Context as Low-Rank Weights: Bounded Parametric Dynamic Memory for Long-Context Modeling

Xiangyu Zhang
Nanjing University of Aeronautics and Astronautics
zhangx1angyu@nuaa.edu.cn

Yu Zhou*
Nanjing University of Aeronautics and Astronautics
zhouyu@nuaa.edu.cn

Taolue Chen
Birkbeck, University of London
t.chen@bbk.ac.uk

Abstract

Transformers rely on a growing key–value (KV) cache to store context, causing linear memory growth and limited generalization beyond the training window. We show that contextual influence can instead be viewed as updates to a low-dimensional subspace in the model's weight space, meaning context can be represented as a bounded, low-rank parameter update rather than token-level storage. Motivated by this, we propose PARADE, which internalizes context as parametric memory. It compresses past inputs into a fixed set of weight-space basis vectors and composes them via query-dependent routing into per-query weight updates, enabling streaming inference with constant memory usage independent of sequence length. Experiments show that PARADE matches full attention on long-context tasks and outperforms efficient baselines, especially when relevant information lies far beyond the attention window, while improving as context exceeds the model's training length.

1 Introduction

Transformers [Vaswani et al., 2017] exhibit a remarkable ability to integrate model knowledge with contextual information for prediction and generation. At each step, learned weights extract and transform signals from the input sequence, enabling the use of both prior knowledge and context.

Model knowledge and context are implemented differently: knowledge is encoded in model weights (e.g., MLP layers [Geva et al., 2022]) and applied uniformly, while contextual information is stored externally in a transient KV cache and accessed via attention. This creates separate computational pathways for parametric and contextual information. Although effective, its reliance on explicit, token-level representations for contextual information introduces practical limitations. The KV cache grows linearly with sequence length, incurring significant memory and computation costs [Ge et al., 2023, Zhang et al., 2025], and the model's usable context is bounded by the lengths seen during training, limiting generalization to longer inputs [Anil et al., 2022, Press et al., 2021, Su et al., 2024].

In contrast, model knowledge encoded in the weights is represented in a highly compressed form through superposition [Elhage et al., 2022], where multiple semantic features are encoded as overlapping directions within a shared parameter space, rather than as explicit token-level representations. This allows the model to represent a large number of features with a limited number of parameters, achieving a form of implicit compression. Moreover, model knowledge is accumulated across the entire training corpus through gradient updates that write into this shared parameter space, integrating

*Corresponding author.

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

information from independently processed sequences into a unified representation. As a result, knowledge acquisition is decoupled from the length of individual sequences [Allen-Zhu and Li, 2023].

This raises a natural question: *As context becomes increasingly long, can it be represented within the model's parameters rather than being stored explicitly alongside them?* Existing work has suggested that in-context learning (ICL) can be interpreted as performing implicit weight updates within the model parameter [Dherin et al., 2025, Goldwaser et al., 2025]. In particular, they show that each token in the context induces a query-dependent rank-1 update to the model's weights, providing a mechanistic explanation of how contextual information influences model outputs.

However, this line of work primarily focuses on explaining model's computation rather than their practical applicability in real-world settings. In practice, these methods require storing and processing a sequence of per-token updates, analogous to maintaining a growing memory. Moreover, the analysis is typically restricted to the ICL setting. It remains unclear whether this weight-update perspective extends to general long-range context in standard language modeling, where the effective rank of the accumulated updates may far exceed the model's intrinsic dimensionality.

In this paper, we empirically observe that, although context accumulates over thousands of tokens, the effective rank of the resulting updates rapidly saturates at a small value, indicating that contextual influence is intrinsically low-dimensional. Surprisingly, even over contexts of up to 32K tokens, the induced weight-space updates remain within a 300-dimensional subspace that captures over 90% of the variation. Moreover, this subspace emerges early in the sequence and remains remarkably stable as it grows. Subsequent tokens contribute primarily within this subspace, while newly introduced directions account for only a small fraction of the total variation. These findings suggest a different view of context. Rather than functioning as an ever-expanding memory, it can be viewed as a finite-dimensional update whose structure is formed early and incrementally refined over time.

Building on this insight, we introduce PARADE (PARAmetric Dynamic mEmory), a plug-in framework for pretrained Transformers that internalizes context as parametric memory by representing contextual information through low-rank updates in weight space rather than storing it in token-level representations. Instead of maintaining a growing KV cache, PARADE retains only a finite local attention window, while incrementally compiling past context into a set of compact, reusable memory slots stored as updates in the model's parameter space. During inference, these representations are dynamically composed into query-conditioned weight updates, enabling the model to incorporate relevant contextual information directly into its computation.

By shifting context from representation space to parameter space, this design fundamentally changes how contextual information is accumulated and utilized, namely, it is progressively encoded into, updated within, and retrieved from a compact set of parameters. Compared to token-level KV storage, PARADE provides a more efficient representation of long-range context and enables constant-memory streaming inference beyond the model's native context window, as demonstrated in our experiments.

Contributions.

1. We present an empirical study of the influence of context on model computation, showing that context-induced updates lie in a low-dimensional subspace that is established early and remains stable throughout the sequence (Section 2).
2. We propose a framework PARADE that encodes context as a fixed-capacity set of memory slots in weight space, enabling long context modeling without increasing memory cost (Section 3).
3. Experiments on both long-context (Qwen-2.5-0.5B) and short-context (LLaMA-2-7B) base models show that PARADE matches Full Attention on long-context benchmarks while maintaining constant memory, outperforms efficient attention baselines, and extrapolates beyond the base model's native context window (Section 4).

2 Context as Implicit Updates to MLP Weights

We interpret contextual information as inducing updates to MLP weights rather than merely storing activations in the KV cache, and empirically show that these updates lie in a stable low-dimensional subspace, indicating that long-range context can be represented compactly.

2.1 Context as Weight-Space Updates

Linear case. Consider a single-layer Transformer with linear attention. It consists of a linear MLP with an up-projection $W_{\text{up}} \in \mathbb{R}^{d_{\text{mid}} \times d}$ and a down-projection $W_{\text{down}} \in \mathbb{R}^{d \times d_{\text{mid}}}$, together with a residual connection. The attention output at position t aggregates contextual tokens as $\text{Attn}(x_t) = \sum_{s<t} v_s k_s^\top x_t = W_t x_t$, where $W_t = \sum_{s<t} v_s k_s^\top$ is the fast weight matrix [Schlag et al., 2021], where $k_s, v_s \in \mathbb{R}^d$ denote the key and value representations of the contextual token at position s, and. After applying the residual connection $h_t = (I + W_t)x_t$, the block output becomes

$$y_t \;=\; W_{\text{down}}\, W_{\text{up}}\, (I + W_t)\, x_t \;=\; W_{\text{down}}\, (W_{\text{up}} + \Delta W_t)\, x_t, \quad \Delta W_t \;=\; W_{\text{up}}\, W_t.$$

This formulation shows that the effect of context can be absorbed into an additive update ΔW_t to the MLP up-projection.

Nonlinear case. Under softmax attention and nonlinear activations, the equivalence no longer holds directly. Dherin et al. [2025] and Goldwaser et al. [2025] resolve this via a query-dependent weight update construction. Let $A(C, x)$ denote the attention output at position x under context C, and $A(x)$ the corresponding output without context. The key result is that the rank-1 matrix

$$\Delta W(C, x) \;=\; \frac{W_{\text{up}} \left[A(C, x) - A(x)\right] A(x)^\top}{\|A(x)\|^2}$$

satisfies an exact equivalence $T_W(C, x) \;=\; T_{W + \Delta W(C,\, x)}(x)$, where T_W denotes the Transformer block mapping. That is, processing x with context C under weights W is exactly equivalent to processing x alone under the modified weights $W_{\text{up}} + \Delta W(C, x)$.

Although these results show that contextual influence can be represented in weight space in ICL, they do not eliminate the fundamental scaling problem of long-context modeling. In the linear setting, the effective rank of accumulated updates grows with context length. In the nonlinear setting, query-dependent rank-1 updates must be maintained separately for each token. Thus, despite the different formulations, both approaches still require memory and computation that scale with sequence length, analogous to the KV cache.

2.2 Low-Rank Structure of Contextual Weight Updates

Section 2.1 shows that contextual influence can be represented in weight space. We extend this view to the long-context regime by empirically characterizing how context-induced updates accumulate as context grows.

Object of study. Our goal is to characterize how long-range context influences the MLP pathway, where implicit weight updates manifest. For a query position s, let a_s^{full} denote the attention output under full attention and a_s^{local} the output under a local window of $W = 128$ tokens. Their difference isolates the contribution of long-range context.

We measure how this difference perturbs the MLP input by projecting it through the MLP up-projection:

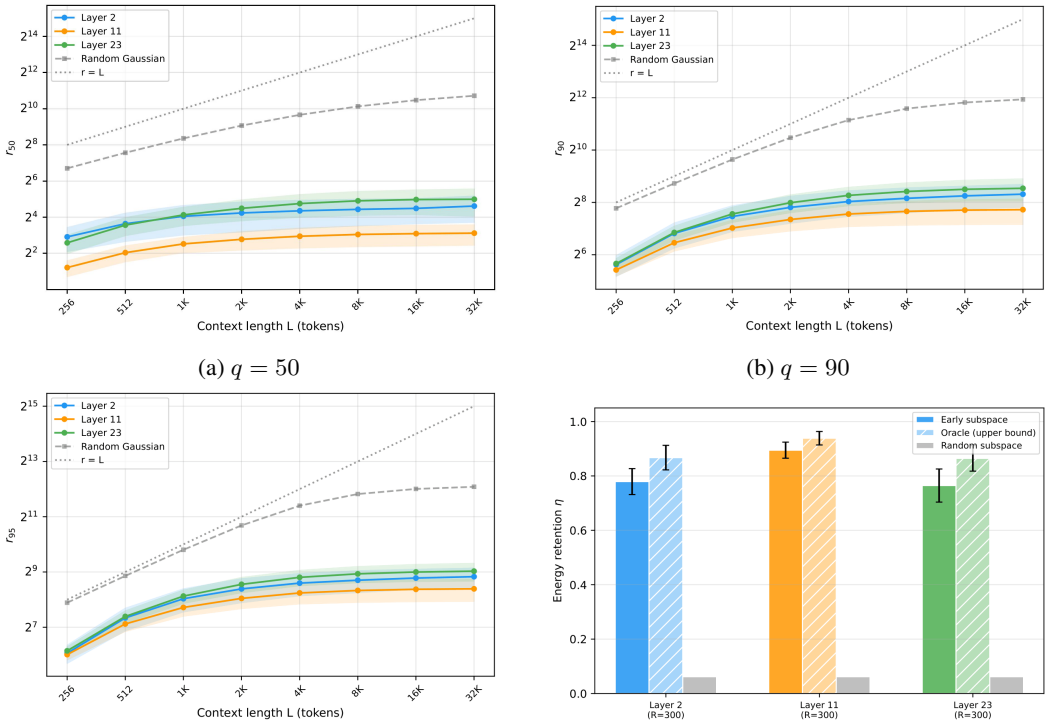
$$\delta h_s \;=\; W_{up} \cdot \text{Norm}(x_s + a_s^{\text{full}}) \;-\; W_{up} \cdot \text{Norm}(x_s + a_s^{\text{local}}) \;\in\; \mathbb{R}^{d_{mid}},$$

where $\text{Norm}(\cdot)$ denotes the same normalization function used by the base model, x_s is the hidden representation at position s, and δh_s measures the change in the MLP input induced by long-range context. Intuitively, δh_s captures the additional signal injected into the MLP by tokens outside the local attention window.

We refer to the collection of δh_s as the *footprint* of context C in weight space, which provides an empirical proxy for the direction of the induced weight-space update.

Setup. We use the pretrained Qwen2.5-0.5B [Qwen et al., 2025] as the base model. We sample 100 documents of length 32K tokens from a mixture of PG19 [Rae et al., 2019], ArXiv [Cohan et al., 2018], and The Stack [Kocetkov et al., 2022], and perform two forward passes per document (full attention and local-window attention). We extract footprint vectors at three uniformly spaced layers, reporting representative shallow (Layer 2), middle (Layer 11), and deep (Layer 23) layers.

For each context, the footprint vectors across L query positions form a matrix $U \in \mathbb{R}^{L \times d_{mid}}$, where each row corresponds to a footprint vector at a different position. We perform singular value



(a) $q = 50$ (b) $q = 90$

(c) $q = 95$ (d) Subspace generalization

Figure 1: Empirical geometry of context's implicit weight updates in Qwen2.5-0.5B. **(a)–(c)** Effective rank r_q versus context length L at energy thresholds $q = 50, 90, 95$. Solid colored curves: representative layers (mean $\pm$ std over 100 contexts). Dashed gray: random Gaussian footprints (architectural reference, $d_{mid} = 4{,}864$). Dotted gray: hypothetical $r = L$ scaling. **(d)** Energy retention η of footprints at positions $4\text{K}-32\text{K}$ when projected onto the rank-$R = 300$ subspace fitted to the first 4K tokens. Solid bars: empirical η. Hatched bars: oracle upper bound (subspace fitted to the late footprints). Gray: random R-dimensional subspace.

decomposition (SVD) on U to obtain its singular value spectrum $\{\sigma_i\}$, which characterizes the distribution of energy across different directions in the footprint space. We define the effective rank r_q as the least number of singular values required to capture $q\%$ of the total spectral energy.

2.2.1 Finding 1: Footprint dimensionality is intrinsically bounded

Figure 1a–1c shows that the effective dimensionality of the footprint saturates early as the context length increases from $L = 256$ to 32K. Even at $L = 32$K, the mean values of r_{50} and r_{90} across layers are only 21.7 and 300.4, respectively, remaining far below both the context length and the architectural dimension $d_{mid} = 4{,}864$. Beyond $L \approx 2$K, the dimensionality increases by less than 30% despite a $16\times$ increase in context length, whereas random Gaussian vectors reach $r_{90} \approx 4000$.

These results suggest that contextual influence accumulates within a stable low-dimensional subspace rather than along independent per-token directions, indicating that long-range context admits a compact bounded-rank representation in weight space.

2.2.2 Finding 2: Context accumulates through updates within low-dimensional subspaces

Finding 1 shows that the footprint subspace has bounded dimensionality. We now investigate how subsequent updates interact with and refine the representation established by earlier context as the sequence grows.

To this end, we construct a rank-R subspace $\mathcal{S}_{\text{early}}$ from the leading right singular vectors of the footprints induced by the first $L_{\text{early}} = 4$K tokens. We then measure how much of the energy of

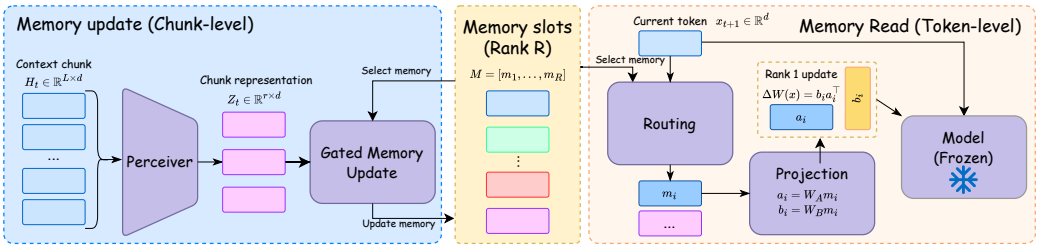


Figure 2: Overview of PARADE.

subsequent footprints, from positions 4K to 32K, is preserved when projected onto this subspace:

$$\eta = \mathbb{E}_{s>L_{\text{early}}}\left[\frac{\|P_{\mathcal{S}_{\text{early}}}\delta h_s\|^2}{\|\delta h_s\|^2}\right].$$

We compare this quantity with two reference subspaces. The first one is an oracle subspace obtained from the late footprints themselves; the second one is a random R-dimensional subspace.

Figure 1d shows that at $R = 300$, the early subspace retains 78% of the energy of later footprints, compared to only 6% for a random subspace, and remains close to the oracle (89% vs. 93% at Layer 11). This indicates that later contextual updates largely remain within the subspace established early in the sequence. Subsequent updates therefore operate primarily within this fixed subspace, refining existing representations rather than introducing fundamentally new directions.

Summary. The empirical findings suggest that long-range context can be represented as a bounded low-rank update in weight space rather than an ever-growing sequence of token-level activations. This motivates a long-context architecture that incrementally compiles context into fixed-capacity parametric memory instead of a dynamically expanding KV cache.

3 Method

In this section, we introduce PARADE, a framework that represents context as bounded *parametric memory* rather than an explicit token-level cache. Motivated by the findings in Section 2, which show that context-induced updates lie in a low-dimensional subspace and accumulate through continual refinement, PARADE models context as a bounded set of parametric updates in weight space. As shown in Figure 2, PARADE processes context in a streaming manner, maintaining a fixed set R of memory slots, each of which is a rank-one contribution to the MLP weight update, cf. Section 3.1). The slots are incrementally updated as new context arrives and dynamically composed at the chunk (a contiguous segment of tokens) level.

3.1 Memory Representation

We represent the parametric memory as a fixed set of R memory slot $m_i \in \mathbb{R}^d$, $i = 1, \ldots, R$, forming a bounded set of components that span the low-dimensional subspace of context-induced updates. (Recall d denotes the hidden dimension of the model.) Each m_i generates a pair of directions in the input and output spaces of the MLP up-projection $W_{\text{up}} \in \mathbb{R}^{d_{\text{mid}}\times d}$ via two learned projection matrices $W_A \in \mathbb{R}^{d\times d}$ and $W_B \in \mathbb{R}^{d_{\text{mid}}\times d}$:

$$a_i = W_A\, m_i \in \mathbb{R}^d, \quad b_i = W_B\, m_i \in \mathbb{R}^{d_{\text{mid}}}.$$

Each m_i thus defines a rank-one update $b_i a_i^\top \in \mathbb{R}^{d_{\text{mid}}\times d}$ to model weights, and collectively, $\{b_i a_i^\top\}_{i=1}^R$ forms a structured, compositional representation of contextual memory in weight space.

3.2 Memory Update

Chunk-Level Processing. PARADE operates at the granularity of chunks rather than individual tokens. Since neighboring tokens often contain highly redundant information, token-level updates are unnecessary and inefficient. Chunk-level processing therefore enables more stable and efficient memory updates.

Specifically, PARADE maintains a sliding context window of size N. Once the window is full, the oldest $L = N/2$ tokens are evicted and their representations are compressed into parametric memory. The remaining L token states, together with the updated memory, are then used to guide subsequent generation.

Given the hidden representations $H_t \in \mathbb{R}^{L\times d}$ of the evicted chunk produced by the base model, we first extract a compact semantic representation of the chunk. Since token-level representations are often highly redundant, it is sufficient to summarize the chunk using a much smaller set of vectors. To this end, following the Perceiver architecture [Jaegle et al., 2021], we aggregate H_t into $r \ll L$ summary vectors via cross-attention with a set of learned queries, where each query selectively attends to and compresses a subset of relevant information from the chunk:

$$Z_t = \mathrm{CrossAttn}(Q_{\mathrm{compress}}, ; H_t) \in \mathbb{R}^{r\times d},$$

where $Q_{\mathrm{compress}} \in \mathbb{R}^{r\times d}$ are learned compression queries.

Gated Memory Update. Given the compressed chunk representation Z_t, we update the parametric memory by selectively incorporating new information into a subset of memory slots. This design is motivated by the observation that context accumulates through updates within a fixed low-dimensional subspace, where each new chunk primarily refines existing components rather than introducing entirely new directions, as introduced in Section 2. As a result, each chunk typically interacts with only a subset of the memory slots. We perform cross-attention to determine which memory slots are most relevant to the current chunk. Specifically, each memory vector m_i uses its previous state $m_i^{(t-1)}$ as a query to attend to Z_t, producing a candidate update $c_i^{(t)}$ that captures the contribution of the current chunk to that slot:

$$\tilde{c}_i^{(t)} = \mathrm{CrossAttn}(m_i^{(t-1)},\ Z_t), \quad c_i^{(t)} = \tilde{c}_i^{(t)} + \mathrm{FFN}(\tilde{c}_i^{(t)}).$$

A learned gate $g_i^{(t)} \in [0,1]$ then controls how much of the candidate is merged into each slot. To avoid catastrophic forgetting of earlier context, we restrict updates to the top-k slots by gate activation, ensuring that the remaining slots are preserved exactly across arbitrarily many update steps:

$$g_i^{(t)} = \sigma\big(W_g\,[m_i^{(t-1)} \| c_i^{(t)}]\big) \cdot 1\left[i \in \text{top-k}(\{g_j^{(t)}\}_{j=1}^{R})\right],$$
$$m_i^{(t)} = (1 - g_i^{(t)})\, m_i^{(t-1)} + g_i^{(t)}\, c_i^{(t)}.$$

3.3 Memory Read

Section 2 shows that the contextual modification is query-conditioned: each token selectively incorporates only the contextual information relevant to its own query. PARADE mirrors this structure by routing each input token to a sparse subset of memory slots at inference time, rather than applying the full memory uniformly. This preserves the selective nature of the original attention mechanism and ensures that the base model's behavior is not disturbed by irrelevant memory contributions.

Routing. We compute routing scores between the input and each slot's key direction:

$$s_i(x) = (W_Q x)^\top (W_K a_i), \quad \alpha_i(x) = \mathrm{softmax}\big(\text{top-k}(s(x))/\tau\big),$$

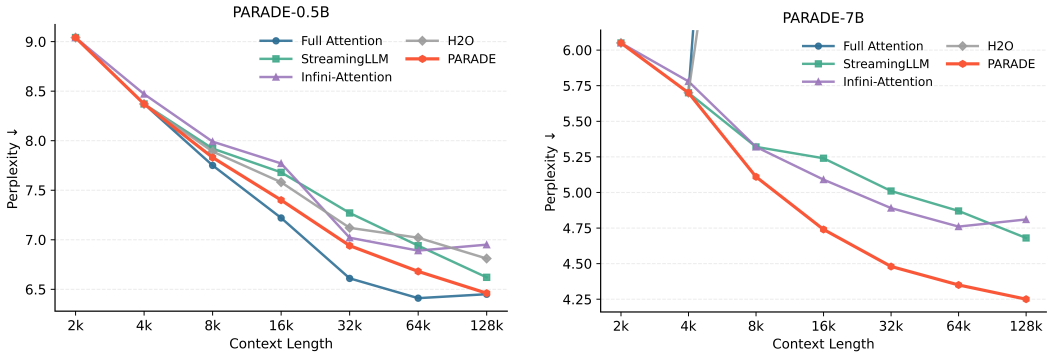
where W_Q and W_K are learned projections that decouple the routing decision from the slot's computational role, preventing the dual use of a_i as both a relevance key and a transformation direction from conflicting during optimization. Restricting composition to the top-k slots preserves the low-rank structure of the parametric memory and ensures that the knowledge encoded in the base model's MLP weights is not overwhelmed by irrelevant memory contributions.

Composition and Injection. The selected slots are composed into a token-specific weight modification mirroring the query-dependent structure of $\Delta W(x)$:

$$\Delta W(x) = \sum_{i=1}^{R} \alpha_i(x)\, b_i\, a_i^\top,$$

which is injected onto the up-projection of the base Transformer's MLP, where it interacts directly with the model's parametric knowledge: $W_{\mathrm{up}}\, x \longrightarrow (W_{\mathrm{up}} + \Delta W(x))\, x$.

Additional implementation details, including training objective (Appendix A.1), streaming KV cache management (Appendix A.2), cross-attention initialization from the base model (Appendix A.3) are provided in the Appendix.



(a) PARADE-0.5B (Qwen-2.5-0.5B, long native win.). (b) PARADE-7B (LLaMA-2-7B, 4K native win.).

Figure 3: Perplexity across context lengths for PARADE and baselines.

4 Experiments

Model and Configuration. We conduct experiments using two base models: Qwen-2.5-0.5B [Qwen et al., 2025] with a long context window and LLaMA-2-7B [Touvron et al., 2023] with a 4K context window, referred to as PARADE-0.5B and PARADE-7B, respectively. For efficiency, the LLaMA-2-7B model is loaded using 4-bit quantization during training and evaluation. These models provide complementary settings for evaluation.

The long-context model serves as a reference to assess how closely PARADE approximates full-attention performance, while the short-context model allows us to examine whether PARADE enables generalization beyond the model's native context length. Using models of different sizes further allows us to evaluate the robustness of PARADE across model scales.

The base model weights are kept frozen throughout training, and only the parameters introduced by PARADE are updated. Rather than directly modifying the model weights, these parameters generate context-dependent modifications to the W_{up} projections in the MLP layers. We use chunked streaming inference with chunk size $L = 2048$ and attention window $N = 4096$. PARADE maintains $R = 128$ memory slots with sparse top-k updates and routing ($k = 16$), and is injected into 8 evenly spaced layers.

We train on a 3B-token corpus sampled from PG19 [Rae et al., 2019], Arxiv [Cohan et al., 2018], and The Stack [Kocetkov et al., 2022]. Each training sample has a minimum length of 4K tokens, and documents longer than 128K tokens are truncated. We use AdamW with a learning rate of 5×10^{-5}, linear warmup over 1,000 steps, and linear decay.

Baselines. We compare against four representative approaches: (i) **Full Attention**, which serves as an upper bound by attending to all past tokens; (ii) **StreamingLLM** [Xiao et al., 2023], which maintains a sliding window augmented with attention sinks to support long-context generation; (iii) **H_2O** [Zhang et al., 2023], which supports long-context modeling by retaining salient information from past context along with a sliding window; and (iv) **Infini-Attention** [Munkhdalai et al., 2024], which augments attention with a bounded recurrent memory state. All baselines use the same base model and are evaluated under matched KV cache budgets.

Evaluation. We evaluate PARADE under two complementary settings: (1) Perplexity on sequences ranging from 2K to 128K tokens, evaluated on 100 held-out test samples of length 128K from PG19, ArXiv, and The Stack (disjoint from the training data), measuring the model's ability to utilize extended context; and (2) LongBench [Bai et al., 2024], a multi-task benchmark evaluating long-context understanding across question answering, summarization, and code reasoning tasks.

Method	NarrQA	Qasper	MFQA	HQA	2Wiki	MuSiQue	GovRep	LCC	Repo	Avg
Qwen-2.5-0.5B										
Full Attention	14.60	21.37	36.11	23.39	22.14	10.42	18.57	42.71	45.67	26.11
StreamingLLM†	13.18	15.28	30.87	15.55	21.27	6.25	15.54	42.18	45.23	22.82
H_2O†	13.53	15.47	29.71	15.82	21.06	6.72	15.76	42.12	45.38	22.84
Infini-Attn	13.98	16.92	**31.85**	17.20	**21.29**	7.03	**16.42**	42.02	45.97	23.63
PARADE-0.5B	**14.35**	**24.18**	31.17	**22.75**	20.64	**10.07**	16.19	**44.12**	**48.47**	**25.77**
LLaMA-2-7B										
Full Attention‡	17.70	16.94	34.20	29.76	30.28	9.65	16.13	54.34	50.06	28.78
StreamingLLM†	17.56	16.80	**34.82**	28.47	30.59	9.60	16.27	53.08	49.77	28.55
Infini-Attn	17.89	17.03	28.81	29.58	**30.62**	9.79	15.83	54.67	51.11	28.37
PARADE-7B	**18.72**	**19.14**	27.90	**31.79**	30.46	**12.74**	**16.66**	**58.62**	**54.20**	**30.03**

Table 1: LongBench results (higher is better). Results are reported for PARADE-0.5B (Qwen-2.5-0.5B base) and PARADE-7B (LLaMA-2-7B base). †denotes training-free methods. ‡On LLaMA-2-7B, the Full Attention baseline truncates inputs to the last 4K tokens to stay within the native window. H_2O is omitted on the 7B setting because it cannot extrapolate beyond the training window.

5 Results

5.1 Perplexity on Long Sequences

We evaluate perplexity on sequences ranging from 2K to 128K tokens. Figure 3 presents results for both PARADE-0.5B and PARADE-7B, offering complementary perspectives. The long-context base model (Qwen-2.5-0.5B) allows us to examine how closely PARADE approximates Full Attention, while the short-context base model (LLaMA-2-7B) evaluates whether PARADE can effectively extrapolate beyond the native context window.

On PARADE-0.5B, built upon Qwen-2.5-0.5B with a long context window, Full Attention serves as an upper bound across all sequence lengths (Figure 3a). At short to moderate lengths ($\leq$16K), all methods perform similarly, indicating that PARADE preserves short-context capability. As the context length increases, the methods begin to diverge. StreamingLLM and H_2O gradually degrade due to information eviction, while Infini-Attention suffers from information compression, leading to higher perplexity at long lengths. In contrast, PARADE remains closely aligned with Full Attention across the entire range, staying within a small margin even at 128K. This suggests that PARADE effectively preserves long-range information under constrained memory.

On PARADE-7B, based on LLaMA-2-7B with a native context window of 4K, the evaluation probes extrapolation beyond the training regime (Figure 3b). In this setting, Full Attention becomes unstable beyond 4K due to RoPE extrapolation, and H_2O exhibits similar behavior, with perplexity rapidly increasing once the context exceeds the training window. In contrast, StreamingLLM, Infini-Attention, and PARADE avoid this issue by restricting attention within the native window. Among these methods, PARADE consistently achieves the lowest perplexity beyond 8K, with the performance gap widening as context length increases. Unlike prior approaches whose gains largely saturate at longer contexts, PARADE continues to benefit from additional context, demonstrating substantially more effective length generalization.

5.2 Performance on LongBench

Table 1 summarizes LongBench results for both base models. On PARADE-0.5B, Full Attention achieves the highest overall score (26.11), serving as an upper bound. Among efficient methods, PARADE attains the best average performance (25.77), substantially outperforming StreamingLLM (22.82), H_2O (22.84), and Infini-Attention (23.63), and closely approaching Full Attention. The gains are particularly evident on tasks that require aggregating long-range information, such as NarrativeQA, Qasper, and MuSiQue, where relevant evidence often lies outside the attention window. PARADE also shows consistent improvements on reasoning-intensive tasks like HotpotQA and GovReport. In contrast, on tasks dominated by local context (e.g., LCC and RepoBench-P), the differences across methods are smaller, as most relevant information resides within the attention

window. We observe relatively weaker performance on MFQA, which we hypothesize is due to a distribution mismatch between the task domain and the pretraining corpus.

On PARADE-7B, the evaluation reflects a different regime. Due to the 4K native context window of LLaMA-2-7B, the Full Attention baseline is restricted to the most recent 4K tokens, limiting access to earlier context. In this setting, PARADE consistently outperforms both Full Attention and other streaming baselines. The gains are particularly pronounced on document-level tasks such as NarrativeQA, HotpotQA, MuSiQue, GovReport, and RepoBench, where relevant information is distributed across long contexts and often lies far beyond the attention window. These results indicate that PARADE effectively recovers and utilizes information that would otherwise be inaccessible under truncated attention.

The advantage of PARADE becomes more evident when useful context extends beyond the attention window. On the long-context model, PARADE approaches the Full Attention upper bound while maintaining constant memory. On the short-context model, it substantially surpasses the truncated Full Attention baseline by leveraging additional context beyond the native training window. This consistent behavior suggests that parametric memory enables more effective utilization of long-range information under fixed memory constraints.

Additional ablations and efficiency analyses are provided in Appendix A.4 and Appendix A.6.

6 Related Work

Efficient Long-Context Transformers. A central challenge in long-context Transformers is avoiding the quadratic cost of full attention. Sliding-window methods [Beltagy et al., 2020, Child et al., 2019] restrict attention to local neighborhoods, while StreamingLLM [Xiao et al., 2023] combines sliding attention with attention sinks for streaming inference. KV cache compression methods such as H_2O [Zhang et al., 2023] and Scissorhands [Liu et al., 2023] retain only salient activations, and Infini-attention [Munkhdalai et al., 2024] augments attention with bounded recurrent memory. More recently, recurrent architectures and state-space models such as RetNet [Sun et al., 2023], Mamba [Gu and Dao, 2023], and test-time training approaches such as TTT [Sun et al., 2024] replace explicit token-level memory with recurrent state updates. Existing efficient long-context methods either discard past information (e.g., sliding-window attention and KV eviction), compress context into a single recurrent state (e.g., Infini-attention, Mamba, RetNet, and TTT), or replace the Transformer architecture altogether (e.g., state-space models). In contrast, PARADE is a plug-in framework for pretrained Transformers that represents context as a bounded set of query-conditioned weight-space updates. This preserves the model's original computation while extending its effective memory through the same mechanism used to encode parametric knowledge.

Fast Weights and Weight-Space Memory. Fast weight models [Schmidhuber, 1992] store temporary associations through weight updates accumulated into a memory matrix. Subsequent work showed that linear attention can be interpreted as a form of fast weight programming [Schlag et al., 2021], establishing a close connection between attention and associative memory. More recent analyses further characterize Transformers as performing implicit gradient-based or weight-space memory updates during in-context learning [Dherin et al., 2025, Goldwaser et al., 2025]. Our work builds on this perspective by showing that contextual influence in Transformers can be expressed as updates to MLP weights, motivating the use of persistent parametric memory in PARADE.

Dynamic Parameter Generation. Dynamic parameter generation methods produce input-conditioned weight updates for a target model. Recent approaches such as Doc-to-LoRA [Charakorn et al., 2026] and SHINE [Liu et al., 2026] map an entire context into LoRA adapters in a single forward pass, enabling query answering without revisiting the original context. Doc-to-LoRA and SHINE compile an entire context into a single static LoRA adapter through a one-shot hypernetwork pass, applying the same modification uniformly across all queries. These approaches also rely on curated supervision and distillation pipelines for training. In contrast, PARADE incrementally accumulates context over arbitrarily long streams and constructs query-specific weight updates through sparse routing. It is trained end-to-end using standard next-token prediction on raw long-form text, without requiring task-specific supervision or data engineering.

7 Conclusion

We empirically verify that contextual influence in Transformers can be represented as weight updates within a low-dimensional subspace, which suggests that long-range context does not require explicit token-level storage. Based on this insight, we propose PARADE, which represents context as bounded parametric memory and composes it through input-dependent weight modifications. This enables constant-memory streaming while preserving long-context capability. PARADE matches full attention in long-context settings and outperforms prior efficient methods under the same memory constraints. It also supports extrapolation beyond the training context length. These results indicate that representing context in weight space is a scalable approach for long-context modeling.

8 Limitations

While PARADE demonstrates strong performance and favorable scaling behavior, several limitations remain. First, PARADE requires training additional modules, unlike training-free methods (such as StreamingLLM and H_2O). We mitigate this by keeping the base model fully frozen and introducing only a small set of plug-in parameters, which preserves the pretrained model's behavior and avoids any modification to its existing weights. Second, our experiments are limited to models up to 7B parameters. While the consistent behavior across two architecturally distinct base models (Qwen-2.5-0.5B and LLaMA-2-7B) and context lengths spanning 2K to 128K strongly suggests that the proposed mechanism is not tied to a particular scale, validating these benefits in the 70B+ regime is an important direction for future work. Finally, on the long-context model (PARADE-0.5B), there remains a small gap to full attention on LongBench (25.77 vs. 26.11), indicating that some information loss persists under bounded memory. We note that this gap is achieved while using approximately 30× less KV memory at 128K and matches or surpasses all efficient baselines tested.

Acknowledgment

This work is supported by the Collaborative Innovation Center of Novel Software Technology and Industrialization, the High Performance Computing Platform of Nanjing University of Aeronautics and Astronautics, the Short-term Visiting Program of Nanjing University of Aeronautics and Astronautics for Ph.D. Students Abroad (No. 250609DF16) and Postgraduate Research & Practice Innovation Program of Jiangsu Province (26CXJH1037). T. Chen is partially supported by overseas grants from the State Key Laboratory of Novel Software Technology, Nanjing University (KFKT2025A05, KFKT2026A04).

References

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation. *arXiv preprint arXiv:2309.14402*, 2023.

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35:38546–38556, 2022.

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 (volume 1: Long papers)*, pages 3119–3137, 2024.

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

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-lora: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026.

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

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, 2018.

Benoit Dherin, Michael Munn, Hanna Mazzawi, Michael Wunder, and Javier Gonzalvo. Learning without training: The implicit dynamics of in-context learning. *arXiv preprint arXiv:2507.16003*, 2025.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pages 30–45, 2022.

Adrian Goldwaser, Michael Munn, Javier Gonzalvo, and Benoit Dherin. Equivalence of context and parameter updates in modern transformer blocks. *arXiv preprint arXiv:2511.17864*, 2025.

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

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, et al. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533*, 2022.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. Shine: A scalable in-context hypernetwork for mapping context to lora in a single pass. *arXiv preprint arXiv:2602.06358*, 2026.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 101:15, 2024.

Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

Qwen, An Yang, Baosong Yang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

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

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, 2021.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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.

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.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Xiangyu Zhang, Yu Zhou, Guang Yang, Harald C Gall, and Taolue Chen. Anchor attention, small cache: Code generation with large language models. *IEEE Transactions on Software Engineering*, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

A Implementation Details

A.1 Training Objective

We train the memory mechanism end-to-end using next-token prediction, augmented with two auxiliary losses.

Language Modeling Loss. The primary loss is the standard cross-entropy over predicted tokens. To encourage the model to leverage memory effectively, we additionally upweight positions where memory is actively used:

$$\mathcal{L}_{\mathrm{LM}} = \mathrm{CE}_{\mathrm{all}} + \beta \cdot \mathrm{CE}_{\mathrm{memory}}, \tag{1}$$

where $\mathrm{CE}_{\mathrm{all}}$ is the mean cross-entropy over all positions, and $\mathrm{CE}_{\mathrm{memory}}$ is the mean cross-entropy restricted to positions where at least one layer's memory gate is active.

Retention Loss. As the memory is incrementally updated, the write gate may overwrite slots that still carry useful information from earlier chunks. To discourage this, we introduce a parameter-free retention loss that verifies whether past context remains retrievable from the current memory.

At each step, the compressed representation $Z^{(t)} \in \mathbb{R}^{r \times d}$ is detached and stored as an *anchor*, serving as a snapshot of the information encoded at that point. At a later step, we uniformly sample K historical anchors and attempt to retrieve each from the current memory. Concretely, we compute the cosine similarity between the anchor and every memory slot, apply softmax to obtain a mixing weight over slots, and take the weighted combination as the retrieved vector. If a slot whose direction closely matches the anchor still exists, the weight concentrates on it and the retrieval faithfully recovers the anchor. Conversely, if the gate has since overwritten that slot, no slot aligns well, the weight disperses, and the retrieval degrades. Formally:

$$\hat{Z}_k = \mathrm{softmax}\Big(\tilde{Z}_k \tilde{M}^\top\Big)\,\tilde{M}, \qquad \mathcal{L}_{\mathrm{retain}} = \frac{1}{K}\sum_{k=1}^{K}\Big\|\hat{Z}_k - \tilde{Z}_k\Big\|_2^2, \tag{2}$$

where $\tilde{Z}_k$ is the ℓ_2-row-normalized anchor and $\tilde{M} \in \mathbb{R}^{R \times d}$ is the ℓ_2-row-normalized current memory.

Orthogonal Regularization. To maximize the representational capacity of the R slots, we encourage their key directions to be orthogonal:

$$\mathcal{L}_{\mathrm{orth}} = \|\hat{A}\hat{A}^\top - I_R\|_F^2, \quad \hat{A}_{i,:} = \frac{A_{i,:}}{\|A_{i,:}\|_2}, \tag{3}$$

where $A \in \mathbb{R}^{R \times d_{\mathrm{in}}}$ stacks the key vectors $\{a_i\}_{i=1}^{R}$ as rows and $\hat{A}$ is its row-normalized version. This prevents rank collapse, ensuring that each slot occupies a distinct direction in input space.

Total Loss. The complete training objective is:

$$\mathcal{L} = \mathcal{L}_{\mathrm{LM}} + \lambda_{\mathrm{retain}}\,\mathcal{L}_{\mathrm{retain}} + \lambda_{\mathrm{orth}}\,\mathcal{L}_{\mathrm{orth}}. \tag{4}$$

A.2 Streaming KV Cache

PARADE processes long documents in a streaming manner using an incremental KV cache. At each step, only the newly arriving chunk of tokens is passed through the base model, while previously computed KV states are retained in the cache to provide context. This avoids recomputing the full window at every step and substantially reduces computational cost.

When the cache reaches its capacity (the window size N), the oldest chunk of tokens (excluding the initial sink positions described below) is evicted in chronological order, and their hidden representations are compressed into parametric memory as described in Section 3.2.

Following StreamingLLM [Xiao et al., 2023], we permanently retain the KV states of the first n_{sink} tokens (4 tokens in our setting) in the cache. This design is motivated by the empirical observation that many attention heads allocate a disproportionately large fraction of their attention mass to the initial tokens, regardless of their semantic content. Evicting these tokens destabilizes the attention distribution and causes a sharp increase in perplexity. By preserving them as *attention sinks*, the sliding window maintains a stable attention pattern throughout streaming.

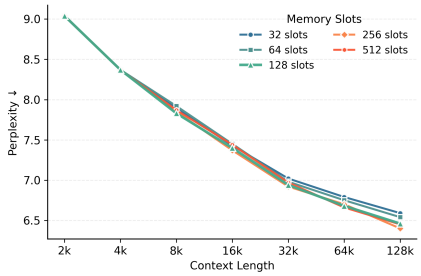


Figure 4: Perplexity vs. context length under different numbers of memory slots R on PARADE-0.5B.

Variant	2k	4k	8k	32k	128k
Full PARADE	**9.04**	**8.37**	7.83	**6.94**	**6.46**
w/o gating	9.04	8.37	7.91	7.02	6.57
w/o sparse write	9.04	8.37	**7.80**	6.97	6.57
w/o routing	9.04	8.37	7.85	7.08	6.60
w/o retention loss	9.04	8.37	7.89	7.12	6.56

Table 2: Component ablation on PARADE-0.5B. Each row removes one component from the full system. Perplexity (lower is better) is reported across context lengths.

A.3 Cross-Attention Initialization

The cross-attention modules in PARADE adopt the Perceiver architecture (see Section 3.2). We initialize these modules from the self-attention weights of a middle layer of the frozen base model (e.g., layer 12 of Qwen2.5-0.5B, which has 24 layers in total). Specifically, the query and output projection weights are directly copied, and the corresponding LayerNorm parameters are also transferred.

This warm-start strategy endows the hypernetwork with meaningful feature extraction capabilities from the outset, rather than requiring it to learn cross-attention from scratch. Since the hypernetwork operates on the base model's hidden representations, initializing it with the base model's own attention weights ensures that the queries and keys are already aligned with the underlying representation space, leading to more stable and efficient training.

A.4 Ablation Study

We conduct ablation experiments on PARADE-0.5B to isolate the contribution of each design choice.

Effect of Memory Capacity. Figure 4 shows perplexity across context lengths for different numbers of memory slots R. At short contexts ($\leq$8K), all configurations perform similarly, as most relevant information lies within the attention window. As context length increases, larger R consistently improves performance, indicating that additional slots help preserve long-range information. The improvement is most significant when increasing R from 32 to 128 (e.g., 6.59 $\rightarrow$ 6.46 at 128K), while further scaling to 256 or 512 yields only marginal gains. This suggests diminishing returns beyond a moderate memory size. These results align with our analysis in Section 2: the effective contextual dependency is constrained by attention sparsity rather than sequence length, so a relatively small number of slots is sufficient to capture most useful long-range information.

Component Ablation. Table 2 isolates the contribution of each component. While all variants perform similarly at short contexts, differences become more pronounced as the context grows, where long-range memory is critical. Removing *routing* leads to the largest degradation at long contexts (6.60 vs. 6.46 at 128K), highlighting the importance of input-dependent memory selection. Removing *gating* (6.57 at 128K) and *retention loss* (6.56) results in comparable degradation, indicating that both selective updates and temporal credit assignment are necessary for effective long-term retention. Removing *sparse write* yields a similar drop at long contexts (6.57), suggesting that preserving untouched memory slots is important for maintaining previously stored information. Interestingly, it slightly improves performance at shorter lengths (7.80 vs. 7.83 at 8K), indicating that unrestricted updates may benefit local modeling but harm long-range retention. Overall, these results demonstrate that each component contributes to stable and effective long-context modeling, with their impact becoming increasingly significant as the sequence length grows.

Hyperparameter Sensitivity. We further investigate the sensitivity of PARADE to key hyperparameters, including the number of memory slots selected during writing and reading, the number of injection layers, and the chunk size. Table 3 reports perplexity across different context lengths.

Table 3: Hyperparameter sensitivity of PARADE-0.5B. Perplexity (lower is better) is reported at different context lengths. Default settings are highlighted in bold.

Hyperparameter	Setting	8K	32K	128K
Write top-k	4	7.91	6.97	6.52
	16 (Default)	**7.83**	**6.94**	**6.46**
	64	7.83	6.98	6.55
Routing top-k	4	7.94	7.04	6.57
	16 (Default)	**7.83**	**6.94**	**6.46**
	64	7.85	7.02	6.58
Injection layers	4	7.85	7.01	6.52
	8 (Default)	**7.83**	**6.94**	**6.46**
	24	7.80	6.89	6.42
Chunk size	256	7.91	7.27	6.69
	2048 (Default)	**7.83**	**6.94**	**6.46**
	8192	7.78	6.90	6.40

Table 4: Subspace intervention on LongBench with PARADE-0.5B. In-subspace retains the memory components within the contextual footprint subspace, while Out-subspace retains its orthogonal complement.

Setting	NQA	QSP	MFQA	HPQA	2Wiki	MQ	Gov	LCC	Repo	Avg.
Full PARADE	14.35	24.18	31.17	22.75	20.64	10.07	16.19	44.12	48.47	**25.77**
In-subspace (500D)	14.28	23.35	30.27	21.18	21.57	7.33	16.08	43.32	48.02	<u>25.04</u>
Out-subspace	13.28	17.60	30.35	18.87	19.92	8.65	15.60	43.78	47.74	23.98
No Memory	12.89	14.47	29.28	15.07	20.10	6.03	15.05	41.85	44.32	22.11

The results show that moderate sparsity in both memory writing and routing achieves a favorable balance between information retention and interference. Increasing the number of injection layers and the chunk size generally improves perplexity, but also introduces additional computational overhead. We therefore adopt the default configuration as a practical trade-off between modeling performance and efficiency.

A.5 The Low-Rank Structure of Parametric Memory

To examine whether PARADE exploits the low-dimensional contextual structure identified in Section 2, we conduct a subspace intervention on its learned memory. Recall that the memory-induced MLP activation is

$$\Delta h(x) = \Delta W(x)x = \sum_{i=1}^{R} \alpha_i(x)(a_i^\top x)b_i. \tag{5}$$

Since the output directions b_i lie in the same space as the contextual footprints δh, we estimate the footprint subspace for each evaluation sample following Section 2, using PARADE's eviction strategy. Let P_S denote the projector onto its top 500 principal directions. We intervene by replacing b_i with either

$$b_i^{\text{in}} = P_S b_i, \qquad b_i^{\text{out}} = (I - P_S)b_i, \tag{6}$$

while keeping all other components unchanged.

As shown in Table 4, the in-subspace intervention achieves an average score of 25.04, close to the full model (25.77). In contrast, retaining only the orthogonal components reduces performance to 23.98, despite preserving approximately 90% of the activation dimensions. These results suggest that PARADE's learned memory relies disproportionately on the low-dimensional contextual footprint subspace, strengthening the connection between the empirical observations in Section 2 and our parametric memory design.

A.6 Efficiency Analysis

We analyze the efficiency of PARADE-0.5B compared to baselines under the same streaming configuration.

	Throughput (tok/s)				KV Cache (MB)			
Method	4K	16K	64K	128K	4K	16K	64K	128K
Full Attention	173	175	63	34	48	192	768	1536
StreamingLLM	149	150	150	150	48	48	48	48
H_2O	149	154	155	155	48	48	48	48
Infini-Attn	152	131	131	132	48	48	48	48
PARADE	152	134	134	134	48	48	48	48

Table 5: Inference throughput (tokens/sec) and KV cache size (MB) of PARADE-0.5B on a single RTX 4090 GPU. Full attention degrades significantly at long contexts due to quadratic cost and linearly growing KV cache.

Memory Efficiency. As shown in Table 5, full attention requires 1536 MB of KV cache at 128K tokens, growing linearly with sequence length, while its throughput drops from 173 tok/s at 4K to 34 tok/s at 128K. PARADE bounds the KV cache to the sliding window size regardless of context length, maintaining near-constant throughput comparable to other streaming methods. Beyond the KV cache, PARADE introduces a fixed parametric memory of only 1.75 MB, a negligible overhead that does not grow with context length.

Throughput. All streaming methods maintain near-constant throughput by bounding the KV cache to the window size. PARADE performs comparably to Infini-Attn at long contexts, with both incurring modest overhead from their respective memory mechanisms compared to simpler streaming baselines.

NeurIPS Paper Checklist

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 describe the paper's main contributions: (1) the empirical characterization of low-rank contextual updates in weight space (Section 2), (2) the PARADE framework for bounded parametric memory (Section 3), and (3) experimental evaluation on long-context benchmarks and extrapolation settings (Sections 4 and 5).

   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: The paper includes a dedicated Limitations section (Section 8) discussing training overhead, evaluation scope, scalability to larger models, and the remaining performance gap to full attention.

   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: The paper does not present new formal theoretical results requiring proofs. Section 2 discusses prior equivalence results from existing work and provides empirical analyses motivating the proposed architecture.

   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 paper describes the model configurations, training setup, datasets, streaming configuration, memory architecture, evaluation benchmarks, and hyperparameters in Sections 3–5 and Appendix A, providing sufficient detail to reproduce the experiments.

   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: We provide the code for reproducing the experimental results in the supplementary material, together with implementation details

   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 paper reports model architectures, datasets, training configurations, hyperparameters, evaluation protocols, and baseline settings in Sections 3–5 and Appendix A.

   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: Due to the computational cost of long-context training and evaluation, we do not report results across multiple random seeds.

   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 paper describes the model sizes, streaming configurations, and hardware used for efficiency evaluation (Appendix A.6).

   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 authors have read and will adhere 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: [N/A]

Justification: This work focuses on efficient long-context modeling and does not introduce application-specific societal impact beyond those generally associated with large language models.

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 involve high-risk applications, human subjects, or deployment-specific safety interventions.

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 describes all datasets used for training and evaluation, including PG19, ArXiv, The Stack, and LongBench, together with corresponding citations.

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: All datasets used are publicly available research datasets with existing documentation and citations provided in the paper.

    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 work does not involve human subjects or human participant data collection.

    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 work does not involve 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 authors used large language models solely for language editing and writing assistance.

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.