sample 50 tasks and report the averaged inference latency, decoding throughput (Thrpt), peak memory consumption, and other efficiency metrics.

Table 9: Inference efficiency comparison on RULER.

Method	Time (s)	Thrpt (Tok/s)	Mem (GB)
Base	2.84	29.4	18.58
Inplace-TTT	3.55	25.3	21.49
TTT-NTP	3.42	26.1	21.15
GAP-TTT (Ours)	**3.21**	**28.1**	**19.43**

Hyperparameter. Here, we study the sensitivity of GAP-TTT to three key hyperparameters: the multi-view coefficient γ, low-rank projection dimension r, and layer insertion interval **every:** n. All experiments freeze the base model and train only the TTT-adapter parameters. We report the average RULER performance over 4K–32K contexts using LLaMA3.1-8B as the backbone. For γ, we evaluate values from 0 to 0.8. As shown in Table 10, performance improves as γ increases to 0.5, suggesting that longer semantic trajectories provide richer predictive signals. Performance decreases at $\gamma = 0.8$, likely because distant predictions introduce greater uncertainty into the estimated semantic gap. For the layer insertion interval **every:** n, we test $n = 2, 4, 6, 8$, with $n = 4$ achieving the best performance by balancing adaptation capacity and parameter efficiency. Finally, for the low-rank projection dimension r, we evaluate $r = 16, 32, 64, 128$. GAP-TTT performs best at $r = 128$, indicating that a larger projection space provides sufficient capacity to capture diverse contextual residuals.

Table 10: Hyperparameter analysis of GAP on RULER.

Setting	Value	Avg
γ	0.0	86.44
	0.3	86.58
	0.5	**87.63**
	0.8	86.52
$every : n$	2	85.88
	4	**87.63**
	6	86.95
	8	86.02
$rank : r$	16	85.15
	32	85.87
	64	86.74
	128	**87.63**

5 Conclusion

In this paper, we introduce GAP-TTT, a predictive memory framework for test-time training that redefines fast memory as a mechanism for storing context-dependent corrections rather than complete representations. By viewing the pretrained LLM as a slow predictor, GAP-TTT extracts predictive gaps in an output-aligned semantic space and derives both memory targets and transition-aware write features through a unified future semantic consistency formulation. This Bellman-consistent design enables causal ridge-based memory construction without iterative test-time optimization. Experiments on long-context benchmarks demonstrate that GAP-TTT consistently improves over existing TTT approaches with fewer trainable parameters while preserving general model capabilities. These results highlight the potential of predictive residuals as an effective principle for designing adaptive memory in large language models. Future work will explore more scalable semantic memory representations and broader applications of predictive residual adaptation.

AI USE STATEMENT

In writing this paper, we used generative AI tools to help polish the text, check grammar, improve the readability of technical explanations, and provide feedback on the organization of figures and experimental demonstrations. Furthermore, we utilized AI tools to assist discussion, explore different ways of expression, and enhance the clarity of the article. However, the core scientific contributions of this paper, including the proposed framework, theoretical motivation, mathematical derivation, experimental methods, and the implementation of the proposed methods, were not generated using any generative AI tools. All experimental results, analyses, and conclusions were independently verified by the authors. The authors bear full responsibility for all content in this paper, including the methodology, arguments, experimental results, and the correctness of the final manuscript.

REFERENCES

Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past, 2016. URL https://arxiv.org/abs/1610.06258.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2025. URL https://arxiv.org/abs/2412.15204.

Rachit Bansal, Aston Zhang, Rishabh Tiwari, Lovish Madaan, Sai Surya Duvvuri, Devvrit Khatri, David Brandfonbrener, David Alvarez-Melis, Prajjwal Bhargava, Mihir Sanjay Kale, and Samy Jelassi. Let's (not) just put things in context: Test-time training for long-context llms, 2025. URL https://arxiv.org/abs/2512.13898.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, 2024. URL https://arxiv.org/abs/2501.00663.

Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. Atlas: Learning to optimally memorize the context at test time, 2025. URL https://arxiv.org/abs/2505.23735.

Guhao Feng, Shengjie Luo, Kai Hua, Ge Zhang, Wenhao Huang, Di He, and Tianle Cai. In-place test-time training. In *International Conference on Learning Representations*, volume 2026, pp. 114452–114472, 2026.

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.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. Enabling large language models to generate text with citations. 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models, 2024. URL https://arxiv.org/abs/2305.18466.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Hee Seung Hwang, Xindi Wu, Sanghyuk Chun, and Olga Russakovsky. Reinforced fast weights with next-sequence prediction, 2026. URL https://arxiv.org/abs/2602.16704.

Damien Jian, Yu-Chao Huang, and Hsi-Sheng Goan. Test-time training with quantum auto-encoder: From distribution shift to noisy quantum circuits, 2024. URL https://arxiv.org/abs/2411.06828.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://arxiv.org/abs/2006.16236.

Yuri Kuratov, Matvey Kairov, Aydar Bulatov, Ivan Rodkin, and Mikhail Burtsev. Gradmem: Learning to write context into memory with test-time gradient descent, 2026. URL https://arxiv.org/abs/2603.13875.

Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 133(1):31–64, 2025.

Junchen Liu, Sven Elflein, Or Litany, Zan Gojcic, and Ruilong Li. Test-time training with kv binding is secretly linear attention, 2026. URL https://arxiv.org/abs/2602.21204.

Yuejiang Liu, Parth Kothari, Bastien Van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. Ttt++: When does self-supervised test-time training fail or thrive? *Advances in Neural Information Processing Systems*, 34:21808–21820, 2021.

Jing Ma. Improved self-training for test-time adaptation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23701–23710. IEEE, 2024.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention, 2024. URL https://arxiv.org/abs/2404.07143.

Xuan Ouyang, Zefan Cai, and Junjie Hu. Test-time training with next-token prediction. *arXiv preprint arXiv:2606.21803*, 2026.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624, 2023.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers, 2021. URL https://arxiv.org/abs/2102.11174.

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, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states, 2025a. URL https://arxiv.org/abs/2407.04620.

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, 2025b. URL https://arxiv.org/abs/2407.04620.

Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin, Jed McCaleb, Yejin Choi, and Yu Sun. End-to-end test-time training for long context, 2025. URL https://arxiv.org/abs/2512.23675.

Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, et al. Mesanet: Sequence modeling by locally optimal test-time training. In *International Conference on Learning Representations*, volume 2026, pp. 26397–26447, 2026.

Zixuan Wang, Xingyu Dang, Rui-Jie Zhu, Zixin Wen, Hengyu Fu, Wenhao Chai, and Jason D. Lee. Learning what to remember: Test-time training via context distillation, 2026. URL https://arxiv.org/abs/2608.01672.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Xihong Yang, Yiqi Wang, Jin Chen, Wenqi Fan, Xiangyu Zhao, En Zhu, Xinwang Liu, and Defu Lian. Dual test-time training for out-of-distribution recommender system. *IEEE Transactions on Knowledge and Data Engineering*, 37(6):3312–3326, 2025b.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175.

Xinyu Zhu, Zhe Xu, Xiaohan Wei, Yunchen Pu, Fei Tian, Chonglin Sun, Kaushik Rangadurai, Hua Zhi, Frank Shyu, Sandeep Pandey, Luke Simon, Yu Meng, and Xi Liu. Self-guided test-time training for long-context llms, 2026. URL https://arxiv.org/abs/2607.09415.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Xinwei Long, Ermo Hua, Biqing Qi, et al. Ttrl: Test-time reinforcement learning. *Advances in Neural Information Processing Systems*, 38:131459–131483, 2026.

A IMPLEMENTATION DETAILS.

In all experiments, models are loaded with **bfloat16** precision and FlashAttention-2 is enabled. GAP is inserted every 4 layers, with introduced modules including key projection, semantic predictor, and value decoder. We optimize GAP using the AdamW optimizer with a cosine learning rate schedule. The learning rates for key projection and value decoder are set to 5×10^{-5}, for the semantic predictor to 2×10^{-4}, and for the ridge coefficient to 1×10^{-4}. Gradient clipping is applied with a maximum norm of 1.0. When unfreezing the backbone network, we set the learning rate of its parameters to 1×10^{-6}.

Table 11: Implementation hyperparameters of GAP.

Hyperparameter	Value
Backbone	Llama-3.1-8B / Qwen3-4B
Adapted layers	every 4
Rank	128
Sequence length	32K(stage 1) / 64K(stage 2)
Optimizer	AdamW
Warmup ratio	0.01
Weight decay	0.01
Gradient clipping	1.0
Gap loss weight	2.0

B CONTINUAL PRE-TRAINING DATA

GAP is trained on the official ProLong-64K continual pre-training mixture, which contains diverse long-context data from ten domains. The mixture combines code, books, educational documents, web corpora, mathematical texts, academic papers, and knowledge-intensive documents. The data distribution follows the original ProLong-64K recipe, as shown in Table 12.

Table 12: Composition of the ProLong-64K continual pre-training mixture.

Domain	Ratio
Code repositories	30%
Books	30%
FineWeb-Edu	10%
FineWeb	10%
StackExchange	4%
Wikipedia (Dolma)	4%
Textbooks	3%
Tulu-v2	3%
ArXiv	3%
OpenWebMath	3%

C MEMORY-EFFICIENT EXACT CAUSAL RIDGE EVALUATION

For each valid write position i, the Bellman memory produces the rank-one sufficient-statistic increments

$$\Delta \mathbf{A}_i = \mathbf{d}_i \mathbf{d}_i^\top, \qquad \Delta \mathbf{B}_i = \boldsymbol{\delta}_i \mathbf{d}_i^\top, \tag{8}$$

where $\mathbf{d}_i \in \mathbb{R}^R$ is the TD writer feature and $\boldsymbol{\delta}_i \in \mathbb{R}^S$ is the Bellman gap. Masked or invalid transitions contribute zero increments. At read position q, strict causality requires the ridge statistics

$$\mathbf{A}_q = \lambda \mathbf{I} + \sum_{i<q} \Delta \mathbf{A}_i, \qquad \mathbf{B}_q = \sum_{i<q} \Delta \mathbf{B}_i. \tag{9}$$

The corresponding correction is

$$\mathbf{z}_q = (\mathbf{A}_q + \epsilon \mathbf{I})^{-1} \mathbf{k}_q, \qquad \mathbf{c}_q = \mathbf{B}_q \mathbf{z}_q, \tag{10}$$

where $\mathbf{k}_q \in \mathbb{R}^R$ is the reader key, $\lambda > 0$ is the ridge coefficient, and $\epsilon \geq 0$ is an optional numerical stabilizer. Directly materializing $\mathbf{A}_q$ and $\mathbf{B}_q$ for every position requires $\mathcal{O}\big(T(R^2 + SR)\big)$ intermediate memory. We avoid this cost using a hierarchical exclusive scan composed of tile summaries, tile-level prefixes, and memory-bounded grouped solves.

Tile summaries. We partition the sequence into $N = \lceil T/L \rceil$ contiguous tiles $\{\mathcal{T}_j\}_{j=0}^{N-1}$, each containing at most L positions. For each tile, we compute only its aggregate contribution:

$$\overline{\mathbf{A}}_j = \sum_{i \in \mathcal{T}_j} \Delta\mathbf{A}_i, \qquad \overline{\mathbf{B}}_j = \sum_{i \in \mathcal{T}_j} \Delta\mathbf{B}_i. \tag{11}$$

These summaries compactly represent the state transition induced by an entire tile without materializing token-wise prefix states.

Exclusive tile prefix. An exclusive scan over the tile summaries gives the memory state at the entrance of tile j:

$$\mathbf{A}_j^{\text{start}} = \lambda\mathbf{I} + \sum_{m<j} \overline{\mathbf{A}}_m, \qquad \mathbf{B}_j^{\text{start}} = \sum_{m<j} \overline{\mathbf{B}}_m. \tag{12}$$

The scan is exclusive: updates generated inside tile j are not included in its entrance state. Within each tile, a second exclusive scan reconstructs the exact state for every read position $q \in \mathcal{T}_j$:

$$\mathbf{A}_q = \mathbf{A}_j^{\text{start}} + \sum_{\substack{i \in \mathcal{T}_j \\ i<q}} \Delta\mathbf{A}_i, \qquad \mathbf{B}_q = \mathbf{B}_j^{\text{start}} + \sum_{\substack{i \in \mathcal{T}_j \\ i<q}} \Delta\mathbf{B}_i. \tag{13}$$

Grouped batched solve. To bound peak memory, we materialize the token-wise states of only G tiles at a time. For all positions in the active group, the systems in Eq. 10 are solved using batched Cholesky factorization. Each position retains its own $(\mathbf{A}_q, \mathbf{B}_q, \mathbf{k}_q)$; grouping only controls execution scheduling and does not share or approximate the ridge state across positions. Once the corrections for the active group have been computed, its token-wise states may be released before processing the next group.

Exactness and causality. For any $q \in \mathcal{T}_j$, the indices preceding q decompose into the disjoint union

$$\{i : i < q\} = \left(\bigcup_{m<j} \mathcal{T}_m \right) \cup \{i \in \mathcal{T}_j : i < q\}. \tag{14}$$

Substituting Eq. 11 and 12 into Eq. 13 therefore recovers Eq. 9 exactly. Consequently, the hierarchical implementation is algebraically equivalent to token-by-token causal ridge evaluation and introduces no tile-boundary approximation. In particular, the model output is invariant to L and G, up to floating-point accumulation error. Because $\Delta\mathbf{A}_i \succeq 0$ and $\lambda + \epsilon > 0$, each system matrix satisfies

$$\mathbf{A}_q + \epsilon\mathbf{I} \succeq (\lambda + \epsilon)\mathbf{I} \succ 0, \tag{15}$$

which guarantees a unique solution and permits stable Cholesky-based evaluation.

Memory complexity. The peak intermediate memory of the hierarchical procedure is

$$\mathcal{O}\big((N + GL)\,(R^2 + SR)\big), \qquad N = \left\lceil \frac{T}{L} \right\rceil, \tag{16}$$

excluding model activations and the final output tensor. This replaces the $\mathcal{O}\big(T(R^2 + SR)\big)$ cost of fully materialized token-wise prefixes. The procedure does not change the underlying dense-solve arithmetic complexity; rather, it provides a tunable memory–throughput trade-off through L and G while preserving exact gradients and strict token-level causality. Unlike chunk-delayed TTT, the tile length L is therefore a purely computational parameter: it determines how the exact causal computation is scheduled, rather than how frequently the fast memory is updated.