SADA: Bridging In-Context Learning and Fine-Tuning via State-Aligned Distillation Adapters

Anonymous ACL submission

Abstract

Prompt-based in-context learning (ICL) and parameter fine-tuning are two dominant paradigms for incorporating external information into large language models (LLMs), but they incur high inference costs or require expensive retraining. To bridge this gap, context-to-parameter mapping converts prompts into temporary adapter weights. However, we identify a critical failure mode in existing methods: *hidden-state collapse*, where the adapter-augmented model's internal states diverge sharply from the full-context oracle in deeper layers. We trace this failure to two coupled gaps: suboptimal **Input-Selection** and inadequate **Supervision-Signal**. To address these issues, we propose SADA (**S**tate-**A**ligned **D**istillation **A**dapters). We establish the *attention-block output* as a principled feature interface to improve input selection and introduce *state-alignment distillation* to enforce consistency between the adapter-augmented model and the full-context oracle. Experiments on long-context language modeling (PG19) and downstream NLU and summarization benchmarks show that SADA consistently outperforms StreamAdapter and GenerativeAdapter, achieving performance comparable to ICL while significantly reducing memory footprint and latency. We further analyze when parameterized context compression is effective and when explicit context retention remains preferable. Our code is available at `https://anonymous.4open.science/r/SADA-F924`.

1 Introduction

In recent years, Large Language Models (LLMs) have achieved strong performance across many tasks (Minaee et al., 2024; Zhao et al., 2023), yet general-purpose models still struggle with domain-specific or complex real-world requirements (Chen et al., 2025a). Consequently, efficient and precise knowledge injection has become

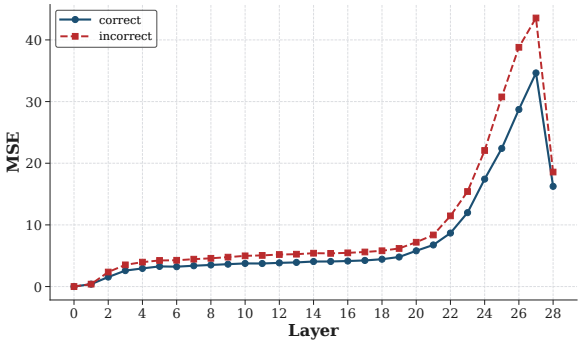


Figure 1: **Hidden-state fidelity predicts correctness.** Layer-wise hidden-state MSE between the injected model and the full-context oracle.

a central problem (Lv et al., 2025). Currently, two paradigms dominate this landscape: parameter-update-based fine-tuning (FT) and prompt-based In-Context Learning (ICL) (Dong et al., 2024; Han et al., 2024).

Fine-tuning internalizes new information into model weights, yielding efficient inference once training is done, but it is costly to repeat and hard to control for localized knowledge edits. Updating parameters can also unintentionally overwrite prior capabilities (i.e., catastrophic forgetting) (Li et al., 2024a; van de Ven et al., 2024).

In contrast, ICL keeps evidence explicit in the prompt, making updates reversible and more controllable. However, long contexts are slow and memory-intensive (Wan et al., 2024): compute and the KV cache scale linearly with context length, quickly exhausting GPU memory and limiting batch size.

To bridge the gap between the flexibility of ICL and the efficiency of FT, a new paradigm known as **Context-to-Parameter Mapping** (or Test-Time Adaptation) has emerged. Pioneering works such as *GenerativeAdapter* (Chen et al., 2025b) and *StreamAdapter* (Muhtar et al., 2024) convert contextual representations into temporary adapter pa-

rameters (e.g., LoRA weights) via a lightweight hypernetwork, aiming to achieve "fine-tuning-like adaptation at inference time" in a single forward pass.

However, we identify a critical failure mode in these approaches regarding *behavioral fidelity*. As illustrated in Figure 1, we compare the layer-wise internal states of the injected model StreamAdapter (no explicit prompt; adapted via injected adapters) against a full-context oracle. The results reveal a decisive correlation: when the model answers correctly, its hidden states remain close to the oracle (low MSE); conversely, incorrect answers diverge sharply in deeper layers. This highlights a concrete consequence: supervision on logits alone is insufficient, because once the intermediate reasoning trajectory deviates, the model can fail on complex reasoning or long-range dependencies even if the final outputs are matched.

Viewed through this lens, existing methods fail to achieve such alignment due to two coupled gaps:

- **(i) Input-Selection Gap:** Prior methods such as *StreamAdapter* use the KV cache, while *GenerativeAdapter* uses raw hidden states as features for parameter generation. These choices are not explicitly grounded in how attention-driven context shifts induce effective weight updates, so the mapping module can fail to preserve the oracle trajectory.
- **(ii) Supervision-Signal Gap:** Current objectives primarily optimize for outcome matching (e.g., Cross-Entropy on Ground Truth). As evidenced by Figure 1, such supervision is insufficient to constrain intermediate states that support multi-step reasoning, leading to trajectory divergence in deep layers.

Motivated by this evidence, we propose **SADA** (**S**tate-**A**ligned **D**istillation **A**dapters). To resolve the *input-selection* gap, we analytically justify the **attention output** as a principled feature interface for context-to-parameter conversion, drawing a connection to implicit gradient-based adaptation. To resolve the *supervision-signal* gap, we introduce **State-Alignment Distillation** to align intermediate feature distributions between the injected model and the full-context oracle, improving behavioral fidelity beyond output matching while keeping memory footprint small.

We evaluate SADA on long-context modeling, ICL, and downstream benchmarks. SADA consistently outperforms StreamAdapter and exceeds GenerativeAdapter where reported, while also reducing memory and latency compared to ICL and lowering training cost relative to fine-tuning. We also characterize a clear task boundary: SADA is robust on **concept-oriented tasks** via pattern compression, whereas entity-intensive tasks that demand precise factual recall are better served by retrieval or KV cache compression methods (Zhang et al., 2023; Li et al., 2024b), suggesting that parameterized mapping acts as lossy compression that attenuates detailed entity bindings.

The main contributions of this paper are summarized as follows:

- **SADA Framework:** We propose a parameterized knowledge injection framework featuring a Mapping Module. This module dynamically converts the internal representations of context prompts into model parameters, bridging the gap between ICL flexibility and FT efficiency.
- **Theoretical Foundation & Alignment:** We provide theoretical motivation linking attention outputs to gradient-based adaptation, offering a principled foundation for feature selection in context-to-parameter mapping, and introduce a projection-based state-alignment strategy that better preserves internal reasoning trajectories.
- **Extensive Validation:** Empirical results show that SADA outperforms *StreamAdapter* and *GenerativeAdapter* on long-context modeling and summarization, while improving inference efficiency (memory and latency) relative to standard ICL and reducing training resource consumption compared to fine-tuning.

2 Related Work

Efficient Long-Context Modeling. Long-context ICL can be slow and memory-intensive, so prior work mainly follows two directions: modifying attention and compressing context. Architectural approaches such as sparse attention (Child et al., 2019) and sliding windows (e.g., *StreamingLLM* (Xiao et al., 2024)) reduce compute but may weaken global awareness. Compression methods such as *LLMLingua* (Jiang et al., 2023) and *SnapKV* (Li et al., 2024b) remove tokens or KV pairs to fit a budget, which may drop information. **In contrast**, SADA avoids

growing the KV cache by compressing history into dynamically generated parameters.

Parameter-Efficient Fine-Tuning (PEFT). PEFT adapts frozen LLMs via lightweight modules (Han et al., 2024), spanning addition-based Adapters (Houlsby et al., 2019), prompt-based Prefix-Tuning (Li and Liang, 2021), and reparameterization methods like LoRA (Hu et al., 2022). However, these approaches yield static weights that remain fixed after training. **SADA distinguishes itself** by functioning as a hypernetwork that dynamically generates instance-specific parameters from the input stream, combining PEFT's efficiency with the real-time adaptability of In-Context Learning.

Context-to-Parameter Mapping. The most relevant prior works are hypernetwork-based methods like *StreamAdapter* (Muhtar et al., 2024) and *GenerativeAdapter* (Chen et al., 2025b), which map context directly to adapter weights. Unlike these approaches, which typically rely on heuristic feature sources and logit-level supervision, **SADA distinguishes itself** by grounding the mapping in *attention-output* features and enforcing *layer-wise state alignment* via State-Aligned Distillation.

3 Methodology

In this section, we formally introduce **SADA**, a framework designed to bridge the gap between transient context prompting and static parameter fine-tuning. We begin by analyzing the mechanics of ICL through the lens of gradient updates, which provides the theoretical grounding for our architecture. Subsequently, we detail the SADA mechanism, structured around state-space modeling (Dao and Gu, 2024), and conclude with our projection-based distillation training objective. The exposition mirrors the two coarse-grained gaps discussed earlier: we first justify the feature source for context-to-parameter conversion, then describe how to align internal states rather than only matching output logits.

3.1 Motivation: ICL as Layer-wise Implicit Gradient Updates

To motivate a module that maps context into parameters, we interpret in-context information as inducing *layer-wise, first-order* effective weight updates. We deliberately restrict the argument to a single-layer linearization: it provides the right intuition without assuming global linearity across depth.

Layer-wise view. Let a Gated MLP layer be

$$\mathbf{y} = \mathbf{W}_{down}(\phi(\mathbf{W}_{gate}\mathbf{x}) \odot (\mathbf{W}_{up}\mathbf{x})) \,. \quad (1)$$

In ICL, attention retrieves contextual information and adds it to the residual stream,

$$\mathbf{x}' = \mathbf{x} + \boldsymbol{\delta}_{ctx}. \quad (2)$$

A first-order expansion around $\mathbf{x}$ shows that this input shift is equivalent to *low-rank* perturbations of the layer parameters, i.e., a LoRA-like update with a shared right factor $\boldsymbol{\delta}_{ctx}$:

$$\mathbf{W}' \approx \mathbf{W} + \mathbf{U}(\mathbf{x})\,\boldsymbol{\delta}_{ctx}^{\top}, \quad (3)$$

where $\mathbf{U}(\mathbf{x})$ is an input-dependent factor determined by the local Jacobian (full derivation in Appendix A). This connects ICL to an *implicit gradient-style* modulation: context specifies the update direction while the current state $\mathbf{x}$ determines the local scaling.

Why attention output features? We take $\boldsymbol{\delta}_{ctx}$ as the attention output, $\boldsymbol{\delta}_{ctx} = \mathrm{softmax}(QK^{\top})V$. It is naturally aligned with $\mathbf{x}$ via queries, normalized by softmax, and aggregated across heads, yielding a stable signal that directly reflects how context modifies the residual stream. Using the attention output as input to our mapping module therefore provides a principled and effective feature for generating $\Delta\mathbf{W}$ (Table 4).

Implication. We thus model "memory injection" as dynamically generating layer-specific low-rank updates $\Delta\mathbf{W}$ that modulate both content transformation and routing. The main text keeps only this first-order layer-wise intuition; deeper nonlinear interactions are beyond our scope and deferred to future work.

3.2 The SADA Architecture

SADA is an auxiliary memory mechanism that summarizes *evicted tokens* (history) and uses the resulting state to modulate the processing of *local tokens* (current context). At a high level, SADA implements: *compress(history)* $\rightarrow$ *evolve(state)* $\rightarrow$ *inject a low-rank update into selected frozen layers*. Figure 2 illustrates the training signals used to supervise these dynamically generated updates.

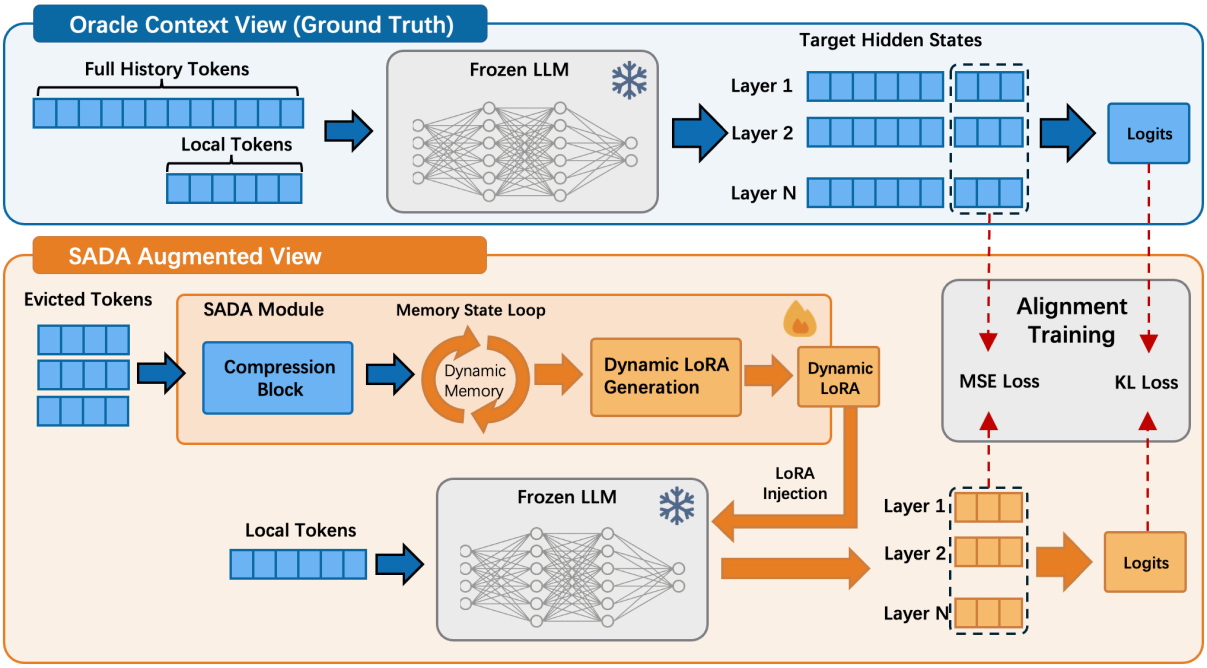


Figure 2: Training strategy for SADA showing distillation signals supervising dynamically generated weight updates.

3.2.1 Memory Encoder: Compression and State Evolution

Let $\mathbf{X}_{hist} \in \mathbb{R}^{L\times d}$ denote evicted tokens, i.e., the historical *attention outputs* (layer-wise residual features) produced by the frozen LLM at the corresponding layer. We split $\mathbf{X}_{hist}$ into chunks $\{\mathbf{X}_k\}_{k=1}^{K}$ of size C. For each chunk, we extract a fixed-size semantic summary using N learnable queries $\mathbf{Q}_{meta}$ via intra-chunk cross attention.

Chunk projection.

$$\mathbf{K}_k = \mathbf{X}_k\mathbf{W}_K, \quad \mathbf{V}_k = \mathbf{X}_k\mathbf{W}_V, \tag{4}$$

Compression.

$$\mathbf{U}_k = \mathrm{CrossAttn}(\mathbf{Q}_{meta}, \mathbf{K}_k, \mathbf{V}_k) \tag{5}$$

where $\mathbf{W}_K, \mathbf{W}_V$ are standard projections.

State evolution. Inspired by the *selective state update* idea in Mamba-2 (Dao and Gu, 2024), we maintain continuity across chunks with a gated recurrent update over meta memory states $\mathbf{M}_k \in \mathbb{R}^{N\times d_{state}}$:

$$\begin{aligned}\mathbf{M}_k &= \gamma_k \odot \mathbf{M}_{k-1} + \mathbf{U}_k,\\ \gamma_k &= \sigma(\mathbf{U}_k\mathbf{W}_\gamma + \mathbf{b}_\gamma)^{1/\tau},\end{aligned} \tag{6}$$

where γ_k is a decay gate, τ controls retention, and $\odot$ is element-wise multiplication. The resulting $\mathbf{M}_k$ serves as a compact, evolving representation of the evicted history.

3.2.2 Dynamic Injection via Meta-LoRA

We inject the memory state into a frozen LLM by generating a context-conditioned low-rank update for selected linear layers (e.g., the attention output projection). We use the row-vector convention: for a token representation $\mathbf{x}_t \in \mathbb{R}^{1\times d}$ and a linear weight $\mathbf{W}_0 \in \mathbb{R}^{d\times d_{out}}$, the layer output is $\mathbf{y}_t = \mathbf{x}_t\mathbf{W}_0$. Under this convention, we adopt the LoRA form $\Delta\mathbf{W}_k = \mathbf{A}\mathbf{B}_k$, where $\mathbf{A}$ is static and $\mathbf{B}_k$ is generated from memory.

Specifically, we set the static down-projection $\mathbf{A} \triangleq \mathbf{W}_{MetaA} \in \mathbb{R}^{d\times d_{state}}$ and produce a dynamic up-projection from the current memory state $\mathbf{M}_k \in \mathbb{R}^{N\times d_{state}}$:

$$\mathbf{B}_k = \mathbf{M}_k^\top \mathbf{W}_{MetaB} \in \mathbb{R}^{d_{state}\times d_{out}}, \tag{7}$$

where $\mathbf{W}_{MetaB} \in \mathbb{R}^{N\times d_{out}}$ maps memory slots to the layer output space. The forward pass is:

$$\begin{aligned}\mathbf{y}_t &= \mathbf{x}_t\mathbf{W}_0 + \alpha\big(\mathbf{x}_t\mathbf{W}_{MetaA}\big)\mathbf{B}_k\\ &= \mathbf{x}_t\mathbf{W}_0 + \alpha\,\mathbf{x}_t\underbrace{\big(\mathbf{W}_{MetaA}\mathbf{B}_k\big)}_{\Delta\mathbf{W}_k},\end{aligned} \tag{8}$$

where α is a scaling factor and $\Delta\mathbf{W}_k$ is a rank-d_{state} update conditioned on the compressed history $\mathbf{M}_k$.

Remark. This design separates (i) memory formation in $\mathbf{M}_k$ from (ii) parameter modulation via $\Delta\mathbf{W}_k$, enabling efficient adaptation without storing the full historical context.

3.3 Training Strategy

The overall training pipeline is illustrated in Fig. 2, showing how the mapping module aligns with oracle states from full-context processing. Our method follows a two-stage recipe. In **Stage I (General Distillation)**, we use large-scale pre-training data with a sliding-window scheme and optimize with MSE on hidden states plus KL divergence on logits. In **Stage II (Task Alignment)**, we distill long-context hidden states while optimizing the target output with LM_Loss. Accordingly, SWSD is used in Stage I for long-context distillation, while TAD is used in Stage II for task alignment.

3.3.1 Stage I (General Distillation): Sliding Window State Distillation (SWSD)

For general language generation and long-context processing, we employ SWSD to train the Mapping Module to compress historical context into weight space. For a sequence X of length L, we utilize a window size C' and a stride size Δ. The process follows an iterative cycle:

- **Context Eviction and Encoding**: In each step, we evict the earliest Δ tokens and encode their attention-output hidden states (i.e., the attention-induced residual increment) to predict parameter updates ΔW.
- **Dual-Path Forward Pass**: We run two forward passes for the incoming Δ tokens: a *Teacher Path* with the full-context KV cache and a *Student Path* using ΔW with a truncated context.
- **Gradient Accumulation**: The loss aligns the student's hidden states and logits with the teacher and is accumulated over X before updating SADA parameters.

We optimize the mapping module with hidden-state regression and logit alignment:

$$\mathcal{L}_{\text{stage1}} = \mathcal{L}_{\text{MSE}} + \mathcal{L}_{\text{KL}} \tag{9}$$

where $\mathcal{L}_{\text{MSE}}$ is the mean-squared error averaged across layers,

$$\mathcal{L}_{\text{MSE}} = \frac{1}{|\mathcal{S}|} \sum_{\ell \in \mathcal{S}} \left\| H_\ell^t - H_\ell^s \right\|_2^2, \tag{10}$$

and $\mathcal{L}_{\text{KL}} = \text{KL}(p^t \,\|\, p^s)$ aligns teacher and student output distributions.

3.3.2 Stage II: Task Alignment Distillation (TAD)

To adapt SADA for a broader range of post-training tasks, we employ TAD with a 2-forward-1-backward strategy. For each sample, we form a task-specific context (e.g., exemplars, instructions, or prompts) and a target sequence to be predicted.

1. **First Forward (Oracle States, no-grad)**: We run the frozen base model on the *full* input (context + target) *without gradient* to obtain the teacher cache and oracle hidden states at the target positions.
2. **Second Forward (Parameter Injection)**: The SADA mapping module encodes the teacher cache (e.g., teacher attention outputs) to predict parameter updates ΔW. We then run the student forward using the truncated explicit context with injected ΔW, and process the target sequence.
3. **Task Alignment (single backward)**: We compute a combined loss consisting of (i) hidden-state MSE against the oracle (averaged across selected layers and target positions) and (ii) the language-modeling loss on the target tokens, and backpropagate only through the mapping/SADA module.

In this stage, we combine hidden-state distillation with the language modeling objective:

$$\mathcal{L}_{\text{stage2}} = \mathcal{L}_{\text{LM}} + \mathcal{L}_{\text{MSE}}. \tag{11}$$

4 Experiment

4.1 Setup

We evaluate SADA across two primary dimensions: long-context language modeling and post-training alignment on downstream NLU and summarization tasks.

Baselines. We compare against five representative approaches: **ICL** (full prompt + full KV cache at inference, used as a full-context reference), **Fine-tuning** (SFT on task data), **SnapKV** (KV-cache pruning/compression), and two context-to-parameter hypernetwork baselines, **StreamAdapter** and **GenerativeAdapter** (see related work for details).

Models. We utilize **Qwen2.5-1.5B-Instruct** and **Qwen2.5-3B-Instruct** as the foundation models (Qwen et al., 2025).

Table 1: Comparison on PG19 test set across varying maximum context lengths

	Method	2K	4K	8K	16K	32K	64K	128K
Qwen-2.5-1.5B	Sliding Window	13.97	13.72	13.57	13.59	13.67	13.72	13.72
	StreamAdapter	13.99	13.69	13.49	13.42	13.49	13.31	13.05
	GenerativeAdapter	14.11	13.82	13.63	13.55	13.59	13.39	13.09
	Ours	**13.72**	**13.45**	**13.27**	**13.18**	**13.20**	**12.98**	**12.65**
Qwen-2.5-3B	Sliding Window	12.09	11.86	11.74	11.76	11.84	11.83	11.82
	StreamAdapter	12.12	11.87	11.72	11.68	11.74	11.64	11.63
	GenerativeAdapter	12.07	11.82	11.68	11.64	11.67	11.56	11.55
	Ours	**11.98**	**11.74**	**11.60**	**11.55**	**11.58**	**11.45**	**11.42**

Table 2: Main results on NLU (Seen/Unseen; accuracy) and long-form summarization (GovReport and QMSum; ROUGE) with Qwen2.5-Instruct backbones. ICL uses the full prompt and serves as an upper bound.

	Qwen2.5-1.5B-Instruct				**Qwen2.5-3B-Instruct**			
	NLU		**Summarization**		**NLU**		**Summarization**	
Method	**Seen**	**Unseen**	**GovReport**	**QMSum**	**Seen**	**Unseen**	**GovReport**	**QMSum**
ICL	80.22	57.48	25.98	21.62	79.31	63.68	31.97	24.31
Fine-tuning	56.92	54.62	7.87	17.77	77.34	61.75	8.21	14.86
SnapKV	77.07	55.73	10.72	14.36	77.92	62.04	12.93	14.78
StreamAdapter	85.01	55.62	18.03	11.08	87.25	60.43	22.32	19.17
GenerativeAdapter	85.08	56.11	18.17	12.12	86.60	61.69	21.14	17.82
Ours	**86.50**	**56.68**	**19.07**	**18.64**	**88.18**	**63.33**	**24.57**	**19.68**

Datasets. For long-context capabilities, we evaluate language modeling perplexity using the **PG19** dataset(Rae et al., 2019). For post-training task alignment, we evaluate on a mixture of **NLU** benchmarks and long-form **summarization** datasets (GovReport (Huang et al., 2021) and QMSum (Zhong et al., 2021)), and use a Seen/Unseen split to measure in- vs. out-of-distribution generalization.

4.2 Long-Context Modeling (PG19).

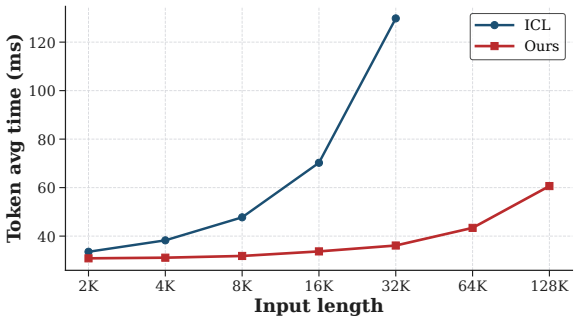
Training Details. We set the training sequence length to 8,192 and employed a sliding-window approach for progressive distillation, matching Stage I distillation settings. We evaluated the perplexity (PPL) of our method against *StreamAdapter*, *GenerativeAdapter*, and baseline sliding-window approaches using the PG19 dataset.

Main Results. We evaluate PG19 test perplexity across context lengths ranging from 2K to 128K, using stride-averaged PPL to compare our method against Sliding Window, *StreamAdapter*, and *GenerativeAdapter*. SADA achieves the lowest PPL across all tested lengths, consistently outperforming sliding-window and state-of-the-art adapter baselines at both model scales, particularly in extended contexts. Notably, these improvements persist well beyond the training-context regime, indicating that state-aligned distillation supports reliable extrapolation to much longer contexts without retaining the full KV cache.

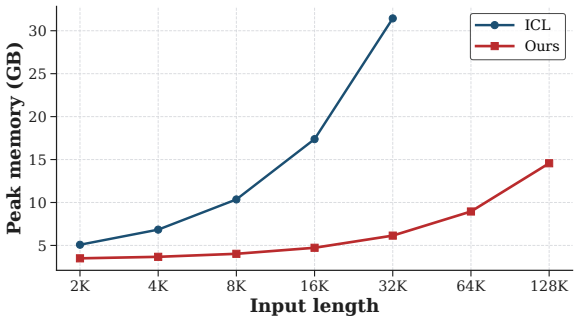
Efficiency Test. To quantify inference-time efficiency, we measure (i) average per-token latency and (ii) peak GPU memory on **Qwen2.5-1.5B-Instruct** while sweeping input length from 2K to 128K on PG19. As shown in Fig. 3, SADA reduces both latency and memory relative to prompt-based ICL by replacing long KV-cache retention with parameterized compression of the historical context. We report the average per-token latency measured during the *decode* stage.

4.3 Post-Training Alignment on NLU and Summarization.

Training Details. We perform post-training alignment using a mixture of NLU datasets: BoolQ (Clark et al., 2019), CoPA (Melissa et al., 2011), SST-2 (Richard et al., 2013), CB (De Marneffe et al., 2019), and RTE (Bentivogli et al., 2009). To analyze sensitivity, we scale the few-shot context from 4 to 20 instances; the final two serve



(a) Average per-token latency (ms) on PG19 across input lengths (Qwen2.5-1.5B).



(b) Peak GPU memory (GB) under the same sweep, highlighting KV-cache savings.

Figure 3: Efficiency evaluation on PG19 contrasting SADA and ICL: (a) average per-token latency and (b) peak GPU memory across input lengths.

as the in-context prefix, while others are compressed into model parameters. Training is conducted over 3 epochs with a batch size of 64 and a learning rate of 6×10^{-5} on eight 40GB GPUs. We evaluate performance on **Seen** (in-distribution) and **Unseen** (OOD) benchmarks, the latter including ARC (Clark et al., 2018), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), and WinoGrande (Sun and Emami, 2024). Table 2 reports 20-shot results by default.

Main Results. Table 2 reports accuracy/ROUGE on Seen/Unseen NLU and long-form summarization under an ICL-style protocol, where a fixed set of demonstrations is provided and only the most recent few are kept as an explicit prefix. On Seen NLU, SADA performs best for both backbones, consistently improving over the strongest mapping baseline by a clear margin, while remaining close to full-prompt ICL on Unseen benchmarks. On summarization, SADA yields substantial gains over adapter baselines, but still falls short of full-prompt ICL. This pattern indicates that parameterized compression transfers in-distribution task behavior effectively, while explicit prompts retain an advantage under distribution shift and evidence-heavy generation, which we further probe in the QA/summarization analysis and the alignment ablations.

Discussion. Across model sizes and shot budgets (Fig. 4), SADA fits in-distribution patterns strongly (Seen) while narrowing—but not fully closing—the gap to full-prompt ICL under distribution shift (Unseen). This trade-off is consistent with replacing explicit context retention by parameterized compression.

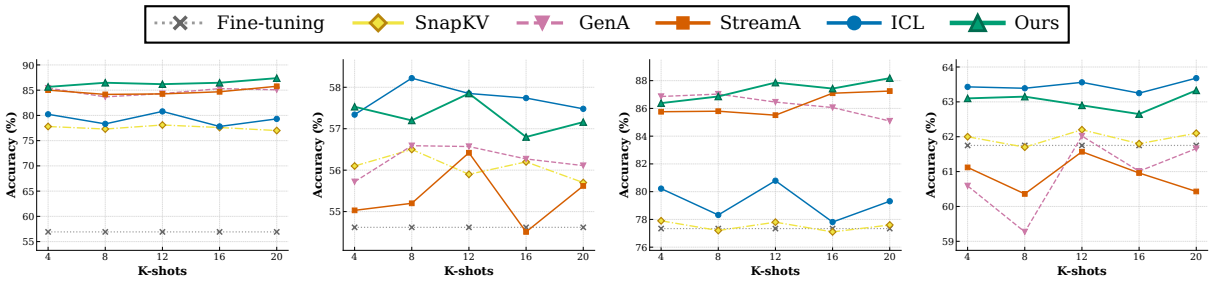
4.4 Ablation Studies

Conceptual vs. Detailed Memory. Building on the PG19 checkpoint, we further fine-tuned on downstream Sum and QA tasks and observed a divergence: As shown in Tab. 3, QA drops markedly while summarization degrades more gently. This suggests SADA preserves high-level semantics but struggles with fine-grained retrieval compared to prompt-only ICL. GenerativeAdapter yields stronger QA than SADA but remains far below ICL, whereas SADA achieves the best summarization among adapter-based baselines, highlighting a trade-off between entity recall and semantic compression. A plausible explanation is that low-rank parameter updates retain global topical structure and discourse flow, but weaken entity bindings and exact spans needed for QA, especially on multi-hop settings requiring precise entity chains and temporal relations. This suggests pairing SADA with retrieval/KV retention for entity-centric queries, while remaining effective for concept-oriented tasks that benefit from semantic compression.

Table 3: Qwen2.5-1.5B-Instruct on QA (HotpotQA (Yang et al., 2018), 2WikiMQA (Ho et al., 2020)) and summarization (GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021)).

	QA		Sum	
Qwen2.5-1.5B	**HotpotQA**	**2WikiMQA**	**GovReport**	**QMSum**
ICL	42.41	35.58	25.98	21.62
SnapKV	**38.51**	**29.47**	10.72	14.36
StreamAdapter	20.28	19.13	18.03	11.08
GenerativeAdapter	28.19	25.19	18.17	12.12
Ours	23.86	26.15	**19.07**	**18.64**

Impact of Loss Components. Our task-alignment stage (Stage II) couples language



(a) Qwen2.5-1.5B seen. (b) Qwen2.5-1.5B unseen. (c) Qwen2.5-3B seen. (d) Qwen2.5-3B unseen.

Figure 4: ICL evaluation results for SADA across model sizes and settings.

modeling supervision with hidden-state distillation: (i) hidden-state alignment ($\mathcal{L}_{\text{MSE}}$) that constrains the internal reasoning trajectory, and (ii) LM_Loss ($\mathcal{L}_{\text{LM}}$) that ensures correct target generation. To focus on the alignment stage, we keep Stage I fixed and run an ICL-based ablation protocol on **Qwen2.5-1.5B-Instruct** (TAD setting): we fix the prompt construction and compression policy (retain the last two demonstrations as the explicit prefix and compress the remaining exemplars into the generated parameters), and evaluate under a **20-shot** setting on the Seen/Unseen benchmark split described earlier. We vary only the alignment-stage loss while keeping the architecture and training hyperparameters unchanged: *(1)* **Full** objective $\mathcal{L}_{\text{LM}} + \mu\mathcal{L}_{\text{MSE}}$; *(2)* **w/o** $\mathcal{L}_{\text{MSE}}$, which removes state trajectory supervision and tests whether task supervision alone is sufficient; *(3)* **w/o** $\mathcal{L}_{\text{LM}}$, which removes task supervision and tests whether distillation alone can recover task accuracy.

Both loss terms matter, but the LM loss is more critical: removing $\mathcal{L}_{\text{LM}}$ causes a large drop, while removing $\mathcal{L}_{\text{MSE}}$ yields a smaller decline. LM supervision anchors task performance, and state matching stabilizes behavior transfer under compression.

Input Feature Sources. We additionally ablate the *feature source* fed into the Mapping Module, since our method hinges on which representation is most "convertible" into effective parameter updates. Under the same TAD setup as above (1.5B, 20-shot, identical loss weights), we replace the mapping input with: *(1)* **Token embeddings** (pre-Transformer), *(2)* **Attention input** (the normalized residual stream entering the attention block, i.e., the features used to form $Q/K/V$), *(3)* **Attention output** (post-attention residual), *(4)* **Post-MLP activations**, and *(5)* **KV features** (aggregated keys/values from the teacher cache).

Table 4: Proposed ICL ablations on **Qwen2.5-1.5B-Instruct** under a 20-shot setting (Seen/Unseen split as in Fig. 4).

Variant	Seen Acc.	Unseen Acc.	Avg.
Impact of Loss Components			
Full ($\mathcal{L}_{\text{LM}} + \mathcal{L}_{\text{MSE}}$)	**87.38**	**57.16**	**72.27**
w/o $\mathcal{L}_{\text{MSE}}$	85.84	56.32	71.08
w/o $\mathcal{L}_{\text{LM}}$	79.90	55.46	67.68
Input Feature Sources			
Attention output	**87.38**	**57.16**	**72.27**
Attention input	84.20	53.45	68.83
KV features (K/V aggregated)	86.34	56.44	71.19
Post-MLP activations	85.28	56.47	70.88
Token embeddings	85.13	55.77	70.45

This validates our claim in Section 3.1 that the attention *output* is the optimal interface for parameter modulation. Empirically, we expect embeddings to underperform due to their syntactic nature and post-MLP features to suffer from nonlinear distortion. KV features, while competitive, likely lag due to head-wise fragmentation and formatting sensitivity.

5 Conclusion

We introduced SADA, a framework that maps context into low-rank parameter updates via attention-output features and state-aligned distillation. Across PG19 and ICL benchmarks, SADA improves long-context modeling, delivers strong in-distribution accuracy, outperforming StreamAdapter and GenerativeAdapter, and reduces latency and memory compared to prompt ICL. Ablations clarify the roles of LM supervision and state matching, while analysis shows remaining gaps on entity-heavy and out-of-distribution settings.

6 Limitations

Lossy Compression. Transforming context into parameters acts as a lossy compression mechanism. While effective for high-level semantic tasks like summarization, it degrades performance on entity-intensive tasks (e.g., QA) that require precise factual recall, where explicit KV caches remain superior.

OOD Generalization Gap. Our experiments show a slight performance gap on "Unseen" benchmarks compared to standard ICL. Explicit context prompting proves more robust for novel task distributions not encountered during the distillation phase, whereas parameterized mapping may slightly overfit to training distributions.

Training Overhead. Unlike training-free ICL, SADA incurs computational costs to train the auxiliary Mapping Module via the two-stage process. Although cheaper than full fine-tuning, this requirement prevents instant, zero-cost deployment for arbitrary contexts without prior adaptation.

7 Ethical considerations

This paper presents work whose goal is to advance the field of Machine Learning. Since our work concerns context-to-parameter mapping and involves lossy compression mechanisms described in Section 6, which may impact factual recall in entity-intensive tasks, we expect our work to be used under the guidance of human values.

References

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth PASCAL recognizing textual entailment challenge.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Haolong Chen, Hanzhi Chen, Zijian Zhao, Kaifeng Han, Guangxu Zhu, Yichen Zhao, Ying Du, Wei Xu, and Qingjiang Shi. 2025a. An overview of domain-specific foundation model: key technologies, applications and challenges. *Science China Information Sciences*, 69(1).

Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Benjamin Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. 2025b. Generative adapter: Contextualizing language models in parameters with a single forward pass. In *The Thirteenth International Conference on Learning Representations*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *Preprint*, arXiv:1803.05457.

Tri Dao and Albert Gu. 2024. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *CoRR*, abs/2405.21060.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The CommitmentBank: Investigating projection in naturally occurring discourse.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, Miami, Florida, USA. Association for Computational Linguistics.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Neil Houlsby, Andrei Giouvanos, Zornitsa Kozareva, Möhrmann Wei, Michael Hall, Romal Thoppilan, and Google Sung. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of

large language models. In *International Conference on Learning Representations*.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online. Association for Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Hongyu Li, Liang Ding, Meng Fang, and Dacheng Tao. 2024a. Revisiting catastrophic forgetting in large language model tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4297–4308, Miami, Florida, USA. Association for Computational Linguistics.

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

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024b. SnapKV: LLM knows what you are looking for before generation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Kangtao Lv, Haibin Chen, Yujin Yuan, Langming Liu, Shilei Liu, Yongwei Wang, Wenbo Su, and Bo Zheng. 2025. How to inject knowledge efficiently? knowledge infusion scaling law for pretraining large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 26204–26219, Suzhou, China. Association for Computational Linguistics.

Roemmele Melissa, Bejan Cosmin Adrian, and Gordon Andrew S. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium. Association for Computational Linguistics.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

Dilxat Muhtar, Yelong Shen, Yaming Yang, Xiaodong Liu, Yadong Lu, Jianfeng Liu, Yuefeng Zhan, Hao Sun, Weiwei Deng, Feng Sun, and 1 others. 2024. Streamadapter: Efficient test time adaptation from contextual streams. *arXiv preprint arXiv:2411.09289*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *Preprint*, arXiv:1911.05507.

Socher Richard, Perelygin Alex, Jean Wu, Chuang Jason, Manning Christopher D., Ng Andrew, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Jing Han Sun and Ali Emami. 2024. EvoGrad: A dynamic take on the Winograd schema challenge with human adversaries. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 6701–6716, Torino, Italia. ELRA and ICCL.

Gido M van de Ven, Nicholas Soures, and Dhireesha Kudithipudi. 2024. Continual learning and catastrophic forgetting. *arXiv preprint arXiv:2403.05175*.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. 2024. Efficient large language models: A survey. *Transactions on Machine Learning Research*. Survey Certification.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can

a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021. QMSum: A new benchmark for query-based multi-domain meeting summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5905–5921, Online. Association for Computational Linguistics.

A Layer-wise Derivation for Input Shift

This appendix provides the first-order, single-layer derivation sketched in Section 3.1. We linearize a gated MLP layer and express the impact of an attention-derived input shift as low-rank weight perturbations.

Setup. For a gated MLP layer,

$$\mathbf{y} = \mathbf{W}_{down}\left(\phi(\mathbf{W}_{gate}\mathbf{x}) \odot (\mathbf{W}_{up}\mathbf{x})\right), \quad (12)$$

where ϕ is the activation (e.g., SiLU) and $\odot$ is the Hadamard product. Under ICL, the effective input is $\mathbf{x}' = \mathbf{x} + \boldsymbol{\delta}_{ctx}$ with $\boldsymbol{\delta}_{ctx}$ produced by attention.

First-order expansion. Expanding around $\mathbf{x}$ (ignoring $\mathcal{O}(\|\boldsymbol{\delta}_{ctx}\|^2)$):

$$\begin{aligned}\mathbf{h}(\mathbf{x}+\boldsymbol{\delta}_{ctx}) \approx \phi(\mathbf{W}_{gate}\mathbf{x}) \odot (\mathbf{W}_{up}\mathbf{x}) \\ + \left[\phi'(\mathbf{z}_g) \odot (\mathbf{W}_{gate}\boldsymbol{\delta}_{ctx})\right] \odot (\mathbf{W}_{up}\mathbf{x}) \\ + \phi(\mathbf{z}_g) \odot (\mathbf{W}_{up}\boldsymbol{\delta}_{ctx}),\end{aligned} \quad (13)$$

where $\mathbf{z}_g = \mathbf{W}_{gate}\mathbf{x}$. Propagating to the output,

$$\Delta\mathbf{y} \approx \mathbf{W}_{down}\left(\Delta\mathbf{h}_{gate} + \Delta\mathbf{h}_{val}\right). \quad (14)$$

Each term is re-expressible as a rank-1 perturbation:

$$\Delta\mathbf{h}_{val} = \left[\mathrm{diag}(\phi(\mathbf{z}_g))\mathbf{W}_{up}\right]\boldsymbol{\delta}_{ctx}, \quad (15)$$

$$\Delta\mathbf{h}_{gate} = \left[\mathrm{diag}(\phi'(\mathbf{z}_g) \odot \mathbf{W}_{up}\mathbf{x})\mathbf{W}_{gate}\right]\boldsymbol{\delta}_{ctx}. \quad (16)$$

Thus

$$\begin{aligned}\mathbf{W}'_{up} &= \mathbf{W}_{up} + \mathbf{U}_{up}(\mathbf{x})\,\boldsymbol{\delta}_{ctx}^{\top}, \\ \mathbf{W}'_{gate} &= \mathbf{W}_{gate} + \mathbf{U}_{gate}(\mathbf{x})\,\boldsymbol{\delta}_{ctx}^{\top},\end{aligned} \quad (17)$$

with $\mathbf{U}_{up}, \mathbf{U}_{gate}$ determined by $\mathbf{W}_{down}$, ϕ, and ϕ'. This is a layer-wise, first-order approximation; multi-layer nonlinear accumulation is not assumed linear.

B Hyperparameter Settings

B.1 SADA Module Hyperparameters

Chunk size. We set the chunk size $C = 64$, matching the chunking operator used in Section 3.3 and the definition in Section 3.2.1 where evicted tokens are partitioned into fixed-size segments.

Learnable queries. We use $N = 16$ learnable queries ($\mathbf{Q}_{meta}$ in Section 3.3) for intra-chunk cross attention, yielding a fixed-size representation per chunk.

Low-rank head dimension. We set the per-head low-rank dimension $d_r = 16$ (corresponding to the low-rank projection used to form the meta key/value features). The resulting memory width is $d_{state} = H \cdot d_r$, where H is the number of attention heads of the backbone model.

B.2 Training Hyperparameters

Optimization. Unless stated otherwise, we train with AdamW (betas $(0.9, 0.95)$, $\epsilon = 10^{-6}$, weight decay 0) and a cosine learning-rate schedule with linear warmup (warmup steps $= 100$; total steps determined by the number of update steps). We clip gradients with a global norm of 1.0 and train in BF16 with FlashAttention-2 enabled.

Stage I (PG19 / SWSD). We train for 1 epoch with per-device batch size 1 and gradient accumulation steps 8 on 8 GPUs (global batch size 64). The learning rate is 6×10^{-5}.

Stage II (ICL/NLU / TAD). Starting from the PG19 checkpoint, we train for 3 epochs using the same per-device batch size 1, gradient accumulation steps 8, and 8 GPUs (global batch size 64). The learning rate is 6×10^{-5}.

C Dialogue Templates

This appendix reports the chat-style dialogue templates used in our ICL benchmarks. We group templates by Seen vs. Unseen tasks. Placeholders are denoted with angle brackets (e.g., `<premise>`).

C.1 Seen Tasks

User: `<sentence>`
Question: Is this sentence positive or negative?
Assistant: `positive | negative`

User: `<premise> therefore | because`
Assistant: `<choice>`

User: `<passage>`
Question: `<question?>`
Assistant: `yes | no`

User: `<premise>`
Question: `<hypothesis>`. True, False, or Neither?
Assistant: `True | False | Neither`

User: `<premise>`
Question: `<hypothesis>` True or False?
Assistant: `True | False`

C.2 Unseen Tasks

User: `<question>`
Please answer the question.
Assistant: `<option label>`

User: `<question>`
Please answer the question.
Assistant: `<option label>`

User: `<context>`
Assistant: `<option label>`

User: Answer the question: `<goal>`.
Assistant: `<option label>`

User: `<sentence with blank as _>`
Options: `A. <choice A>`
`B. <choice B>`
`...`
Assistant: `<label>. <choice text>`

C.3 QA Tasks

User: `<passages>`
Answer the question based on the given passages.
Only give me the answer.
Question: `<question>`
Assistant: `<answer>`

User: `<passages>`
Answer the question based on the given passages.
Only give me the answer.
Question: `<question>`
Assistant: `<answer>`

C.4 Summarization Tasks

User: `<report>`
Write a one-page summary of the report.
Assistant: `<summary>`

User: `<transcript>`
Answer the query in one or more sentences.
Query: `<query>`
Assistant: `<answer>`