Reinforced Fast Weights via Next-Sequence Prediction

Hee Seung Hwang[1*] **Xindi Wu**[1*] **Sanghyuk Chun**[1] **Olga Russakovsky**[1] **Zhiwei Deng**[2]

[1]Princeton University [2]Google DeepMind

Abstract

Fast weight architectures offer a promising alternative to standard transformers for long-context modeling by replacing the KV-cache with a recurrently updated fixed-size memory. Despite this architectural shift, they are typically trained with the same next-token prediction (NTP) objective as standard transformers. This creates a mismatch: fast weight models rely on an evolving memory state to support future predictions, while NTP provides only token-level supervision for the immediate next token. We address this mismatch with next-sequence prediction (NSP), a sequence-level extension of NTP that trains models to produce coherent multi-token continuations from their recurrent memory state. To optimize this sequence-level objective, we propose REFINE (**Re**inforced **F**ast we**I**ghts via **N**ext s**E**quence prediction). REFINE selects informative token positions based on prediction entropy, generates multi-token rollouts, assigns sequence-level rewards, and optimizes the model with Group Relative Policy Optimization (GRPO). REFINE is applicable throughout the training lifecycle of pre-trained language models: mid-training, post-training, and test-time training. Experiments on LaCT-760M and DeltaNet-1.3B show that REFINE consistently outperforms NTP-based supervised fine-tuning in needle-in-a-haystack retrieval, long-context question answering, and diverse tasks in LongBench. REFINE provides an effective and versatile framework for improving long-context modeling in fast weight architectures. Code is available at `https://github.com/princetonvisualai/ReFINE`.

1 Introduction

Long-context modeling has become essential for large language models (LLMs). Tasks such as long-document understanding [1, 2], many-shot in-context learning [3, 4], and code generation [5, 6] require models to extract, store, and reuse information from contexts spanning thousands of tokens [7–9]. While attention-based transformers demonstrate strong performance on these tasks, their computational costs scale quadratically [10] with context length, creating a fundamental bottleneck for both training and inference.

Fast weight architectures offer a promising alternative by addressing the scaling problem through structural changes to the transformer block. Models such as DeltaNet [11], GatedDeltaNet [12], and LaCT [13] replace the KV-cache in standard transformers with a dynamically updated, fixed-size memory that learns input-dependent KV mappings on-the-fly (Fig. 1a). This design enables efficient inference with constant memory overhead [14].

However, compressing past tokens into a bounded memory via recurrent updates poses a unique challenge for fast weight models. Unlike transformers, which retain explicit representations of past tokens in the KV-cache, fast weights continually adapt as new tokens are processed. This recurrent compression makes the memory susceptible to forgetting [15, 16] and error accumulation [17–20], as

[*]Equal contribution

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

small deviations in the autoregressive trajectory can compound over time. This requires the training objective of fast weight models to supervise the long-horizon stability of the memory state [21].

Despite their architectural differences, fast weight models are pre-trained with the same next-token prediction (NTP) objective used for standard transformer LLMs [14, 22, 23, 11–13]. In this work, we argue that NTP is a *suboptimal* objective for fast weight models. NTP provides supervision through teacher-forced ground truth tokens at each position, but does not evaluate the quality of the recurrent memory state after it evolves over multiple steps on the model's own predictions [24].

To improve the adaptive capacity of fast weights over long horizons, we propose the **next-sequence prediction (NSP)** objective as a sequence-level extension of NTP. NSP encourages a model to predict a semantically coherent sequence of future tokens conditioned on a given prefix. This objective evaluates whether the information stored in fast weights enables accurate continuation over multiple steps, providing a more appropriate training signal than standard NTP. However, training with NSP introduces two key challenges: (i) standard cross-entropy loss does not naturally extend to multi-token prediction, and (ii) generating sequences for every prefix is computationally prohibitive. We address these challenges by formulating NSP as a reinforcement learning (RL) problem and propose REFINE (Reinforced Fast Weights via Next Sequence Prediction), a *phase-agnostic* training framework for fast weight models. REFINE focuses on informative regions of the context and optimizes for NSP using policy gradient updates.

In a motivating experiment, we compare how autoregressive generation trajectories diverge from ground truth trajectories in fast weight and transformer models. Fast weight models exhibit substantially larger divergence in output representations than transformers, suggesting that recurrent memory updates may be more sensitive to compounding prediction errors. Training with REFINE reduces this divergence compared to NTP-based SFT, supporting the role of sequence-level supervision in stabilizing long-horizon generation (Fig. 1c).

Our experiments also show that REFINE improves performance in multiple stages of the language model training lifecycle (Tab. 1): (i) *mid-training*, where REFINE reinforces long-context adaptation of fast weights; (ii) *post-training*, where REFINE reinforces task-specific adaptation of fast weights; and (iii) *test-time training*, where REFINE reinforces prompt-specific adaptation of fast weights. Concretely, applying REFINE on LaCT-760M notably improves the average performance on RULER [25] long-context QA tasks by 8.5% (mid-training), 15.3% (post-training), and 9.5% (test-time training), compared to pure NTP training with SFT. For DeltaNet-1.3B, we observe 20.3% (mid-training), 11.0% (post-training), and 15.0% (test-time training) gains with REFINE.

Our contributions are:

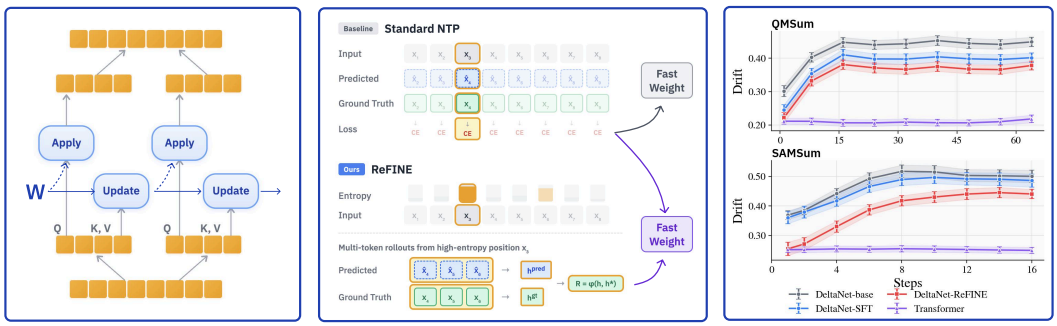
- We introduce the NSP objective for fast weight models to address the lack of sequence-level feedback in standard NTP.
- We propose an RL framework to optimize NSP, combining entropy-based token selection and sequence-level rewards.
- We demonstrate that REFINE improves fast weight models across mid-training, post-training, and test-time training, with 25% and 8% average gains over SFT mid-training on LongBench [8] tasks.

2 Background

Fast Weight Architectures. Fast weight architectures replace the explicit KV-cache in standard transformers with a fixed-size, dynamically updated memory represented by weight matrices. To compress contextual information into the memory, fast weight models recurrently update these weight matrices as new tokens are processed. This perspective connects fast weight models to test-time adaptation [22] and meta-learning [26], as the memory is updated continually and task-agnostically during inference. The update rule for fast weights can be generalized as follows [13]:

$$W_{t+1} \leftarrow W_t - \eta \nabla_{W_t} \ell\big(W_t k_t, v_t\big) \tag{1}$$

where W denotes the fast weight, η is the learning rate, and k_t, v_t are the key, value representations of the input token at position t. This update rule can be viewed as online learning of a key-to-value mapping. The output representation is retrieved from fast weights via an apply operation, *i.e.*,



(a) Fast Weight Architecture (b) Comparison of Standard NTP and ReFINE (c) Output Representation Drift

Figure 1: **(a)** Adapted from Zhang et al. [13]. Fast weight models replace the KV-cache in standard transformers with a recurrent fixed-size memory (W), updated according to Eq. (1). **(b)** Standard NTP provides token-level supervision through cross-entropy loss at each position. REFINE provides sequence-level supervision by optimizing the rewards of multi-token rollouts at high-entropy positions with RL. **(c)** Fast weight models show larger divergence in the last layer's output representations, measured by Euclidean distance between predicted and ground truth response trajectories, than standard transformers. REFINE substantially reduces this divergence compared to SFT. Error bars show $\pm 1\sigma$ across 50 independent generations.

$o_t = W_t q_t$, where o_t, q_t are the token's output and query representations. Fig. 1a illustrates the internal structure of fast weight language models.

Training Phases for Language Models. Pre-trained language models undergo additional training stages that rely on different sources of data and supervision (Tab. 1). We follow the standard taxonomy of three additional training phases: *mid-training*, *post-training*, and *test-time training*.

Mid-training is an extension (or continued version) of pre-training, generally used for the adaptation of a pre-trained model to specific domains or capabilities [27]. In this paper, we apply REFINE to mid-train models on the same training dataset as pre-training to adapt pre-trained models to the NSP objective.

Table 1: **Training Phases for Fast Weight Language Models.** REFINE can be applied across all phases beyond pre-training to improve long-context modeling, using different data sources.

Training Phase	Source of Training Data
Pre-training	General web-scraped data
Mid-training	Similar data to pre-training
Post-training	Task / instruction / preference data
Test-time training	Test data

Post-training fine-tunes pre-trained models to follow instructions and align their responses with human preferences [28–30]. This phase typically involves SFT on task-specific instruction-response pairs, with relatively fewer gradient steps compared to pre-training. We apply REFINE during post-training through the nested learning [31] technique: within each training loop, we first use REFINE to update the model on the instruction prompt alone, and then use SFT to fine-tune the model's final response.

Test-Time Training (TTT) adapts model parameters at inference time using self-supervised objectives to handle distribution shifts from source to target [32–34, 14]. TTT naturally integrates with fast weight architectures by design: each input token updates the fast weights via gradient-based rules, enabling the model to memorize and adapt to the given context on-the-fly [13, 22]. However, fast weight models trained purely with NTP can still struggle on long-context retrieval (e.g., Needle-in-a-Haystack [25]) due to unstable fast weight updates or insufficient long-horizon supervision.

RL for Language Modeling. Recent work has formulated NTP as a reward maximization problem with RL [35, 36]. These methods elicit reasoning traces before predicting the next token and assign rewards based on the similarity between the prediction and the ground truth tokens. However, existing approaches focus on standard transformer LLMs that demonstrate basic reasoning behavior. It is still an open question whether RL can be applied to pre-trained fast weight models that lack such reasoning capabilities. We show that RL can improve fast weight models on long-context tasks without prior instruction tuning. See Appendix §C for more discussion of related work.

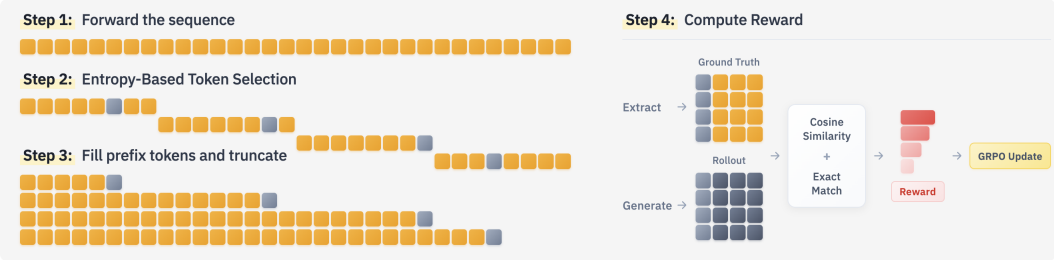


Figure 2: **REFINE.** We forward the sequence through the policy model and compute token-level entropy values, which we use to sample a target token position from each equal-length chunk in the sequence (**Entropy-Based Token Selection**). Prefixes are copied from the original sequence up to each target token, and the policy model predicts continuations (**Rollout Generation**). Reward is computed based on the generated rollouts and ground truth tokens (**Reward Assignment**). Finally, we update the policy model with GRPO (**Optimization with RL**).

3 Method

Our goal is to improve fast weight models in long-context settings via the NSP objective. We present RL as a solution to the lack of sequence-level feedback in SFT.

3.1 From Next-Token to Next-Sequence Prediction

Next-Token Prediction (NTP). Standard language model pre-training involves minimizing the cross-entropy (CE) loss of the NTP objective. Given an input sequence $S = (x_1, \ldots, x_T)$, the CE loss is computed using the predicted probability of the ground truth token at each position:

$$\mathcal{L}_{\text{NTP}} = \sum_t -\log p(x_{t+1} \mid x_{\leq t}). \tag{2}$$

CE loss has two key limitations in long-context settings. First, each term in the summation only considers single-token predictions, ignoring the semantic relationships among the tokens of a multi-token continuation. Second, NTP aggregates the terms uniformly and thus ignores local regions in the sequence that may be especially useful over long contexts [37].

Next-Sequence Prediction (NSP). We propose the NSP training objective for fast weight models to overcome the limitations of standard NTP. While NTP optimizes token-by-token predictions, NSP optimizes multi-token sequence alignment at selected positions $\mathcal{T}^* \subseteq \{1, \ldots, T\}$:

$$\mathcal{L}_{\text{NSP}} = \sum_{t \in \mathcal{T}^*} \mathcal{L}_{\text{seq}}(\hat{x}_{t+1:t+k}, x_{t+1:t+k}), \quad k > 1 \tag{3}$$

where $\mathcal{L}_{\text{seq}}$ measures the discrepancy between predicted ($\hat{x}_{t+1:t+k}$) and ground truth ($x_{t+1:t+k}$) continuations given prefix $x_{\leq t}$.

A straightforward choice for $\mathcal{L}_{\text{seq}}$ is the CE loss. However, naively applying the CE loss at every token position t requires generating k-token completions given all possible prefixes, which is computationally impractical for long contexts. Furthermore, directly matching a single reference would over-penalize plausible answers that do not exactly match the ground truth. For example, "automobiles move quickly" and "cars are fast" are semantically equivalent, but may incur a high CE loss.

We tackle this issue with two approaches. First, instead of unrolling tokens at each t, we select informative positions $\mathcal{T}^*$ with high NTP entropy, indicating high uncertainty. Second, we optimize Eq. (3) with RL to maximize the expected self-supervised reward R of sequence predictions. Let π_θ denote the language model parameterized by θ. We define the sequence-level loss as follows:

$$\mathcal{L}_{\text{seq}} = -\mathbb{E}_{\hat{x}_{t+1:t+k} \sim \pi_\theta(\cdot \mid x_{\leq t})}\big[R(\hat{x}_{t+1:t+k}, x_{t+1:t+k})\big]. \tag{4}$$

This formulation has two advantages: (1) optimizing k-token predictions leverages higher information content than single-token predictions; (2) we can assign smooth rewards to multiple plausible continuations based on their semantic similarity to the ground truth. For brevity, we use $R(t)$ to denote $R(\hat{x}_{t+1:t+k}, x_{t+1:t+k})$.

3.2 REFINE

Our REFINE framework (Fig. 2) consists of four key steps: (1) entropy-based token selection, (2) rollout generation, (3) reward assignment, and (4) optimization with RL.

Entropy-Based Token Selection. We forward the input sequence $S = (x_1, \ldots, x_T)$ through the policy model π_θ and compute the NTP entropy $H_t = H(\pi_\theta(\cdot \mid x_{\leq t-1}))$ for each token position t. We smooth the entropy distribution within S using 1-D average pooling with kernel size w. We partition S into c contiguous chunks of equal length: $S = (S_1, S_2, \ldots, S_c)$. For each chunk S_i, we sample one token position weighted by the softmax of its entropy. Concretely, we compute:

$$p_i(t) = \frac{\mathrm{e}^{\tilde{H}_t}}{\sum_{t' \in \mathcal{T}_i} \mathrm{e}^{\tilde{H}_{t'}}}, \qquad \mathcal{T}_i = \{t' \mid x_{t'} \in S_i\} \tag{5}$$

where $\tilde{H}_t$ is the smoothed H_t. We then sample an index $t_i \sim p_i(t)$. This produces one entropy-weighted token position for each chunk, yielding a set of sampled positions $\mathcal{T}^* = \{t_1, \ldots, t_c\}$. These positions represent regions that exhibit relatively high local uncertainty, allowing training to focus on challenging predictions spread evenly across the context.

Rollout Generation. For each high-entropy position $t_i \in \mathcal{T}^*$, we construct a truncated prefix $x_{\leq t_i}$, yielding c distinct partial sequences $\{x_{\leq t_1}, \ldots, x_{\leq t_c}\}$ from S. These prefixes capture the full context leading up to each sampled high-entropy position. From each truncated prefix $x_{\leq t_i}$, we generate a k-token continuation $\hat{x}_{t_i+1:t_i+k}$ using the current policy. We extract the hidden states of the predicted and ground truth continuations from the final layer before the logits:

$$\mathbf{h}_k^{\text{pred}}(t_i) = \big(\mathbf{h}^{\text{pred}}(t_i+1), \ldots, \mathbf{h}^{\text{pred}}(t_i+k)\big), \qquad \mathbf{h}_k^{\text{gt}}(t_i) = \big(\mathbf{h}^{\text{gt}}(t_i+1), \ldots, \mathbf{h}^{\text{gt}}(t_i+k)\big). \tag{6}$$

We measure the discrepancy between $\mathbf{h}_k^{\text{pred}}(t_i)$ and $\mathbf{h}_k^{\text{gt}}(t_i)$ to compute the reward.

Reward Assignment. Given the hidden states of predicted and ground truth continuations $\mathbf{h}_k^{\text{pred}}(t_i), \mathbf{h}_k^{\text{gt}}(t_i) \in \mathbb{R}^{k \times d}$, we assign a smooth reward R for an arbitrary similarity function φ:

$$R_k^{\varphi}(t_i) = \frac{1}{k} \sum_{j=1}^{k} \varphi\big(\mathbf{h}^{\text{pred}}(t_i+j), \mathbf{h}^{\text{gt}}(t_i+j)\big). \tag{7}$$

We use cosine similarity for φ. This reward encourages the model to generate hidden state trajectories that align with the ground truth, while remaining fully self-supervised. The purpose of a representation similarity-based reward is to improve contextual generalization and training stability. It assigns smooth, non-zero rewards to semantically similar tokens that lead to hidden state embeddings that are closer in the latent space [38]. Qualitative examples of R^{φ} can be found in Appendix §H.

Optimization with RL. Once we compute the reward for each rollout, we have a set of rollouts $o_{\mathcal{T}^*} = \{\hat{x}_{t_1+1:t_1+k}, \ldots, \hat{x}_{t_c+1:t_c+k}\}$ and corresponding rewards $\mathcal{R}_{\mathcal{T}^*} = \{R_k^{\varphi}(t_1), \ldots, R_k^{\varphi}(t_c)\}$. The rewards from the same sequence S are standardized to compute the advantage following Shao et al. [39].

We employ the GRPO algorithm [39] to compute the NSP loss based on the rollouts and their relative advantages. The policy gradients therefore maximize the following objective:

$$\mathcal{J}(\theta) = \mathbb{E}_{x_{\leq t} \sim \mathcal{D},\, \hat{x}_{t+1:t+k} \sim \pi_\theta(\cdot \mid x_{\leq t})}[R_k^{\varphi}(t)], \tag{8}$$

where $\mathcal{D}$ is the set of all $\{x_{\leq t}\}_{t=1}^{T}$. To prevent catastrophic forgetting, the final loss is a weighted sum of the NSP loss and the standard NTP loss (computed over the entire sequence S), with coefficients λ_{RL} and λ_{SFT} respectively. The weights are adjusted based on the training phase.

3.3 Hybrid Reward for RL

TTT introduces unique constraints. First, evaluations are usually conducted in the low-data regime, which leads to smaller batch sizes and limited room for meta-adaptation across episodes. Second, effective memorization of the given context becomes more important than contextual generalization.

Table 2: **Performance on Long-Context Retrieval Tasks.** We evaluate mid-trained (**MidTr**) models on the NIAH tasks in RULER at 4K, 8K, and 16K context lengths (standard SFT vs. REFINE). Highest scores in each category are highlighted in **bold**.

		RULER S-NIAH				RULER MK-NIAH				RULER MQ-NIAH				RULER MV-NIAH			
	MidTr	4K	8K	16K	Avg	4K	8K	16K	Avg	4K	8K	16K	Avg	4K	8K	16K	Avg
LaCT 760M	–	98.8	91.2	95.8	95.3	70.2	44.8	24.2	46.2	39.2	24.9	17.4	27.1	40.6	27.0	17.7	28.4
	SFT	98.4	90.8	**97.6**	95.6	**70.6**	45.0	24.2	46.6	**40.7**	25.2	17.4	27.8	**41.7**	26.3	17.7	28.5
	REFINE	**99.0**	**93.0**	96.8	**96.3**	70.4	**46.0**	**26.6**	**47.7**	40.5	**25.8**	**18.0**	**28.1**	41.2	**29.5**	**18.6**	**29.8**
DeltaNet 1.3B	–	**100.0**	**100.0**	**100.0**	**100.0**	23.6	17.8	3.4	14.9	27.7	3.9	1.8	11.1	23.9	5.4	2.0	10.4
	SFT	**100.0**	**100.0**	**100.0**	**100.0**	23.8	19.2	7.8	16.9	33.2	16.8	3.5	17.8	28.3	19.3	3.7	17.1
	REFINE	**100.0**	**100.0**	99.8	99.9	**25.0**	**21.4**	**8.8**	**18.4**	**37.3**	**18.8**	**3.6**	**19.9**	**29.0**	**20.4**	**4.0**	**17.8**

Table 3: **Performance on Multi-Doc QA Tasks.** We evaluate various training strategies during mid-training (**MidTr**), post-training (**PostTr**), and test-time training (**TTT**) for multi-document question and answering tasks. For Nested SFT and Nested REFINE, we train the model with the training method on the prompt portion of the sample as described in §4.3. Higher is better. First and second highest scores on each task are highlighted in **bold** and underline, respectively.

				RULER SQuADQA				RULER HotpotQA			
	MidTr	PostTr	TTT	4K	8K	16K	Avg	4K	8K	16K	Avg
LaCT 760M	-	-	-	17.5	14.0	6.5	12.7	20.5	14.5	12.0	15.7
	SFT	-	-	19.5	14.0	6.0	13.2	21.5	15.0	12.0	16.2
	REFINE	-	-	20.5	15.5	8.0	14.7	23.0	16.0	12.5	17.2
	REFINE	SFT	-	30.5	20.0	7.5	19.3	25.0	16.5	11.5	17.7
	REFINE	Nested SFT	-	38.0	20.0	7.5	21.8	24.0	16.0	12.5	17.5
	REFINE	Nested REFINE	-	43.5	24.5	8.5	25.5	27.0	19.5	13.0	19.8
	REFINE	Nested REFINE	SFT	43.5	21.5	8.5	24.5	26.5	24.0	13.0	21.2
	REFINE	Nested REFINE	REFINE	**45.5**	**25.5**	**10.0**	**27.0**	**28.5**	**25.5**	**15.0**	**23.0**
DeltaNet 1.3B	-	-	-	11.0	5.0	2.0	6.0	10.0	3.0	2.5	5.2
	SFT	-	-	9.0	6.0	3.0	6.0	9.0	8.0	4.5	7.2
	REFINE	-	-	10.0	6.5	4.0	6.8	12.0	9.5	5.5	9.0
	REFINE	SFT	-	11.0	7.0	5.0	7.7	14.0	11.0	6.5	10.5
	REFINE	Nested SFT	-	12.0	8.0	5.0	8.3	16.5	10.0	7.5	11.3
	REFINE	Nested REFINE	-	14.0	10.5	6.5	10.3	15.0	11.0	**8.5**	11.5
	REFINE	Nested REFINE	SFT	16.5	10.5	6.5	11.2	16.0	10.5	6.5	11.0
	REFINE	Nested REFINE	REFINE	**17.5**	**12.5**	**7.0**	**12.3**	**18.0**	**13.5**	8.0	**13.2**

For scenarios that require stronger context memorization (*e.g.*, TTT), we introduce a binary exact match reward R^{binary} defined as:

$$R_k^{\text{binary}}(t_i) = \frac{1}{k}\sum_{j=1}^{k}\mathcal{I}[x_{t_i+j} = \hat{x}_{t_i+j}], \tag{9}$$

where $\mathcal{I}[\cdot]$ is the indicator function. During post-training, we use the hybrid reward $R_k^{\text{hybrid}}(t_i) = R_k^{\varphi}(t_i) + R_k^{\text{binary}}(t_i)$, since the training and test datasets have similar distributions. The hybrid reward aims to balance contextual generalization and memorization.

4 Experiments

We conduct experiments to show that REFINE improves fast weight models on long-context tasks. We first establish the models, datasets, and baselines used for training and evaluation (§4.1). We then report results across three training phases: mid-training (§4.2), post-training (§4.3), and test-time training (§4.4). Finally, we provide analysis on fast weight drift, reward assignment, and entropy-based token selection (§4.5), followed by ablations on rollout length and the number of chunks (§4.6).

4.1 Setup

Models. We use two pre-trained fast weight language models, LaCT-760M [13] and DeltaNet-1.3B [11]. LaCT adapts the model by updating its fast weight parameters, whereas DeltaNet keeps parameters fixed but updates a parallelizable memory state. We show that REFINE can improve these distinct fast weight mechanisms during mid-training, post-training, and test-time training.

Table 4: **Performance on Long-Context Tasks in LongBench.** We study the impact of the learning algorithm during mid-training (**MidTr**) and test-time training (**TTT**) on tasks with long-context. **SFT** denotes the supervised fine-tuning with next-token prediction. We evaluate on 12 tasks in LongBench, filtered for samples with at most 16K tokens. Details are similar to Tab. 3.

			Single-doc QA			Multi-doc QA		Summarization		Few-shot QA			Coding		
	MidTr	TTT	NQ	QR	MF	HP	2W	QM	MN	SS	TC	TQ	LC	RP	Avg
LaCT 760M	-	-	6.5	10.5	7.2	11.7	9.8	13.6	9.2	14.2	10.5	8.0	26.7	29.7	13.1
	SFT	-	5.8	10.1	7.4	12.6	9.2	13.1	10.5	13.8	7.5	12.2	29.8	30.1	13.5
	REFINE	-	6.5	11.1	12.6	**19.6**	18.0	**14.3**	15.9	17.0	11.0	12.9	**32.9**	31.1	16.9
	REFINE	SFT	5.1	12.6	13.5	15.9	18.4	13.2	**16.2**	**17.4**	12.5	**16.2**	32.0	31.4	17.0
	REFINE	REFINE	**6.7**	**14.5**	**14.1**	18.4	**22.8**	13.9	15.9	**17.4**	**15.5**	11.8	32.2	**32.3**	**18.0**
DeltaNet 1.3B	-	-	6.5	8.7	10.0	4.8	6.4	12.4	**16.3**	9.4	17.7	15.1	33.8	29.0	14.2
	SFT	-	5.7	9.4	8.3	9.2	8.6	14.7	16.1	15.5	22.9	15.2	33.1	29.2	15.7
	REFINE	-	6.5	9.5	10.1	**9.6**	**8.7**	**15.2**	15.0	16.2	25.0	21.4	**35.9**	**31.1**	17.0
	REFINE	SFT	7.2	**9.6**	10.5	6.8	7.2	14.9	16.2	**17.3**	28.0	16.6	34.1	29.2	16.5
	REFINE	REFINE	**7.5**	9.2	**11.5**	9.5	8.6	14.7	16.1	16.5	**31.5**	**24.7**	35.2	30.0	**17.9**

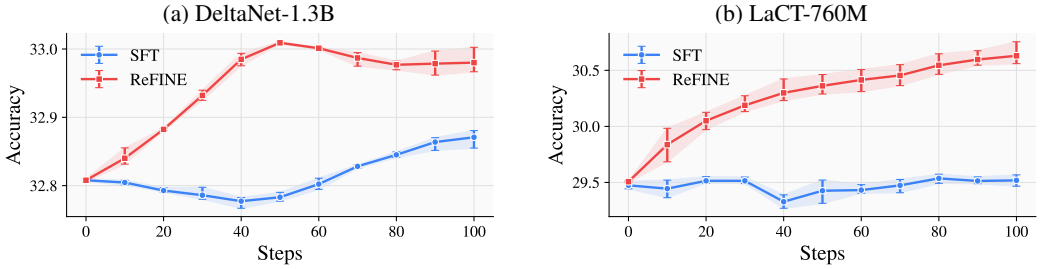


Figure 3: **NTP Accuracy on Booksum.** REFINE mid-training on DeltaNet-1.3B **(a)** and LaCT-760M **(b)** consistently improves NTP accuracy on the validation dataset while SFT is stagnant. The error bars show the minimum and maximum values from three independent trials.

Baselines. We compare REFINE with SFT in our main evaluations. We provide the full REFINE training configurations for each phase in Appendix §E. We also evaluate additional baselines that train with SFT-based multi-token objectives: (1) up-weighting the CE loss of multiple tokens at high entropy positions [37], and (2) enabling multi-token prediction via architectural modification [24]. We show that REFINE outperforms these baselines in Appendix §G, Tabs. G.1 to G.3.

Datasets and Benchmarks. For mid-training, we employ a dataset similar to that used to pre-train the fast weight models. Specifically, *we perform mid-training on Long-Data-Collections* [40], which is the pre-training dataset for LaCT [13]. We evaluate the quality of mid-trained models on RULER NIAH tasks [25] and Booksum [41]. We consider two additional scenarios: (1) multi-doc QA and (2) broader long-context tasks. For multi-doc QA, we *post-train on synthetically generated SQuADQA and HotpotQA tasks from RULER*. Then, we apply TTT during evaluation. For broader long-context tasks, we employ 12 tasks from LongBench [8] and apply TTT on the mid-trained models. More details on these datasets can be found in Appendix §D, Tab. D.1.

4.2 Impact of REFINE on Mid-Training

We mid-train the models on 200M tokens from Long-Data-Collections [40] for 1 epoch. The models are evaluated on four NIAH tasks in RULER [25] (Single-key-query-value, Multi-key, Multi-query, and Multi-value) at 4K, 8K, and 16K context lengths using Language Model Evaluation Harness [42].

Tab. 2 shows that the REFINE mid-trained models consistently outperform the original pre-trained model and the SFT mid-trained model on different tasks. For example, REFINE significantly improves DeltaNet in the Multi-key NIAH (+23.5% from no mid-training and +8.8% from SFT mid-training). This suggests that REFINE leads to improvements in long-context retrieval.

We also report the validation NTP accuracy on the Booksum [41] dataset. Fig. 3 shows the evolution of validation NTP accuracy during mid-training. Interestingly, REFINE improves NTP accuracy more than SFT even though the SFT baseline directly optimizes NTP. This holds for LaCT, whose NTP accuracy stagnates for SFT, suggesting that NSP provides learning signals that NTP does not.

We report the effectiveness of REFINE mid-training on multi-doc QA and LongBench [8] tasks in Tab. 3 and Tab. 4. In Tab. 3, REFINE mid-trained models (3rd rows) outperform SFT mid-trained models (2nd rows) by large margins. For example, REFINE mid-training improves the average performance of pre-trained DeltaNet on RULER HotpotQA by 25.0% compared to SFT mid-training. Tab. 4 shows a similar result; REFINE (3rd rows) steadily outperforms SFT (2nd rows).

4.3 Impact of REFINE on Post-Training

We show that REFINE strengthens post-training, which aims to align the model's responses with a given task. In our experiments, we fine-tune the mid-trained models in §4.2 on synthetically generated samples for the target tasks of RULER [25] SQuADQA and HotpotQA.

During post-training, we apply REFINE as a nested learning algorithm. Within each training loop, we first train the model on the prompt with REFINE, and then fine-tune the final response to align with the reference. We compare three post-training scenarios. (1) *SFT*: we fine-tune the model directly on the post-training dataset with NTP (no nested learning). (2) *Nested SFT*: we apply nested training strategy with NTP loss. (3) *Nested* REFINE: we apply nested training strategy with REFINE.

Tab. 3 shows that post-training with nested REFINE (6th rows) outperforms SFT (4th rows) and nested SFT (5th rows). For instance, nested REFINE on LaCT-760M improves the average score on SQuADQA by 17% compared to nested SFT (25.5 vs. 21.8), and by 24.1% for DeltaNet-1.3B (10.3 vs. 8.3). These results suggest that NSP provides better task-agnostic learning signals than NTP.

4.4 Impact of REFINE on Test-Time Training (TTT)

REFINE can be used during inference to improve performance on a target task. During inference, we apply REFINE on the prompt before letting the model generate the final response. In order to maximize memory capacity over long contexts, we provide more direct learning signals by using binary exact match reward (R^{binary}) and a higher RL loss coefficient ($\lambda_{\text{RL}} = 0.4$). We replace REFINE with pure SFT on the prompt for comparison.

We apply TTT on the post-trained models for multi-doc QA tasks in §4.3. The 7th and 8th rows of Tab. 3 show the results of SFT TTT and REFINE TTT, respectively. REFINE consistently outperforms SFT during TTT, as in the mid-training and post-training scenarios. Similarly, Tab. 4 shows that TTT with REFINE yields superior performance compared to SFT across diverse subtasks in LongBench [8]. This suggests that NSP provides stronger test-time adaptation signals.

4.5 Analysis

Analysis of Drift in Fast Weights. We study how REFINE mid-training affects autoregressive drift in fast weight memory. We sample 20 examples from QMSum and SAMSum in LongBench [8], and compare pre-trained DeltaNet-1.3B with SFT- and REFINE-mid-trained variants. For each model, we generate responses and extract the fast weight memory matrices from all layers after each predicted and ground-truth response token. We measure drift as the Frobenius norm between the memory states induced by the predicted and ground-truth trajectories. As shown in Fig. 4, REFINE consistently reduces drift relative to SFT, suggesting that NSP improves the long-horizon stability of fast weights. More discussion on the drift in fast weights can be found in Appendix §G, Fig. G.1 and Tab. G.6.

Analysis of Reward Functions. We analyze the impact of reward functions for REFINE. During mid-training, REFINE assigns a semantically driven reward to each rollout based on the cosine similarity of the hidden states of predicted and ground truth tokens (Eq. (7)). We replace cosine similarity rewards (R^{φ}) with binary exact match rewards (R^{binary}), hybrid rewards (R^{hybrid}), or negative Euclidean distance rewards (R^{NED}). R^{NED} measures the negative Euclidean distance of predicted and ground truth hidden states. Tab. 5 shows that R^{φ} achieves superior performance for mid-training: +1.8% over R^{binary} for LaCT-760M and +3.0% for DeltaNet-1.3B. R^{NED} performs on par with R^{φ} as negative Euclidean distance is monotonic to cosine similarity after advantage standardization. This suggests that similarity-based rewards lead to better generalization. More analysis on TTT can be found in Appendix §G, Tab. G.8.

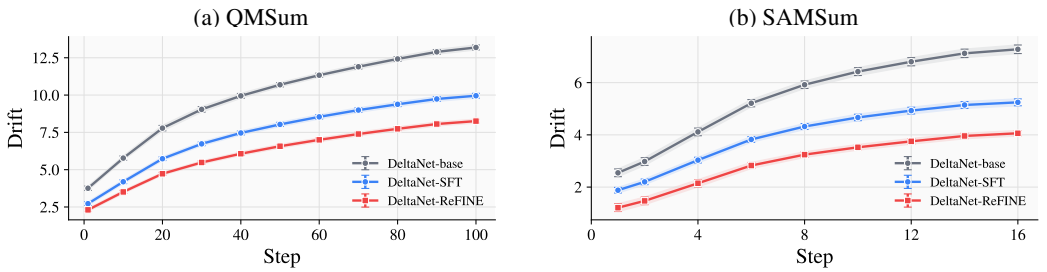


Figure 4: **Fast Weight Drift.** We measure drift as the Frobenius norm between the fast weight memory matrices induced by predicted and ground truth response trajectories, averaged across layers. Mid-training DeltaNet-1.3B with REFINE lowers the drift on QMSum **(a)** and SAMSum **(b)** tasks in LongBench [8] more than SFT. Error bars show $\pm 1\sigma$ across 20 independent generations.

Table 5: **Impact of Reward Function on Mid-Training.** We compare $\mathcal{R}^{\text{binary}}$, $\mathcal{R}^{\text{hybrid}}$, $\mathcal{R}^{\text{NED}}$ and $\mathcal{R}^{\varphi}$ on 12 LongBench tasks.

	Reward	Avg.
LaCT-760M +REFINE MidTr	$\mathcal{R}^{\text{binary}}$	16.6
	$\mathcal{R}^{\text{hybrid}}$	16.8
	$\mathcal{R}^{\text{NED}}$	16.8
	$\mathcal{R}^{\varphi}$ (ours)	16.9
DeltaNet-1.3B +REFINE MidTr	$\mathcal{R}^{\text{binary}}$	16.5
	$\mathcal{R}^{\text{hybrid}}$	16.8
	$\mathcal{R}^{\text{NED}}$	17.0
	$\mathcal{R}^{\varphi}$ (ours)	17.0

Table 6: **Impact of Token Selection on Mid-Training.** We compare uniform, $\arg\max H$, $\arg\min H$, and entropy-weighted sampling.

	Sampling	Avg.
LaCT-760M +REFINE MidTr	Uniform	16.2
	$\arg\max H$	16.4
	$\arg\min H$	16.6
	Ours	16.9
DeltaNet-1.3B +REFINE MidTr	Uniform	15.9
	$\arg\max H$	16.7
	$\arg\min H$	16.8
	Ours	17.0

Analysis of Entropy-Based Token Selection. We analyze the role of entropy-based token selection on REFINE mid-training. REFINE samples a target token from each chunk weighted by the token-level NTP entropy. Rollouts are generated to predict continuations from the sampled target tokens. We repeat the mid-training process after replacing entropy-based sampling with three alternative sampling methods: uniform sampling, maximum entropy selection, and minimum entropy selection. Tab. 6 shows that entropy-weighted sampling achieves the best performance on both models. This suggests that NSP training is most effective when applied to regions with varying levels of uncertainty.

4.6 Ablations

Rollout Length. We examine the effect of rollout length k on REFINE mid-training, which is the number of tokens to unroll per rollout. k determines how far the model is expected to predict given a prefix. We mid-train both models with different values of k and evaluate on LongBench tasks in Tab. 4. We observe that the average score increases until $k = 5$ and decreases at $k = 7$ (Fig. 5, *left*). We hypothesize that the sharpness of the reward starts to degrade when the rollouts extend further. More discussion on the reward distribution can be found in Appendix §G.

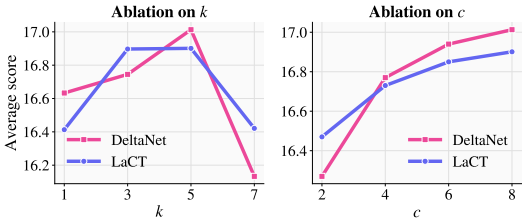


Figure 5: **Ablation on k and c.** We mid-train models with different numbers of tokens per rollout k (**left**) and numbers of chunks per sequence c (**right**). We evaluate on 12 tasks in LongBench [8]. With cosine similarity reward, there is an optimal k. Higher c leads to more NSP training per sequence, improving overall performance.

Number of Chunks per Sequence. We study the impact of the number of chunks c per sequence on downstream performance by mid-training the models with different numbers of chunks. c determines the number of target tokens sampled based on entropy values, as well as the total number of rollouts per sequence. We evaluate the mid-trained models on LongBench tasks in Tab. 4. We find that the average score increases consistently as the number of chunks per sequence increases (Fig. 5, *right*). The average score increases from 16.5 ($c = 2$) to 16.9 ($c = 8$) for LaCT-760M and from 16.3

($c = 2$) to 17.0 ($c = 8$) for DeltaNet-1.3B. This indicates that the quality of fast weight initializations increases as the frequency of sequence-level predictions increases.

5 Discussion

Conclusion. We introduce the NSP training objective for fast weight language models to address the limitations of NTP in providing sequence-level feedback. We propose REFINE, an RL framework which leverages entropy-based token selection and sequence-level rewards to efficiently train fast weight models under the NSP objective. Our experiments demonstrate that REFINE is effective throughout the training lifecycle of fast weight models, showing consistent improvements in long-context benchmarks and reducing inference-time drift. REFINE presents RL for NSP as a flexible and practical pathway towards long-context modeling of fast weight architectures.

Limitations. First, REFINE's cosine similarity reward deteriorates for overly long rollouts, leading to diminishing returns on k. Reward functions that capture richer semantic similarity could address this. Second, the optimal rollout length for a given prefix is context-dependent, suggesting that dynamic adjustment may better isolate semantically meaningful regions.

Future Work. Fully incorporating NSP into the standard training framework of fast weight models requires architectural changes. Efficient transfer of fast weights across truncated prefixes will significantly accelerate rollout generation, allowing REFINE to scale data and compute further.

Acknowledgements

All experiments, data collection, and processing activities were conducted at Princeton University, and no Google DeepMind computing resources were used for this work. We thank Yoonsang Lee and Tianyuan Zhang for helpful discussions and valuable insights throughout this project.

Funding Transparency Statement

Funding. Hee Seung Hwang is supported by the Princeton University First-Year Fellowship in the Natural Sciences and Engineering. Sanghyuk Chun is supported by the Francis Robbins Upton Fellowship at Princeton University. Xindi Wu is supported by the Apple Fellowship.

Competing Interests. The authors declare no competing interests.

References

[1] Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong, Mor Geva, Jonathan Berant, et al. Scrolls: Standardized comparison over long language sequences. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 12007–12021, 2022.

[2] Z Dong, T Tang, J Li, WX Zhao, and JR Wen. Bamboo: a comprehensive benchmark for evaluating long text modeling capacities of large language models. arxiv. *arXiv preprint arXiv:2309.13345*, 2023.

[3] Rishabh Agarwal, Avi Singh, Lei Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, et al. Many-shot in-context learning. *Advances in Neural Information Processing Systems*, 37:76930–76966, 2024.

[4] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*, 2024.

[5] Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

[6] Daniel Nichols, Joshua H Davis, Zhaojun Xie, Arjun Rajaram, and Abhinav Bhatele. Can large language models write parallel code? In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, pages 281–294, 2024.

[7] Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. Helmet: How to evaluate long-context language models effectively and thoroughly. *arXiv preprint arXiv:2410.02694*, 2024.

[8] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

[9] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, 2025.

[10] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In *International conference on algorithmic learning theory*, pages 597–619. PMLR, 2023.

[11] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37:115491–115522, 2024.

[12] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, 2025.

[13] Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T Freeman, and Hao Tan. Test-time training done right. *arXiv preprint arXiv:2505.23884*, 2025.

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

[15] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pages 9355–9366. PMLR, 2021.

[16] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

[17] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

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

[19] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International conference on machine learning*, pages 4528–4537. PMLR, 2018.

[20] Zekun Li, Tongxin Bai, Man Yao, Zhen Qin, Guoqi Li, et al. Mva: Linear attention with high-order query-keys integration and multi-level vocabulary decomposition. In *Forty-second International Conference on Machine Learning*, 2025.

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

[22] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

[23] 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. *arXiv preprint arXiv:2505.23735*, 2025.

[24] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

[25] 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? *arXiv preprint arXiv:2404.06654*, 2024.

[26] Kevin Clark, Kelvin Guu, Ming-Wei Chang, Panupong Pasupat, Geoffrey Hinton, and Mohammad Norouzi. Meta-learning fast weight language models. *arXiv preprint arXiv:2212.02475*, 2022.

[27] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, 2020.

[28] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[29] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

[30] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[31] Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures. *arXiv preprint arXiv:2512.24695*, 2025.

[32] 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*, pages 9229–9248. PMLR, 2020.

[33] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021.

[34] Yossi Gandelsman, Yu Sun, Xinlei Chen, and Alexei A Efros. Test-time training with masked autoencoders. In *Advances in Neural Information Processing Systems*, volume 35, pages 29374–29385, 2022.

[35] Qingxiu Dong, Li Dong, Yao Tang, Tianzhu Ye, Yutao Sun, Zhifang Sui, and Furu Wei. Reinforcement pre-training. *arXiv preprint arXiv:2506.08007*, 2025.

[36] Ali Hatamizadeh, Syeda Nahida Akter, Shrimai Prabhumoye, Jan Kautz, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, and Yejin Choi. Rlp: Reinforcement as a pretraining objective. *arXiv preprint arXiv:2510.01265*, 2025.

[37] Lizhe Fang, Yifei Wang, Zhaoyang Liu, Chenheng Zhang, Stefanie Jegelka, Jinyang Gao, Bolin Ding, and Yisen Wang. What is wrong with perplexity for long-context language modeling? *arXiv preprint arXiv:2410.23771*, 2024.

[38] Kiho Park, Yo Joong Choe, Yibo Jiang, and Victor Veitch. The geometry of categorical and hierarchical concepts in large language models. *arXiv preprint arXiv:2406.01506*, 2024.

[39] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[40] TogetherAI. Long data collections database. `https://huggingface.co/datasets/togethercomputer/Long-Data-Collections`, 2024.

[41] Wojciech Kryściński, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. Booksum: A collection of datasets for long-form narrative summarization. In *Findings of the association for computational linguistics: EMNLP 2022*, pages 6536–6558, 2022.

[42] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024.

[43] Yangzhou Liu, Yue Cao, Hao Li, Gen Luo, Zhe Chen, Weiyun Wang, Xiaobo Liang, Biqing Qi, Lijun Wu, Changyao Tian, et al. Sequential diffusion language models. *arXiv preprint arXiv:2509.24007*, 2025.

[44] Qwen Team. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

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

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

[47] Audrey Huang, Adam Block, Dylan J Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan T Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. *arXiv preprint arXiv:2412.01951*, 2024.

[48] Rachit Bansal, Aston Zhang, Rishabh Tiwari, Lovish Madaan, Sai Surya Duvvuri, Devvrit Khatri, David Brandfonbrener, David Alvarez-Melis, Prajjwal Bhargava, Mihir Sanjay Kale, et al. Let's (not) just put things in context: Test-time training for long-context llms. *arXiv preprint arXiv:2512.13898*, 2025.

[49] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[50] Rewon Child. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[51] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[52] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33: 17283–17297, 2020.

[53] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[54] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[55] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[56] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

[57] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

[58] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

[59] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. RWKV: Reinventing RNNs for the transformer era. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

[60] Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Haowen Hou, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, et al. RWKV-7 "Goose" with expressive dynamic state evolution. *arXiv preprint arXiv:2503.14456*, 2025.

[61] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning*, 2024.

[62] Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. HGRN2: Gated linear RNNs with state expansion. In *Conference on Language Modeling*, 2024.

[63] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended long short-term memory. *Advances in Neural Information Processing Systems*, 37, 2024.

[64] Kimi Team. Kimi Linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

[65] Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026.

[66] Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. Longhorn: State space models are amortized online learners. In *International Conference on Learning Representations*, 2025.

[67] 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*, 2026.

[68] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

[69] Aleksandar Botev, Soham De, Samuel L Smith, Anushan Fernando, George-Cristian Muraru, Ruba Haroun, Leonard Berrada, Razvan Pascanu, Pier Giuseppe Sessa, Robert Dadashi, et al. RecurrentGemma: Moving past transformers for efficient open language models. *arXiv preprint arXiv:2404.07839*, 2024.

[70] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-Mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

[71] Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. In *International Conference on Machine Learning*, 2024.

[72] Maurice Weber, Dan Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, et al. Redpajama: an open dataset for training large language models. *Advances in neural information processing systems*, 37: 116462–116492, 2024.

[73] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[74] Yi Tay, Mostafa Dehghani, Vinh Q Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Siamak Shakeri, Dara Bahri, Tal Schuster, et al. Ul2: Unifying language learning paradigms. *arXiv preprint arXiv:2205.05131*, 2022.

[75] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487, 2022.

[76] Stephen Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, et al. Promptsource: An integrated development environment and repository for natural language prompts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 93–104, 2022.

[77] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[78] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

[79] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[80] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[81] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020.

[82] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[83] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[84] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics*, 2016.

[85] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv preprint arXiv:2304.09433*, 2023.

[86] Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. Openceres: When open information extraction meets the semi-structured web. 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 3047–3056, 2019.

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

A Notation

Table A.1: Glossary and notation.

Symbol	Description
Sequences and Tokens	
$S = (x_1, \ldots, x_T)$	Input sequence with T tokens
x_t	Token at position t
$x_{<t}$, $x_{\leq t}$	Token sequence up to (and including) position t
$x_{t+1:t+k}$	k-token ground-truth continuation starting from position $t+1$
$\hat{x}_{t+1:t+k}$	Generated rollout token sequence from position $t+1$
T	Total sequence length
k	Rollout length (number of tokens to predict per rollout)
c	Number of chunks per sequence
n	Number of rollouts per sampled position
S_i	i-th chunk of the input sequence
$\mathcal{D}$	Training data distribution
Model and Policy	
π_θ	Policy model with parameters θ
$\pi_{\theta_{\text{old}}}$	Old policy (before update)
$p(x_t \mid x_{<t})$	Conditional probability of token x_t given context
$\mathbf{h}^{\text{pred}}(t)$	Hidden state vector of predicted token at position t
$\mathbf{h}^{\text{gt}}(t)$	Hidden state vector of ground-truth token at position t
$\mathbf{h}_k^{\text{pred}}(t)$, $\mathbf{h}_k^{\text{gt}}(t)$	Matrix of k hidden state vectors starting at position $t+1$ ($\in \mathbb{R}^{k\times d}$)
d	Hidden state dimension
Entropy and Sampling	
H_t	Entropy at token position t
$\tilde{H}_t$	1-D average pooled entropy at token position t
$\mathcal{T}^* = \{t_1, \ldots, t_c\}$	Set of sampled high-entropy positions
$\mathcal{T}_i$	Set of token positions in chunk i
$p_i(t)$	Entropy-weighted sampling probability in chunk i for token position t
Loss Functions	
$\mathcal{L}_{\text{NTP}}$	Next-Token Prediction loss (cross-entropy)
$\mathcal{L}_{\text{NSP}}$	Next-Sequence Prediction loss
$\mathcal{L}_{\text{seq}}$	Sequence-level discrepancy measure
$\mathcal{J}(\theta)$	RL objective to maximize

Table A.2: Glossary and notation (continued).

Symbol	Description
Rewards	
$R(t)$	Self-supervised reward at position t (shorthand for $R(\hat{x}_{t+1:t+k}, x_{t+1:t+k})$)
$R_k^{\varphi}(t)$	Cosine similarity reward over k-token rollout
$R_k^{\text{binary}}(t)$	Binary exact match reward over k-token rollout
$R_k^{\text{hybrid}}(t)$	Hybrid reward ($R_k^{\varphi} + R_k^{\text{binary}}$)
$R_k^{\text{NED}}(t)$	Negative Euclidean distance reward over k-token rollout
$\mathcal{R}_{\mathcal{T}^*}$	Set of rewards for all sampled positions in $\mathcal{T}^*$
$\cos(\cdot, \cdot)$	Cosine similarity function
$\mathcal{I}[\cdot]$	Indicator function
$o_{\mathcal{T}^*}$	Set of all rollouts at sampled positions
Fast Weight Architecture	
$\mathbf{W}_t$	Fast weight matrix at token position t
η	Learning rate for fast weight update
$\mathbf{k}_t, \mathbf{v}_t, \mathbf{q}_t, \mathbf{o}_t,$	Key, value, query, and output vector representations at position t

B Ethics Statement

We believe that the broader societal impact of our work aligns with the consequences of improving long-context modeling capabilities of fast weight language models. We adopt verified and publicly available datasets and classes of models for our experiments, and therefore do not see a clear negative impact on the research community and beyond. However, we advise users to take caution in the downstream applications of our work.

C Related Work

Multi-Token Prediction. Predicting more than one token as a training objective has been explored in previous studies. Gloeckle et al. [24] tackle this problem by estimating k tokens in parallel using k independent output heads. This approach achieves significant gains in throughput by applying architectural modifications to the language model, with minimal degradation of performance on downstream tasks. However, this approach is limited in capturing dependencies among the predicted tokens and relies on a fixed prediction horizon k. REFINE computes rewards based on each multi-token rollout as a whole, capturing the semantic connection among them.

Liu et al. [43] use diffusion-based generation to predict multiple masked tokens simultaneously and optimize the cross-entropy loss between the predicted tokens and masked ground truth tokens. Their method is primarily designed for standard attention-based transformer architectures that predict the next token with masking. However, REFINE is designed to train fast weight language models that store information and update their parameters with a fundamentally different set of rules. We view this work as valuable motivation, although it is evaluated on a different class of models.

Continued Pre-Training with RL. Studies have explored using RL as a tool for the next-token prediction objective. Dong et al. [35] sample reasoning traces before next-token prediction and assign rewards based on the similarity of the byte-sequences of predicted and ground truth tokens. Hatamizadeh et al. [36] take a similar approach by sampling reasoning traces for next-token prediction but assigning rewards by measuring the gap between the log-likelihood of the ground truth token with and without the reasoning trace as context. However, these works focus on attention-based

transformer models (DeepSeek-R1-Distill-Qwen-14B [30] and Qwen3-1.7B-Base [44], respectively) and assume basic reasoning capabilities that enable exploration with Chain-of-Thought. Whether RL can be used for pre-trained fast weight models without prior instruction tuning or human preference optimization, especially in long-context settings, is yet to be explored. Further, REFINE leverages RL for training on the NSP objective which optimizes sequence-level predictions rather than single-token predictions under the NTP objective.

TTT in Language Models. Recent work has explored training standard transformer-based language models during inference to improve performance on target tasks. These methods usually involve extracting task-related learning signals from the model itself for offline adaptation. Akyürek et al. [45] generate relevant in-context examples from the given task and train the model on those examples before generating the final answer to the actual task. RL-based methods extract pseudo-labels by aggregating multiple responses to the same task and assigning rewards to each response based on their similarity to the pseudo-label [46, 47]. Recently, context-based test-time training has also been explored in transformer-based language models. In an attempt to overcome the inherent limitations of static attention, Bansal et al. [48] execute gradient updates on the query projection matrices using the context while keeping other parameters frozen. These approaches apply task-aware or task-agnostic TTT methods to standard transformer-based language models. TTT with REFINE, on the other hand, aims to improve contextual adaptation and memory of fast weights for long contexts, which is a novel setting that has not yet been explored.

Efficient Attention Variants. Standard transformer-based language models with full attention incur quadratic computational complexity as a function of context length. Recent work has developed techniques to reduce the computational and memory overhead in long-context modeling. Sparse attention addresses this problem by introducing computational sparsity in the original attention mechanism. Grouped Query Attention [49] employs sparsity along the head dimension to assign each query head to different groups that share a single key-value head. Sliding-window attention [50–52] architectures leverage sparsity along the context by computing local attention on a fixed number of contiguous tokens.

Linear attention approximates the softmax kernel in the attention formulation to achieve linear computational complexity. Linformer [53] replaces the attention mechanism with low-rank matrix operations, which has been shown to be effective for sequence processing but limited in autoregressive generation. Performer [54] uses orthogonal random features to approximate softmax kernels in the attention, achieving linear complexity without employing low rank matrices. Similarly, Linear Transformer [55] approximates the softmax with linear dot-product of kernel feature maps. The success of linear attention has motivated the development of architectures that operate on linear computational complexity by design, including State Space Models (SSMs), such as Mamba [56, 57]. Linear attention can also be read as a recurrent model: its key-value outer products accumulate into a fixed-size state that is updated once per token, so decoding requires constant memory regardless of context length. A growing body of work builds on this view, which we group into three families below: gated linear recurrent models, fast weight models, and hybrid recurrent attention models.

Gated Linear Recurrent Models. This family augments linear attention with learned, input-dependent gates that control how the recurrent state decays and accumulates information over time. RetNet [58] introduces an exponentially decayed retention state with equivalent parallel, recurrent, and chunkwise recurrent forms. RWKV [59] combines receptance-controlled key and value aggregation with parallelizable training and recurrent inference. RWKV-7 [60] introduces a generalized delta rule update with vector-valued gates and input-dependent learning rates, improving the expressivity and state tracking capabilities of the recurrent state. Gated Linear Attention [61] introduces data-dependent decay over a matrix-valued linear attention state, while HGRN2 [62] expands the recurrent state to improve its representational capacity. Similarly, the matrix LSTM component of xLSTM [63] maintains a normalized matrix memory using exponential input and forget gates. These models primarily improve recurrent language modeling through more expressive gating, forgetting, and state expansion mechanisms. REFINE is agnostic to this choice of gating or state design: rather than proposing a new recurrent architecture, it targets how the recurrent update of an existing architecture is trained so that it stays accurate once the model starts conditioning on its own predictions.

Fast Weight Models. Fast weight models [15] cast the recurrent state itself as a set of fast weights, parameters that are written to and read from at every step through key-value operations. DeltaNet [11] replaces the additive update of linear attention with a delta rule that corrects the value currently associated with an incoming key before writing a new value. Gated DeltaNet [12] combines this corrective update with input-dependent forgetting, Kimi Delta Attention [64] extends this decay mechanism with channel-wise decay for finer-grained control over memory retention, and Gated DeltaNet-2 [65] further decouples the erase and write gates, allowing the model to control removal of old associations and insertion of new values independently. Other work derives fast weight updates more explicitly from online learning objectives. Longhorn [66] formulates its state update as an amortized solution to online associative recall. Test-Time Training (TTT) layers [14] treat the recurrent state as a predictive model updated through self-supervised learning. Titans [22] introduces a neural long-term memory that learns to memorize the preceding context at test time and combines this persistent memory with attention-based short-term memory. MesaNet [67] updates its memory using a locally optimal test-time regression rule. These works design or refine the form of the online update rule itself. Our REFINE instead leaves the update rule fixed and changes how the model is trained to use it, so that the memory produced during training matches the one the model actually relies on during autoregressive generation.

Hybrid Recurrent Attention Models. This family augments a recurrent backbone with a limited amount of explicit attention to compensate for the fixed-size state's bounded capacity to compress the entire preceding context. Griffin [68] interleaves gated recurrent layers with local attention, and RecurrentGemma [69] scales this architecture to pretrained and instruction-tuned language models. Jamba [70] similarly interleaves Mamba and Transformer layers, while BASED [71] combines linear attention with sliding-window attention to balance recurrent state efficiency against associative recall.

Across these families, contextual information is compressed into an input-dependent state rather than retained in an ever-growing key-value cache. Prior work has primarily focused on designing more expressive state representations and update rules. In contrast, REFINE does not introduce a new recurrent architecture. It provides a training framework for fast weight models that improves how their existing online memory updates are learned and used through sequence-level reinforcement learning.

D Datasets and Benchmarks

We summarize the datasets and benchmarks used for training and evaluation in Tab. D.1. The Long-Data-Collections [40] (Apache 2.0 License) dataset contains a 68.8B-token pre-training corpus subsampled from RedPajama [72], Pile [73], UL2 Oscar [74], NI [75], and P3 [76]. We use a 200M-token subset of Long-Data-Collections for mid-training. For LongBench [8] (MIT License), we select 12 English-language subtasks. We leave out MuSiQue and GovReport tasks because they have fewer than 20 samples under 16K tokens. For RULER [25] (Apache 2.0 license) multi-doc QA tasks, we use the official synthetic data generation script to create evaluation samples.

E Training Configuration

Hyperparameters. We provide the training hyperparameters used for REFINE during all training phases in Tab. E.1. We only adjust the train batch size, reward function, and RL loss coefficient across training phases, while keeping all else equal.

Compute. We use 8 L40 GPUs for mid-training and post-training, and 4 L40 GPUs for TTT. We use fewer GPUs for TTT because the train batch size for TTT is smaller (8 samples per batch) compared to other training phases (128 for mid-training, 64 for post-training). Mid-training LaCT-760M and DeltaNet-1.3B with REFINE on 200M tokens from Long-Data-Collections [40] at 16K context takes approximately 24 hours.

F Evaluation Details

We evaluate models on RULER [25] NIAH tasks with Language Model Evaluation Harness [42]. For evaluation on RULER QA and LongBench [8] tasks, we implement custom pipelines in order to be

Table D.1: Summary of datasets and benchmarks used across training phases.

Phase	Dataset	Metric	Context	Size
Mid-training	Long-Data-Collections	-	16K	∼200M tokens
	RULER NIAH	recall	4K/8K/16K	500 per context
	Booksum	NTP Accuracy, CE loss	≤16K	9600
Post-training	RULER SQuADQA	recall	4K/8K/16K	1600 train / 200 test
	RULER HotpotQA	recall	4K/8K/16K	1600 train / 200 test
Test-time	NarrativeQA (NQ)	F1	≤16K	56
	Qasper (QR)	F1	≤16K	184
	MultiFieldQA (MF)	F1	≤16K	136
	HotpotQA (HP)	F1	≤16K	96
	2WikiMHQA (2W)	F1	≤16K	184
	QMSum (QM)	rouge	≤16K	104
	MultiNews (MN)	rouge	≤16K	192
	SAMSum (SS)	rouge	≤16K	152
	TREC (TC)	accuracy	≤16K	200
	TriviaQA (TQ)	accuracy	≤16K	120
	LCC (LC)	code similarity	≤16K	488
	RepoBench-P (RP)	code similarity	≤16K	320
Commonsense	PIQA	accuracy		All
	HellaSwag (Hella.)	normalized accuracy		All
	WinoGrande (Wino.)	accuracy		All
	ARC-e	accuracy		All
	ARC-c	normalized accuracy		All
	Wikitext (Wiki.)	perplexity		All
	LAMBADA (LMB.)	perplexity, accuracy		All
	FDA	recall		All
	SWDE	recall		All

Table E.1: Training hyperparameters.

Parameter	Value	Parameter	Value
Actor gradient clip	0.2	Learning rate	10^{-6}
Mid-Train batch size	128	Adam (β_1, β_2)	$(0.9,\ 0.999)$
Post-Train batch size	64	Weight decay	0.01
Test-time Train batch size	8	Sampling temperature τ	1.0
Mid-Train PPO mini-batch size	32	Max prompt length	16,384
Post-Train PPO mini-batch size	16	Entropy loss coefficient	0
Test-time Train PPO mini-batch size	4	KL loss coefficient	0
Mid-Train λ_{RL}	0.2	n (rollouts / position)	1
Post-Train λ_{RL}	0.2	k (rollout length)	5
Test-time Train λ_{RL}	0.4	c (chunks / sequence)	8
Mid-Train λ_{SFT}	1.0	Mid-Train reward	R^{φ}
Post-Train λ_{SFT}	1.0	Post-Train reward	R^{hybrid}
Test-time Train λ_{SFT}	1.0	Test-time Train reward	R^{binary}

compatible with the test-time training framework. We follow the official metrics and configurations from Language Model Evaluation Harness [42] in our implementations.

G Additional Analysis

Additional Baselines. We compare REFINE against baseline methods that aim to provide sequence-level loss for standard transformer models. Motivated by Fang et al. [37], we experiment with a variant of pure SFT, where the CE loss of k tokens at high-entropy positions sampled as in Section 3.2 are up-weighted by a factor of two (we denote this method as SFT*). We also implement architectural multi-token prediction method motivated by Gloeckle et al. [24]. During training, we clone the last

Table G.1: **Performance on Long-Context Retrieval Tasks with Additional Baselines.** We evaluate mid-trained (**MidTr**) models on the NIAH tasks in RULER at 4K, 8K, and 16K context lengths. Highest scores in each category are highlighted in **bold**.

Model	MidTr	RULER S-NIAH				RULER MK-NIAH				RULER MQ-NIAH				RULER MV-NIAH			
		4K	8K	16K	Avg	4K	8K	16K	Avg	4K	8K	16K	Avg	4K	8K	16K	Avg
LaCT 760M	-	98.8	91.2	95.8	95.3	70.2	44.8	24.2	46.2	39.2	24.9	17.4	27.1	40.6	27.0	17.7	28.4
	SFT	98.4	90.8	97.6	95.6	**70.6**	45.0	24.2	46.6	**40.7**	25.2	17.4	27.8	**41.7**	26.3	17.7	28.5
	REFINE	**99.0**	93.0	96.8	96.3	70.4	**46.0**	**26.6**	**47.7**	40.5	**25.8**	**18.0**	**28.1**	41.2	**29.5**	**18.6**	**29.8**
	SFT*	98.8	92.6	**98.4**	96.6	69.4	44.8	24.2	46.1	41.2	25.5	17.2	28.0	42.0	27.1	17.8	29.0
	MTP	98.6	**95.0**	98.0	**97.2**	70.0	46.8	24.0	46.9	43.6	25.4	17.1	28.7	41.9	26.0	17.9	28.6
DeltaNet 1.3B	-	**100.0**	**100.0**	**100.0**	**100.0**	23.6	17.8	3.4	14.9	27.7	3.9	1.8	11.1	23.9	5.4	2.0	10.4
	SFT	**100.0**	**100.0**	**100.0**	**100.0**	23.8	19.2	7.8	16.9	33.2	16.8	3.5	17.8	28.3	19.3	3.7	17.1
	REFINE	**100.0**	**100.0**	99.8	99.9	**25.0**	**21.4**	**8.8**	**18.4**	**37.3**	**18.8**	3.6	**19.9**	**29.0**	**20.4**	**4.0**	**17.8**
	SFT*	**100.0**	**100.0**	99.8	99.9	24.2	20.6	6.2	17.0	32.9	15.7	**4.4**	17.7	28.5	19.4	3.4	17.1
	MTP	**100.0**	**100.0**	**100.0**	**100.0**	24.4	17.2	2.6	14.7	29.2	2.3	1.9	11.1	24.3	2.4	1.7	9.5

Table G.2: **Performance on Multi-Doc QA Tasks with Additional Baselines.** We evaluate various training strategies during mid-training (**MidTr**), post-training (**PostTr**), and test-time training (**TTT**) for multi-document question and answering tasks. First and second highest scores on each task are highlighted in **bold** and underline, respectively.

	MidTr	PostTr	TTT	RULER SQuADQA				RULER HotpotQA			
				4K	8K	16K	Avg	4K	8K	16K	Avg
LaCT 760M	-	-	-	17.5	14.0	6.5	12.7	20.5	14.5	12.0	15.7
	SFT	-	-	19.5	14.0	6.0	13.2	21.5	15.0	12.0	16.2
	REFINE	-	-	20.5	15.5	8.0	14.7	23.0	16.0	12.5	17.2
	REFINE	SFT	-	30.5	20.0	7.5	19.3	25.0	16.5	11.5	17.7
	REFINE	Nested SFT	-	38.0	20.0	7.5	21.8	24.0	16.0	12.5	17.5
	REFINE	Nested REFINE	-	43.5	24.5	8.5	25.5	27.0	19.5	13.0	19.8
	REFINE	Nested REFINE	SFT	43.5	21.5	8.5	24.5	26.5	24.0	13.0	21.2
	REFINE	Nested REFINE	REFINE	**45.5**	**25.5**	10.0	**27.0**	**28.5**	**25.5**	**15.0**	**23.0**
	SFT*	-	-	22.0	13.0	6.5	13.8	20.0	13.5	13.0	15.5
	SFT*	Nested SFT*	-	35.5	23.5	**11.0**	23.3	24.5	19.5	14.5	19.5
	SFT*	Nested SFT*	SFT*	35.5	24.5	8.0	22.7	25.5	21.5	14.0	20.3
	MTP	-	-	20.5	13.0	7.0	13.5	20.0	17.0	9.0	15.3
	MTP	Nested MTP	-	34.0	23.0	8.0	21.7	25.5	19.0	12.5	19.0
	MTP	Nested MTP	MTP	33.5	20.5	9.5	21.2	24.5	22.0	13.0	19.8
DeltaNet 1.3B	-	-	-	11.0	5.0	2.0	6.0	10.0	3.0	2.5	5.2
	SFT	-	-	9.0	6.0	3.0	6.0	9.0	8.0	4.5	7.2
	REFINE	-	-	10.0	6.5	4.0	6.8	12.0	9.5	5.5	9.0
	REFINE	SFT	-	11.0	7.0	5.0	7.7	14.0	11.0	6.5	10.5
	REFINE	Nested SFT	-	12.0	8.0	5.0	8.3	16.5	10.0	7.5	11.3
	REFINE	Nested REFINE	-	14.0	10.5	6.5	10.3	15.0	11.0	**8.5**	11.5
	REFINE	Nested REFINE	SFT	16.5	10.5	6.5	11.2	16.0	10.5	6.5	11.0
	REFINE	Nested REFINE	REFINE	**17.5**	**12.5**	**7.0**	**12.3**	**18.0**	**13.5**	8.0	**13.2**
	SFT*	-	-	9.0	5.5	3.5	6.0	12.5	7.5	4.5	8.2
	SFT*	Nested SFT*	-	13.5	8.5	5.5	9.2	12.5	8.5	6.5	9.2
	SFT*	Nested SFT*	SFT*	15.0	7.5	6.0	9.5	15.5	10.5	9.0	11.7
	MTP	-	-	9.5	4.0	2.0	5.2	9.5	5.5	3.0	6.0
	MTP	Nested MTP	-	12.5	7.0	4.0	7.8	11.5	9.5	6.5	9.2
	MTP	Nested MTP	MTP	12.0	5.0	3.5	6.8	15.5	12.0	7.0	11.5

layer into $k = 4$ heads (we choose $k = 4$ because it divides 16384, our training context length) and train the model with normal CE loss. Each head is designed to predict a single token within a contiguous block of $k = 4$ tokens. During inference, we discard all prediction heads except one and generate tokens autoregressively (we denote this method as MTP). Tabs. G.1 to G.3 show the performance gain of training the models with baseline methods during mid-, post-, and test-time training, as well as the original comparison of REFINE and SFT. The results show that the maximum performance gain achievable is highest for REFINE, indicating that REFINE's RL approach in providing sequence-level learning signal is most effective.

Compute Matched SFT Baselines. Each training step in REFINE takes longer than SFT (11x for DeltaNet-1.3B and 23x for LaCT-760M). The vLLM [77] engine in veRL [78] does not yet support fast weight models for rollout generation. The computational overhead should reduce once comparable infrastructure is available for fast weight models.

Table G.3: **Performance on Long-Context Tasks in LongBench with Additional Baselines.** We study the impact of the learning algorithm during mid-training (**MidTr**) and test-time training (**TTT**) on tasks with long-context. **SFT** denotes the supervised fine-tuning with next-token prediction. We evaluate on 12 tasks in LongBench, filtered for samples with at most 16K tokens.

			Single-doc QA			Multi-doc QA		Summarization		Few-shot QA			Coding		
	MidTr	TTT	NQ	QR	MF	HP	2W	QM	MN	SS	TC	TQ	LC	RP	Avg
LaCT 760M	-	-	6.5	10.5	7.2	11.7	9.8	13.6	9.2	14.2	10.5	8.0	26.7	29.7	13.1
	SFT	-	5.8	10.1	7.4	12.6	9.2	13.1	10.5	13.8	7.5	12.2	29.8	30.1	13.5
	REFINE	-	6.5	11.1	12.6	**19.6**	18.0	**14.3**	15.9	17.0	11.0	12.9	**32.9**	31.1	16.9
	REFINE	SFT	5.1	12.6	13.5	15.9	18.4	13.2	**16.2**	**17.4**	12.5	**16.2**	32.0	31.4	17.0
	REFINE	REFINE	6.7	**14.5**	**14.1**	18.4	**22.8**	13.9	15.9	**17.4**	**15.5**	11.8	32.2	**32.3**	**18.0**
	SFT*	-	4.8	11.0	8.0	18.1	9.5	13.4	11.2	14.1	11.0	12.2	28.0	30.5	14.3
	SFT*	SFT*	**7.9**	10.2	8.0	16.8	14.0	14.1	12.7	14.9	13.0	13.8	30.0	30.2	15.5
	MTP	-	4.5	9.0	6.4	17.0	6.3	13.0	11.1	13.1	15.0	13.3	28.4	29.4	13.9
	MTP	MTP	4.2	9.1	7.0	12.8	10.1	13.1	11.8	15.6	15.0	14.5	26.1	30.2	14.1
DeltaNet 1.3B	-	-	6.5	8.7	10.0	4.8	6.4	12.4	16.3	9.4	17.7	15.1	33.8	29.0	14.2
	SFT	-	5.7	9.4	8.3	9.2	8.6	14.7	16.1	15.5	22.9	15.2	33.1	29.2	15.7
	REFINE	-	6.5	9.5	10.1	**9.6**	8.7	**15.2**	15.0	16.2	25.0	21.4	**35.9**	**31.1**	17.0
	REFINE	SFT	7.2	9.6	10.5	6.8	7.2	14.9	16.2	**17.3**	28.0	16.6	34.1	29.2	16.5
	REFINE	REFINE	**7.5**	9.2	11.5	9.5	8.6	14.7	16.1	16.5	**31.5**	**24.7**	35.2	30.0	**17.9**
	SFT*	-	5.2	9.4	10.4	6.5	9.2	14.5	**16.7**	14.8	23.0	21.1	32.3	28.1	15.9
	SFT*	SFT*	5.9	**9.9**	**13.0**	5.1	**11.0**	14.1	16.2	15.7	28.0	20.6	33.8	30.5	17.0
	MTP	-	5.8	7.5	8.7	8.5	8.7	13.7	16.0	11.3	18.5	15.9	32.1	27.7	14.5
	MTP	MTP	6.7	7.7	10.2	5.7	8.1	13.7	15.9	10.8	18.0	18.2	31.8	28.3	14.6

Table G.4: **Compute-Matched Comparison of Performance on LongBench Tasks.** We study the impact of the learning algorithm during mid-training (**MidTr**) on long-context tasks. **CM-SFT** denotes compute-matched supervised fine-tuning with next-token prediction. We evaluate on 12 tasks in LongBench, filtered for samples with at most 16K tokens. Details are similar to Tab. 3.

		Single-doc QA			Multi-doc QA		Summarization		Few-shot QA			Coding		
	MidTr	NQ	QR	MF	HP	2W	QM	MN	SS	TC	TQ	LC	RP	Avg
LaCT 760M	-	6.5	10.5	7.2	11.7	9.8	13.6	9.2	14.2	10.5	8.0	26.7	29.7	13.1
	SFT (2500 steps)	**6.9**	9.5	9.1	15.2	11.4	14.1	8.3	**17.0**	**12.0**	**13.3**	27.9	**31.5**	14.7
	REFINE (100 steps)	6.5	**11.1**	**12.6**	**19.6**	**18.0**	**14.3**	**15.9**	**17.0**	11.0	12.9	**32.9**	31.1	**16.9**
DeltaNet 1.3B	-	**6.5**	8.7	10.0	4.8	6.4	12.4	16.3	9.4	17.7	15.1	33.8	29.0	14.2
	SFT (1400 steps)	5.3	8.7	**11.5**	8.6	**9.2**	14.7	**17.3**	**17.7**	**27.5**	20.1	32.3	28.1	16.8
	REFINE (100 steps)	**6.5**	**9.5**	10.1	**9.6**	8.7	**15.2**	15.0	16.2	25.0	**21.4**	**35.9**	**31.1**	**17.0**

We experiment with a compute-matched SFT baseline and compare its LongBench [8] performance with REFINE. We mid-train DeltaNet-1.3B for 1400 steps on 14x more data, and LaCT-760M for 2500 steps on 25x more data (both more generous than the 11x and 23x compute ratios) while fixing REFINE mid-training at 100 steps. Tab. G.4 shows that mid-training with REFINE still outperforms compute-matched SFT baselines, indicating that our method is more efficient after accounting for compute.

Scaling Experiments. In order to demonstrate that REFINE can be applied to larger models, we experiment with DeltaNet-2.7B, which is twice as big as DeltaNet-1.3B. Larger pre-trained LaCT models are not yet publicly available. Tab. G.5 shows that REFINE mid-training still outperforms SFT on LongBench, showing that our method can scale to larger model parameters.

Additional Analysis of Drift in Fast Weights. We repeat the fast weight drift analysis in Section 4.5 on four additional tasks in LongBench [8]. We take 20 random samples from each task and compute the Frobenius norm of the difference between the fast weight matrices for predicted and ground truth response trajectories, averaged across the layers. Fig. G.1 shows that mid-training DeltaNet-1.3B with REFINE reduces drift in fast weights across tasks compared to SFT, suggesting that NSP is beneficial for long-horizon memory stability.

We further study how the rollout length k and the number of chunks c affect fast weight drift. As shown in Tab. G.6, reducing the rollout length from $k = 5$ to $k = 3$ increases drift across most LongBench tasks. Similarly, reducing the number of chunks from $c = 8$ to $c = 4$ or $c = 2$ generally leads to higher drift. These trends align with the ablations in Section 4.6, where configurations with higher drift also achieve lower average LongBench [8] performance than the default setting of $k = 5$ and $c = 8$. Together, these results support the hypothesis that NSP improves downstream performance partly by stabilizing recurrent fast weight trajectories over long horizons.

Table G.5: **Scaling Mid-Training to DeltaNet 2.7B.** We compare SFT and REFINE mid-training on DeltaNet 2.7B using LongBench tasks. Higher is better. Best scores are highlighted in **bold**.

		Single-doc QA			Multi-doc QA		Summarization		Few-shot QA			Coding		
	MidTr	NQ	QR	MF	HP	2W	QM	MN	SS	TC	TQ	LC	RP	Avg
DeltaNet 2.7B	SFT	3.8	9.7	**11.6**	6.6	**13.2**	**14.2**	17.0	8.2	28.5	14.9	**36.5**	**31.0**	16.3
	REFINE	**8.6**	**9.9**	**11.6**	**6.8**	10.4	14.0	**17.8**	**14.0**	**36.5**	**18.5**	35.8	28.9	**17.7**

Table G.6: **Fast Weight Drift across LongBench Tasks.** We report the mean $\pm 1\sigma$ of drift measured by the Frobenius norm between fast weight memory states induced by predicted and ground-truth trajectories. Lower is better. The lowest values are **bolded**.

Model	MidTr	k	c	QMSum	SAMSum	QASPER	MultiNews	LCC	MultiFieldQA-en
DeltaNet 1.3B	REFINE	5	8	$\mathbf{8.322 \pm 0.135}$	$\mathbf{4.134 \pm 0.112}$	$\mathbf{4.024 \pm 0.094}$	6.145 ± 0.136	$\mathbf{2.938 \pm 0.191}$	$\mathbf{2.828 \pm 0.146}$
	REFINE	3	8	8.416 ± 0.133	4.171 ± 0.104	4.053 ± 0.096	6.085 ± 0.139	2.963 ± 0.195	2.886 ± 0.159
	REFINE	5	4	8.446 ± 0.136	4.186 ± 0.117	4.048 ± 0.096	$\mathbf{6.076 \pm 0.135}$	2.952 ± 0.193	2.858 ± 0.162
	REFINE	5	2	8.560 ± 0.133	4.233 ± 0.106	4.088 ± 0.096	6.177 ± 0.153	2.958 ± 0.193	2.898 ± 0.155

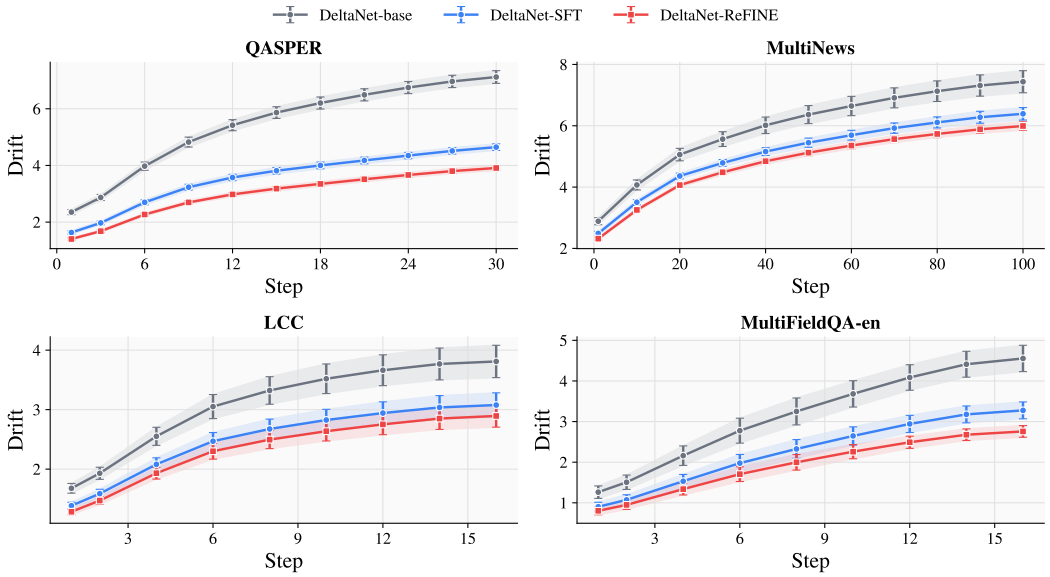


Figure G.1: **Additional Analysis on Drift in Fast Weights.** We measure drift as the Frobenius norm between the fast weight memory matrices induced by predicted and ground truth response trajectories, averaged across layers. Mid-training DeltaNet-1.3B with REFINE lowers the drift on various tasks in LongBench [8] more than SFT. Error bars show $\pm 1\sigma$ across 20 independent generations.

Entropy Distribution. We report the NTP entropy distribution of a randomly selected sample in order to illustrate the effects of entropy-based token selection in REFINE (Fig. G.2). We find that there are no index-dependent patterns in the distribution, which justifies per-chunk target token sampling weighted by entropy.

Reward Distribution. In order to investigate the stability of mid-training with REFINE, we track the cosine similarity reward across training steps for different rollout lengths ($k = 3, 5, 7$), as shown in Fig. G.3. We find that the reward distribution remains stable throughout training. However, as rollout length increases, both the mean and variance of the reward decrease, suggesting that the learning signal may lose sharpness for larger k.

Performance on Short-Context Tasks. We investigate whether enhancing long-context handling capabilities of fast weight models leads to degradation on out-of-distribution tasks such as short-context in-context retrieval and commonsense reasoning. We evaluate mid-trained models on 9 relevant short-context tasks with lm-evaluation-harness [42]: PIQA [79], HellaSwag (Hella.) [80], WinoGrande (Wino.) [81], ARC-easy (ARC-e), ARC-challenge (ARC-c) [82], Wikitext (Wiki.) [83], LAMBADA (LMB.) [84], FDA [85], and SWDE [86]. Tab. G.7 shows that REFINE sustains

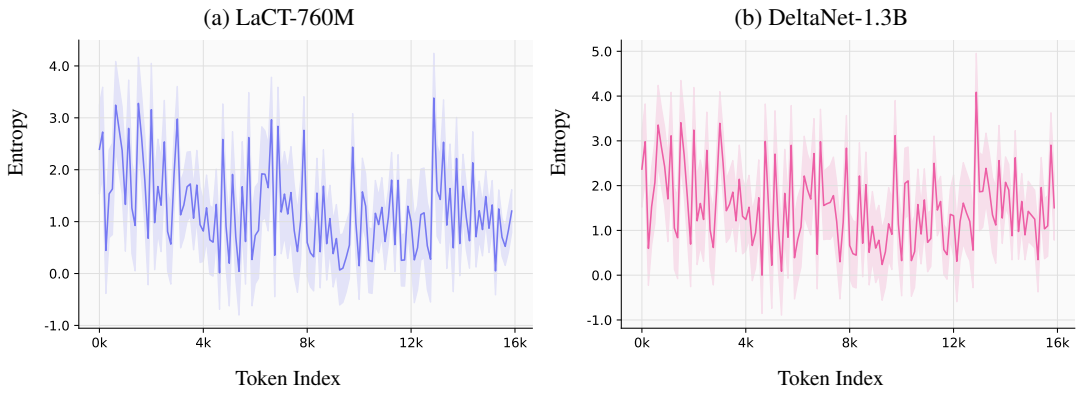


Figure G.2: **NTP Entropy Distribution.** We compute the token-level NTP entropy of a randomly selected sample using LaCT-760M **(a)** and DeltaNet-1.3B **(b)**. We use the same sample to extract the entropy distribution from both models.

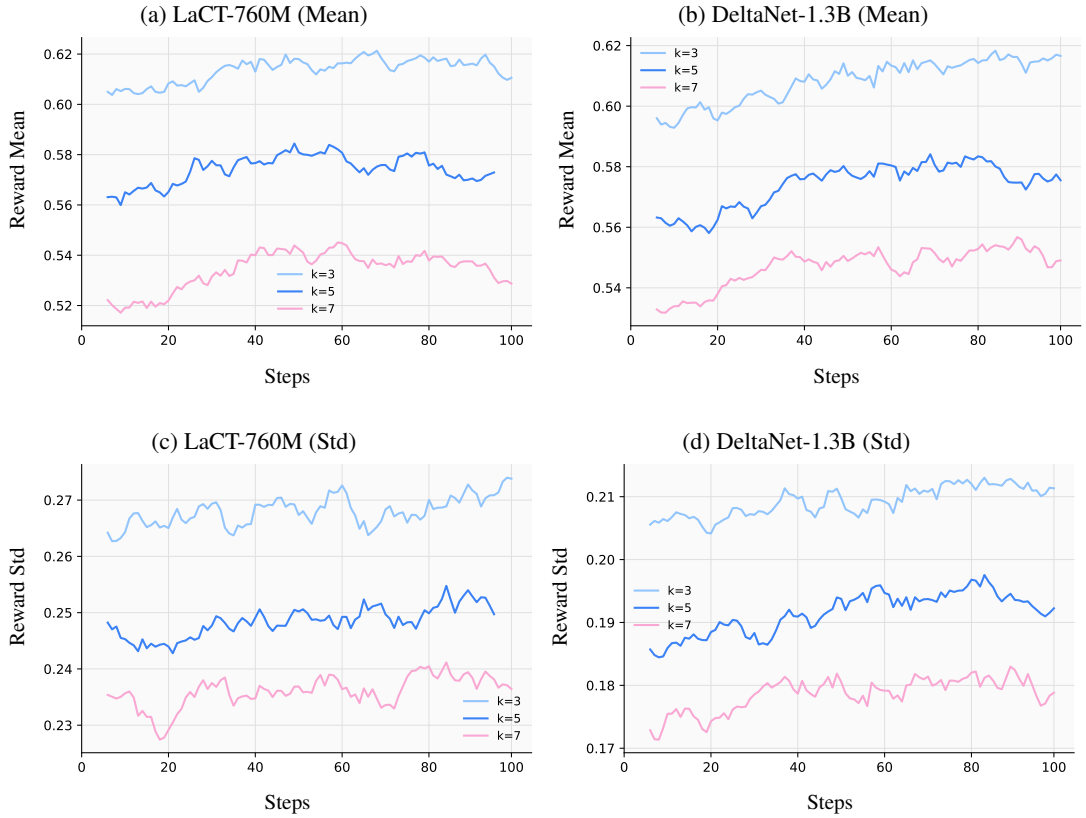


Figure G.3: **Reward Distribution.** We report the mean and standard deviation of the cosine similarity reward during mid-training for different values of k. As the rollout length increases, the mean reward (**a**, **b**) decreases and the standard deviation (**c**, **d**) also decreases.

performance in these tasks, suggesting that NSP complements NTP without inducing catastrophic forgetting [87].

Impact of Reward Functions on TTT. REFINE uses binary exact match reward $\mathcal{R}^{\text{binary}}$ during test-time for offline adaptation of the model before generating the response. We repeat TTT with cosine similarity reward instead and report the average score on LongBench tasks (Tab. 4). Tab. G.8 shows that the binary reward is optimal for TTT, but the cosine similarity reward also performs well, higher than pure SFT for TTT.

Table G.7: **Performance on Short-Context Tasks.** We evaluate mid-trained models on short-context benchmarks to verify that REFINE does not cause catastrophic forgetting.

Models	Wiki ppl ↓	LMB. ppl ↓	FDA recall ↑	SWDE recall ↑	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	Avg
LaCT-760M	20.8	29.8	36.7	66.0	32.4	67.5	41.5	53.4	48.4	27.6	45.1
+SFT MidTr	20.7	28.9	37.2	66.2	32.9	67.4	41.4	52.5	48.4	27.1	45.0
+REFINE MidTr	20.8	30.2	36.1	66.5	32.3	67.3	41.4	52.8	48.8	27.5	45.0
DeltaNet-1.3B	16.7	14.7	18.0	54.4	43.0	70.8	50.5	53.7	58.3	25.9	50.4
+SFT MidTr	16.8	15.3	17.7	54.5	42.7	71.1	50.1	53.8	58.4	26.4	50.4
+REFINE MidTr	16.8	15.6	17.5	54.5	42.2	71.1	50.2	53.6	58.1	26.2	50.2

Table G.8: **Impact of reward function on TTT.** We compare different TTT reward strategies on 12 tasks in LongBench: binary exact match between predicted and ground truth completion ($\mathcal{R}^{\text{binary}}$), and cosine similarity of the hidden states of predicted and ground truth completions ($\mathcal{R}^{\varphi}$).

	MidTr	TTT	MidTr Reward	TTT Reward	Avg. Score
LaCT-760M	REFINE	-	$\mathcal{R}^{\varphi}$	-	16.9
	REFINE	SFT	$\mathcal{R}^{\varphi}$	-	17.0
	REFINE	REFINE	$\mathcal{R}^{\varphi}$	$\mathcal{R}^{\varphi}$	17.5
	REFINE	REFINE	$\mathcal{R}^{\varphi}$	$\mathcal{R}^{\text{binary}}$	18.0
DeltaNet-1.3B	REFINE	-	$\mathcal{R}^{\varphi}$	-	17.0
	REFINE	SFT	$\mathcal{R}^{\varphi}$	-	16.5
	REFINE	REFINE	$\mathcal{R}^{\varphi}$	$\mathcal{R}^{\varphi}$	17.6
	REFINE	REFINE	$\mathcal{R}^{\varphi}$	$\mathcal{R}^{\text{binary}}$	17.9

H Qualitative Examples

We provide qualitative examples of cosine similarity reward assignment during mid-training in Tab. H.1. We randomly sample prefixes from the mid-training dataset and generate four $k = 5$ continuations each using the pre-trained LaCT-760M model. The cosine similarity reward is designed to capture the semantic similarity between the predicted and ground truth continuations. The highest reward values for each example are highlighted in bold. The examples demonstrate that the reward effectively captures semantic similarity beyond exact lexical matching. For instance, in example 2, "enjoyed every minute of it" achieves near-perfect alignment (0.961) with "loved every minute of it", while semantically divergent predictions like "would not recommend it at" receive much lower scores (0.463). Similarly, example 6 shows sensitivity to mathematical concepts, where "is a convergent integral" receives a high score (0.758) for preserving the convergence concept from "is also convergent according", while less relevant predictions such as "shall not be less than" receive lower rewards (0.512). These examples illustrate that the cosine similarity reward provides meaningful learning signals for training the model to generate semantically coherent continuations.

Table H.1: Qualitative examples of cosine similarity reward assignment during mid-training. GT denotes the ground truth continuation, and P1–P4 denote the four predicted continuations generated by the model.

Example	Type	Text	Cosine Reward
Ex. 1	**GT**	*very laid back lodge*	–
	P1	perfect hill	0.508
	P2	unique hotel to spend a	0.498
	P3	great place to stay	**0.535**
	P4	top-notch l	0.418
Ex. 2	**GT**	*loved every minute of it*	–
	P1	would not recommend it at	0.463
	P2	were so impressed by this	0.566
	P3	enjoyed every minute of it	**0.961**
	P4	skied daily. Excell	0.461
Ex. 3	**GT**	*screen washer fluid are*	–
	P1	screen washer software are	**0.922**
	P2	screen wipers are changed	0.641
	P3	screen washing and dipping	0.555
	P4	screen washer fluids	0.824
Ex. 4	**GT**	*bound on the predictive*	–
	P1	bound on the trace of	**0.840**
	P2	-ranked trace-	0.535
	P3	bound on	0.594
	P4	bound on P(	0.582
Ex. 5	**GT**	*continues to grow; and*	–
	P1	reflects wave after wave of	0.484
	P2	stretches into space.	**0.531**
	P3	heats ocean waters a	0.461
	P4	cushions us all from	0.500
Ex. 6	**GT**	*is also convergent according*	–
	P1	is a convergent integral	**0.758**
	P2	shall not be less than	0.512
	P3	must have a value,	0.574
	P4	is convergent. Because	0.578

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's contribution in exploring the effectiveness of next-sequence prediction training objective in fast weight language models.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: Limitations of our method are discussed in section 5.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: This paper does not include theoretical results.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The full training and evaluation configurations as well as datasets are detailed in Appendix Sections D to F.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code and documentation will be provided via supplementary material.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The full training and evaluation configurations as well as datasets are detailed in Appendix Sections D to F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We provide error bars for the drift experiments and mid-training validation metrics. We do not provide error bars for post- and test-time training results due to limitations in compute.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The computational requirements are specified in Appendix §E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Ethics statement on broader societal impact is discussed in Appendix §B.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

Justification: The paper does not release data or models that have a high risk of misuse.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the original papers that produced relevant code and datasets, as well as their licenses in Appendix Section D.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide thorough documentation alongside the code assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This paper does not involve crowdsourcing experiments with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This paper does not involve research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development of this paper does not involve LLMs.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.