Beyond Static Sinks: Reinforcement-Learned KV Retention for Streaming Language Models

Adam Kuca **Peizhou Huang** **Chaoyu Zhang** **Xiangzhong Ye**
Department of Computer Science and Engineering, University of Michigan
{adamkuca, peizhouh, chaoyz, yxiangz}@umich.edu

Abstract

Efficient memory management is critical for deploying Large Language Models (LLMs) in streaming applications. Existing methods like StreamingLLM rely on static heuristics (e.g., retaining initial tokens and recent windows) to limit Key-Value (KV) cache growth, often discarding semantically important information in the middle context. In this work, we propose a learned KV retention framework that dynamically ranks and selects tokens based on their utility for future predictions. We introduce a lightweight policy network trained via a three-stage pipeline: Oracle data collection, Behavioral Cloning (BC), and Direct Preference Optimization (DPO). Experiments on the PG-19 dataset demonstrate that our method outperforms static baselines in extreme compression regimes (cache size ≤ 128), achieving a perplexity improvement of 1.37 points while incurring negligible latency overhead ($< 0.1\%$). Visual analysis confirms that our policy successfully learns to preserve critical semantic anchors beyond the local window.

1 Introduction

Large Language Models (LLMs) deployed in real-time or streaming settings face a growing bottleneck in how they manage memory during long-context inference. With each new token generated, the model must cache all past Key–Value (KV) pairs for attention, leading to rapidly increasing memory use and latency. Sliding-window attention can limit the cache size, but this often leads to instability and sharp increases in perplexity when early tokens are dropped.

Recently, Xiao et al. [2024] introduced **StreamingLLM**, which provides a method for stable long-context inference. This approach retains a limited set of initial attention sink tokens in memory, in addition to a rolling window of recent tokens. This mechanism stabilizes the attention distribution and enables the model to process sequences containing millions of tokens while maintaining perplexity levels comparable to the full-context baseline.

However, StreamingLLM still relies on a fixed rule for which tokens to keep: the first few tokens (sinks) are always preserved, while all others are evicted in a FIFO manner. This static approach ignores the fact that not all tokens contribute equally to future predictions. Inspired by Lin et al. [2025], which introduced REFRAG to dynamically expand retrieved contexts using a lightweight reinforcement learning (RL) policy, this project explores whether a similar idea can be applied to the KV cache itself. Specifically, we learn a small RL policy that decides, at each step, which KV entries to keep, adjusting memory allocation dynamically instead of relying on fixed heuristics.

To address these challenges, we focus on **long-context language modeling** to improve memory efficiency without sacrificing stability. We compare our proposed **RL-Cache**—a lightweight policy that dynamically retains KV entries based on learned utility—against **Static Window** and **StreamingLLM** baselines. Trained to minimize a combined objective of prediction loss and memory usage, we hypothesize that this adaptive mechanism can match the stability of static heuristics while significantly reducing memory footprint.

2 Related Work

This project touches on three main areas of LLM research: **efficient attention**, **long-context modeling**, and **adaptive memory control**.

Earlier work on efficient attention, such as Longformer [Beltagy et al., 2020] and BigBird [Zaheer et al., 2020], introduced fixed sparse patterns to reduce quadratic complexity. Clustering-based approaches like Reformer [Kitaev et al., 2020] and Routing Transformer [Roy* et al., 2020] grouped tokens by similarity using LSH or k-means routing to make attention more selective. And Sinkhorn Transformers [Tay et al., 2020] extended this idea by learning sparse permutations through differentiable sorting. All these methods target attention computation cost rather than runtime caching.

More recent approaches, such as Performer [Choromanski et al., 2020] and Hyena [Poli et al., 2023], approximate or replace attention entirely using kernels or convolutions, achieving sub-quadratic complexity. However, they require retraining and cannot be applied to existing pretrained LLMs. In contrast, **StreamingLLM** [Xiao et al., 2024] targets only cache management during decoding, keeping the LLM architecture intact. It achieves a stable perplexity by always keeping a few "sink" tokens that act as global anchors for attention.

On the input side, compression-based methods such as CEPE [Yen et al., 2024] and LongLLMLingua [Jiang et al., 2024] reduce context length before feeding it into the model. REFRAG [Lin et al., 2025] goes even further by learning a reinforcement policy to selectively expand only the most relevant chunks during generation, achieving large gains (up to 30×) in latency and throughput.

The proposed RL-Cache extends this idea to the decoding stage: instead of learning which inputs to expand, it learns which past states to retain.

3 Methodology

We propose a learned KV-cache compression framework that dynamically selects tokens to retain in the model's memory. Unlike heuristic-based methods (e.g., StreamingLLM), our approach trains a lightweight policy network to rank tokens based on their utility for future predictions. Our method is designed for efficiency, so that any decision overhead is negligible compared to the decoding cost.

3.1 Hybrid KV-Cache Architecture

To maintain decoding stability while enabling high compression ratios, we employ a hybrid cache management strategy. At each time step t, the KV-cache is partitioned into three functional segments:

1. **Sink ($\mathcal{S}$)**: The first N_{sink} tokens (typically 16–32 in our experiments) are always retained. They serve as crucial attention anchors to prevent the "attention sink" phenomenon, ensuring that the self-attention mechanism remains stable even when recent history is discarded.
2. **Window ($\mathcal{W}$)**: The most recent N_{window} tokens (e.g., 96) are strictly preserved to capture immediate local dependencies and syntactic continuity.
3. **Remote ($\mathcal{R}$)**: The intermediate history between the Sink and the Window. **This region represents the long-term context and is the sole target for our compression policy.**

The compression policy is triggered whenever the Remote region exceeds a pre-defined capacity. Given a strict token budget B_{remote}, the policy network evaluates all tokens currently in $\mathcal{R}$. We retain the top-B_{remote} tokens and permanently evict the rest from memory. This simplifies the problem to a simpler ranking task within the remote buffer.

3.2 Lightweight Policy Network

We prioritize inference latency by adopting a minimalist feature set and architecture.

State Representation. For a token i at step t, the input state vector x_i is constructed by concatenating semantic and temporal information:

$$x_i = [\mathbf{h}_i; \text{pos_norm}_i; \text{rel_age}_i] \tag{1}$$

where $\mathbf{h}_i \in \mathbb{R}^{d_{model}}$ is the final-layer hidden state of the student model at token i. $\text{pos_norm}_i = i/(T-1)$ is the normalized absolute position at step t (with T tokens seen so far). Finally, $\text{rel_age}_i = (t-i)/L_{\max}$ is the normalized relative age (distance in tokens to the current step).

Architecture. The policy π_θ is a specialized MLP with Gated Residual connections. It projects the input to a hidden dimension ($d = 256$) and processes it through a stack of Gated Residual Blocks using a Gated Linear Unit (GLU) variant:

$$\mathbf{z} = \text{LayerNorm}(\mathbf{x}) \to \mathbf{v}, \mathbf{g} = \text{Linear}(\mathbf{z}) \to \mathbf{y} = \text{GELU}(\mathbf{v}) \odot \sigma(\mathbf{g}) \to \mathbf{x} + \text{Dropout}(\text{Linear}(\mathbf{y})) \tag{2}$$

This gating mechanism allows the network to selectively filter semantic features relevant for retention before outputting a scalar retention score s_i.

3.3 Training Pipeline

3.3.1 Stage 1: Oracle Data Collection

We employ a "Teacher–Student" framework to construct high-quality training data without relying on handcrafted heuristics. The student model is **Qwen2.5-0.5B**. Oracle cache decisions are generated by a teacher model using the search procedure described above. In our experiments, we consider teachers of different model scales (e.g., **Qwen2.5-0.5B** and **Qwen2.5-7B**) to study the effect of teacher capacity.

The Teacher evaluates a candidate mask m for the Remote region via a *Horizon Look-ahead*. We construct a temporary cache with the masked history and compute the cumulative Negative Log-Likelihood (NLL) of the ground-truth tokens in the immediate future range $[t+1, t+64]$:

$$\mathcal{L}(m) = -\sum_{k=1}^{H} \log P_{\text{Teacher}}(w_{t+k} \mid \mathcal{S}, \mathcal{W}, \text{Mask}(\mathcal{R}, m), w_{t+1:t+k-1}) \tag{3}$$

Since the search space is combinatorial, we approximate the global optimum by evaluating diverse candidate masks (e.g., based on hidden-state norms and random sampling). The mask m^* yielding the lowest cumulative NLL is identified as the *Best Remote Set*. Tokens included in m^* are labeled as positive ($y_i = 1$), and others as negative ($y_i = 0$).

3.3.2 Stage 2: BC Training

We initialize the policy π_θ by cloning the Oracle's behavior. We formulate retention as a binary classification problem. To mitigate the extreme class imbalance (retention budget $\ll$ remote size), we employ a Weighted Binary Cross-Entropy (BCE) loss:

$$\mathcal{L}_{BC} = -\frac{1}{N}\sum_{i=1}^{N} [w_{pos} \cdot y_i \log \sigma(\pi_\theta(x_i)) + (1-y_i)\log(1-\sigma(\pi_\theta(x_i)))] \tag{4}$$

3.3.3 Stage 3: DPO Fine-tuning

Because cache compression is fundamentally a *ranking* problem and since BC treats decisions independently, we further refine the policy using DPO [Rafailov et al., 2023].

We construct a preference dataset $\mathcal{D}_{pref}$ by sampling pairs (x_{win}, x_{lose}) where x_{win} is drawn from the Oracle's Best Remote Set and x_{lose} is a rejected token. Using the BC-trained policy as a frozen reference π_{ref}, we optimize:

$$\mathcal{L}_{DPO} = -\mathbb{E}_{(x_w, x_l)\sim\mathcal{D}_{pref}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(x_w)}{\pi_{ref}(x_w)} - \beta \log \frac{\pi_\theta(x_l)}{\pi_{ref}(x_l)}\right)\right] \tag{5}$$

Here, $\beta = 0.3$ controls the KL-divergence constraint. This objective encourages the policy to assign significantly higher scores to important tokens relative to unimportant ones, creating a larger decision margin. This is crucial for stability, as it aligns the training objective directly with the top-k selection mechanism used during inference.

4 Evaluation

4.1 Experimental settings

All experiments are run in a streaming-style setting: the model reads tokens left-to-right and can only attend to a bounded KV cache. For the 0.5B student we use $C \in \{32, 64, 128, 256, 384, 512\}$. When using **Qwen2.5-7B** as the oracle teacher, we additionally report results for $C \in \{64, 96, 128, 256\}$. For a given budget C and sink size S, the cache is partitioned into a *sink* region that always keeps the first S tokens, a *window* region that keeps the most recent tokens, and a small *remote* region of size B_{remote}. We refer to $C \leq 128$ as the *extreme compression* regime and report perplexity under these compressed-attention settings unless otherwise stated.

Unless otherwise specified, the student model is **Qwen2.5-0.5B**. We evaluate behavior cloning (BC) policies trained with teachers of different capacities, including **Qwen2.5-0.5B** and **Qwen2.5-7B**, to analyze the impact of teacher strength.

The cache policy operates at the token level. For each candidate remote token at step t, we concatenate its last-layer hidden state from the student model with two scalar features: a normalized absolute position and a relative "age" (distance in tokens from the current step). This feature vector is fed into a lightweight 3-layer MLP with a 256-dimensional hidden layer that outputs a scalar retention score. We first train this network with a sigmoid + BCE loss to match 0/1 keep labels derived from teacher-optimal remote sets (behavior cloning), and then optionally fine-tune it with DPO on token-level preference pairs (preferred vs. rejected remote tokens). At inference time, we use the same $(S, B_{\text{remote}}, \text{window})$ settings as in our evaluation protocol for each cache budget C, and the policy selects the top B_{remote} scored tokens to occupy the remote region at each step.

4.2 Datasets and metrics

We conduct all experiments on the **PG-19** dataset [Rae et al., 2019], a standard benchmark for long-form language modeling. PG-19 consists of full-length books (often up to $\sim$100k tokens) and is widely used to probe long-context behavior. For evaluation, we extract contiguous snippets from the validation split and feed them to the model in a streaming fashion, processing tokens left-to-right under a bounded KV cache.

Our comparisons are organized around three attention regimes that form a simple hierarchy. At the top, *Full Attention* lets the model attend to all previous tokens without any KV cache restriction; this offline setting is not streaming-friendly but provides an oracle upper bound on perplexity for a given model size. Under a fixed cache budget C, we then consider a constrained *Sliding Window* (SW) baseline: a small prefix of "sink" tokens is never evicted, and the remaining $C - S$ slots are allocated purely by recency, forming a sink+window pattern commonly used in streaming inference. This baseline isolates the degradation caused purely by limited KV capacity under a hand-crafted, recency-only policy. Finally, **RL-Cache** uses exactly the same budget C and the same sink size S, but splits the non-sink part into a local window and a learnable remote pool of size B_{remote}, with window $+ B_{\text{remote}} = C - S$. Any improvement over SW therefore comes entirely from a better allocation of the fixed cache slots, rather than from using more memory, while the remaining gap to Full Attention reflects the irreducible loss due to compression.

Our main quantitative metric is token-level perplexity (PPL) on held-out PG-19 validation snippets. For each cache budget C and model scale, we evaluate on a *fixed set* of 100 validation passages from PG-19 and report the mean token-level perplexity. Because the same evaluation set is used across all settings, the Full Attention perplexity for the 0.5B model remains constant at **19.25**.

In addition to these scalar metrics, we also visualize retention heatmaps (token-by-time keep patterns) to qualitatively compare RL-Cache against SW and to inspect which tokens are preferentially preserved in the remote region.

4.3 Quantitative results

We evaluate the effectiveness of our learned policy on PG-19, focusing on *extreme compression regimes* with cache size $C \leq 128$ tokens. For each cache budget C and model scale, we evaluate on a *fixed set* of 100 validation passages from PG-19 and report the mean token-level perplexity under the corresponding KV constraint.

Oracle Teacher Model	Cache C	SW PPL	BC PPL	DPO PPL
Qwen2.5-0.5B	32	44.39	43.59	**43.02**
Qwen2.5-0.5B	64	**29.62**	29.98	29.63
Qwen2.5-0.5B	128	24.66	**24.57**	24.76
Qwen2.5-0.5B	256	21.58	**21.55**	21.63
Qwen2.5-0.5B	384	**20.53**	**20.53**	20.55
Qwen2.5-0.5B	512	**20.08**	20.10	20.11
Qwen2.5-7B	64	29.62	29.37	**29.18**
Qwen2.5-7B	96	26.28	26.29	**26.09**
Qwen2.5-7B	128	**24.66**	24.83	**24.68**
Qwen2.5-7B	256	**21.58**	21.60	**21.58**

Table 1: **Quantitative results on PG-19.** We report perplexity (PPL; lower is better) under the same cache budget for the Sliding Window (SW) baseline and the learned policies trained via behavior cloning (BC) or DPO. The oracle teacher model indicates which model is used to generate the oracle cache decisions for training.

Table 1 summarizes the results across cache sizes and model scales. The smallest budget ($C = 32$) corresponds to a highly compressed regime in which only a tiny fraction of the prefix can be stored, making the choice of remote tokens particularly critical.

Overall, RL-Cache consistently matches or improves upon the sliding-window (SW) baseline under the same cache budget (Table 1). In the most compressed setting (e.g., $C = 32$ on the 0.5B model), the DPO-trained policy achieves approximately one point lower perplexity than SW. For other small budgets ($C \leq 128$), both BC and DPO yield smaller but stable improvements over the baseline. In terms of relative performance, BC and DPO are generally close: DPO tends to be more effective in the most extreme compression regimes, while BC can be competitive or slightly better as more cache capacity becomes available.

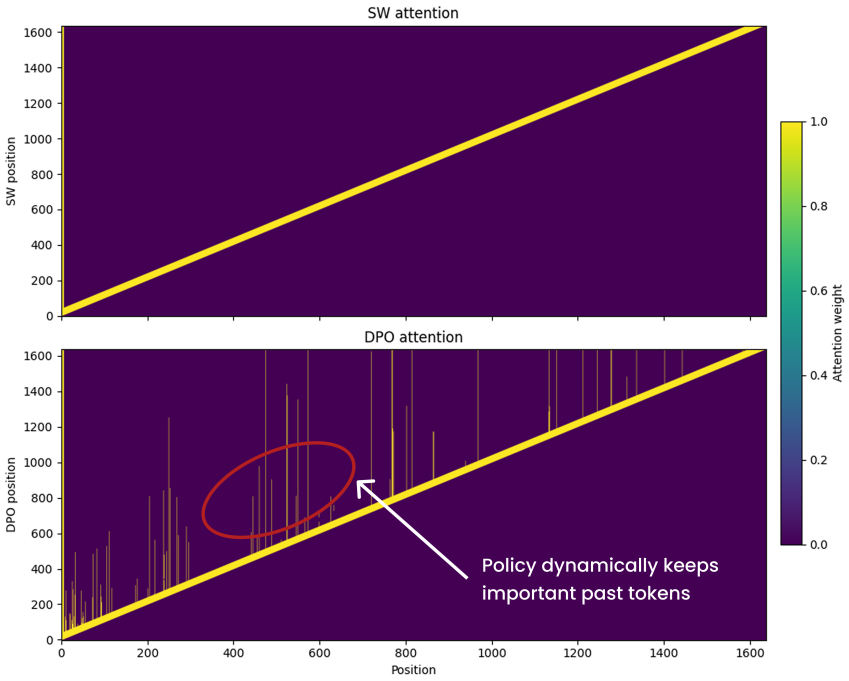


Figure 1: **Visualization of retention patterns. Top:** The static SW baseline exhibits a fixed diagonal pattern, keeping only the most recent tokens. **Bottom:** Our DPO policy (**RL-Cache**) learns a dynamic retention strategy. The vertical streaks (circled) indicate specific tokens in the remote history that are preserved over long periods, corresponding to critical semantic anchors.

4.4 Qualitative comparison

Comparing matched cache budgets (e.g., $C = 64, 128, 256$), using a stronger oracle teacher (7B) generally yields lower PPL for BC/DPO than using a 0.5B teacher. We visualize typical retention patterns in Figure 1. The sliding-window baseline (top) produces an almost perfect diagonal band, keeping only the most recent tokens and discarding anything that falls outside the local window. In contrast, our DPO-trained policy (bottom) shows clear vertical streaks in the remote region: a small set of tokens is preserved over long horizons and reused as *semantic anchors* for later predictions.

Although the retention behavior is more structured, RL-Cache adds only a negligible amount of computation. The policy network contributes less than 0.1% of the total inference FLOPs at $T = 2048$, and we do not observe any measurable drop in throughput or time-to-first-token. In practice, the method therefore behaves like a "smarter" cache with essentially the same runtime cost as the sliding-window baseline.

5 Conclusion

We have presented a simple learned framework for KV-cache compression in streaming language models. Instead of relying on a fixed sliding window, the proposed policy scores candidate tokens in a remote region and selects a small subset to keep, effectively turning cache eviction into a token-level ranking problem.

On PG-19, this learned policy consistently matches or improves upon a sliding-window baseline under the same cache budget, with the clearest gains appearing in the most compressed regimes. In the smallest-cache setting, the DPO-trained variant recovers roughly one point of perplexity compared to a purely recency-based strategy, while in larger-cache settings both behavior cloning and DPO provide smaller but stable improvements. At the same time, the policy network adds negligible computational overhead, so the method behaves effectively like a "smarter" cache with almost no runtime penalty.

This study is exploratory in scope: experiments are limited to a single dataset and relatively small model scales, and the teacher supervision is still defined in terms of next-token likelihood. Future work could extend this analysis to long-range reasoning benchmarks, incorporate more diverse baselines, and investigate end-to-end training schemes that couple cache management more tightly with the underlying language modeling objective.

6 Contribution by Group Members

Adam Kuca developed the theoretical foundation of the project, authored the project proposal, proposed the high-level method, and prepared and presented the final research poster.

Peizhou Huang was responsible for the improvement and parameter tuning of BC and DPO, as well as most of the experimental design, training, and plotting.

Chaoyu Zhang drafted and implemented the core code framework, including training pipeline scripts for oracle data collection, behavioral cloning, and DPO, and also ran a subset of the experiments.

Xiangzhong Ye contributed to the code debugging process, authored the Abstract and Method sections, and was responsible for compiling the first draft of the report.

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *ArXiv*, abs/2004.05150, 2020. URL https://api.semanticscholar.org/CorpusID:215737171.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.91. URL https://aclanthology.org/2024.acl-long.91/.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Xiaoqiang Lin, Aritra Ghosh, Bryan Kian Hsiang Low, Anshumali Shrivastava, and Vijai Mohan. Refrag: Rethinking rag based decoding, 2025. URL https://arxiv.org/abs/2509.01092.

Michael Poli, Stefano Massaroli, Eric Q. Nguyen, Daniel Y. Fu, Tri Dao, Stephen A. Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, 2023.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint*, 2019. URL https://arxiv.org/abs/1911.05507.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: your language model is secretly a reward model. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Aurko Roy*, Mohammad Taghi Saffar*, David Grangier, and Ashish Vaswani. Efficient content-based sparse attention with routing transformers, 2020. URL https://openreview.net/forum?id=B1gjs6EtDr.

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse Sinkhorn attention. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9438–9447. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/tay20a.html.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

Howard Yen, Tianyu Gao, and Danqi Chen. Long-context language modeling with parallel context encoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2588–2610, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.142. URL https://aclanthology.org/2024.acl-long.142/.

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, 2020.

Beyond Static Sink: Reinforcement-Learned KV Retention for Streaming LLMs

Adam Kuca, Peizhou Huang, Chaoyu Zhang, Xiangzhong Ye

Department of Computer Science and Engineering, University of Michigan

Introduction & Motivation

Large Language Models (LLMs) in real-time or streaming settings especially face a growing bottleneck in memory management. KV cache grows linearly with sequence length since the model must cache all past Key–Value (KV) pairs. With each new token, memory requirements grow, and latency increases as throughput slows.

The Limitations of Current Approaches

- **Sliding Window:** Generally leads to instability, high perplexity, and forgetting issues as early tokens, including the input text, are dropped.
- **StreamingLLM (Baseline):** Uses attention sinks to stabilize decoding while applying a fixed sliding-window pattern to the remaining tokens. Tokens that fall outside the retained region are permanently removed, following a static pruning rule that does not adapt to content.

Problem Statement: Static approaches, such as StreamingLLM, ignore the fact that not all tokens are equally important to future generations. As a result, they cannot selectively retain important global context. This project explores whether we can learn a **lightweight policy** that dynamically chooses which KV entries to keep based on their content and importance for future generations, rather than fixed heuristics.

Methodology: Learned Cache Control

We formulate KV-cache retention as a decision-making process. At each step t, the cache is split into three regions:

1. **Sink ($\mathcal{S}$):** First N tokens, always kept for stability.
2. **Window ($\mathcal{W}$):** Most recent tokens, always kept for local context.
3. **Remote ($\mathcal{R}$):** Intermediate history (ignored by StreamingLLM). **This is where selective retention happens.**

The policy must learn to solve a **ranking problem**: given a budget B_{remote}, select the top-B most important tokens from $\mathcal{R}$ to keep.

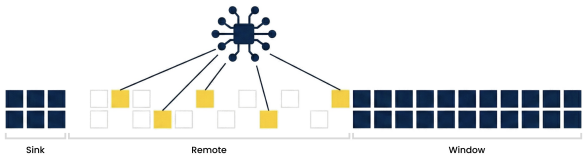


Figure 1. Cache architecture. The policy dynamically ranks and selects tokens from the Remote region.

Training Framework

Stage 1: Oracle Data Collection (Teacher-Student)

We use a larger model (Qwen2.5-7B) as a Teacher to search for the "Optimal Cache" and a smaller model (Qwen2.5-0.5B) for input features. For each prefix, the Teacher performs a **horizon look-ahead** ($H = 64$) to evaluate candidate masks. The remote set with the lowest cumulative NLL is selected as optimal and used later for training.

Stage 2: Behavioral Cloning (BC)

The policy is initialized via supervised learning on oracle labels, formulated as a binary keep/drop classification task. We train using standard Binary Cross-Entropy (BCE) loss:

$$\mathcal{L}_{BC} = -\sum [y \log \pi(x) + (1-y)\log(1-\pi(x))] .$$

Stage 3: Direct Preference Optimization (DPO)

Because BC treats tokens independently, we refine the policy with DPO to learn **relative importance**. For each prefix, preference pairs are constructed within the remote region: oracle-selected tokens are preferred over non-selected ones. DPO optimizes the policy to assign higher scores to preferred tokens without requiring an explicit reward model.

Implementation: Policy & State

State Representation (x_i): To minimize overhead, we avoid additional computation by using existing features.

- **Semantic Feature ($\mathbf{h}_i$):** Cached hidden state from the Student model.
- **Positional Norm:** Normalized absolute position (i/T).
- **Relative Age:** Normalized distance from current step ($(t-i)/L_{max}$).

Policy: To ensure negligible overhead relative to the LLM's decoding cost, the policy is designed as a lightweight, specialized MLP with Gated Residual connections to effectively fuse high-dimensional semantic features with scalar positional features.

Inference Flow: During inference, the policy assigns a scalar score to each remote token. We retain the top-B_{remote} tokens and permanently evict the rest.

Visual Analysis: Attention & Retention

We compare the retention patterns of the static baseline (StreamingLLM) with those of our learned policy to interpret what each model perceives as important.

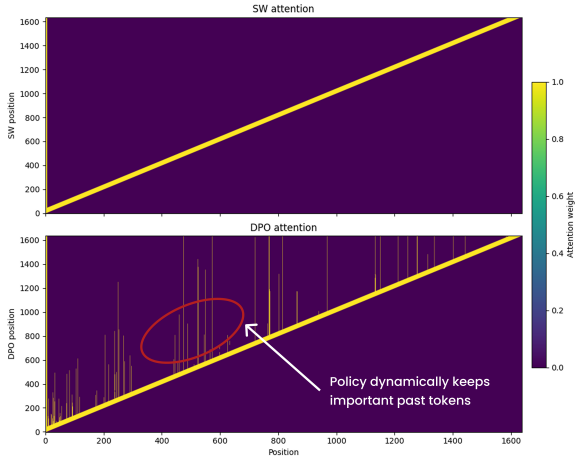


Figure 2. **Top:** StreamingLLM uses a fixed diagonal pattern. **Bottom:** The policy learns dynamic retention, preserving specific sequences in the remote history that correspond to important context.

Quantitative Evaluation

We evaluated the trained policy on the PG-19 dataset, averaging over 100 samples. Our experiments focus on more **extreme compression cases**, testing the policy's ability to select the most important tokens under strict budget constraints.

Results at Max Compression Tested (Cache Size = 32)

Strategy	Perplexity (PPL)	ΔPPL
Streaming Sliding-Window (SW)	44.39	-
RL-Cache (BC)	43.59	0.80
RL-Cache (DPO)	**43.02**	**1.37**

Observation: With a total cache size of only 32 tokens (1.56% of context), the DPO-refined policy outperforms the static baseline by **1.37 PPL points**. This confirms that for strict memory budgets, intelligent selection outperforms StreamingLLM's static recency heuristic.

Impact of Cache Size

We analyzed how performance scales as we increase the memory budget.

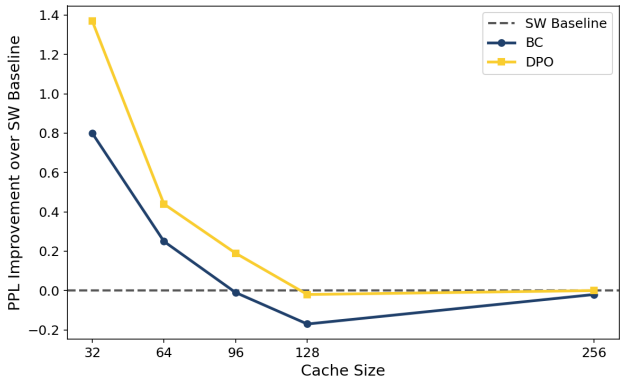


Figure 3. Perplexity improvement (Baseline - Ours) across cache sizes 32, 64, 96, 128, 256. Higher values are better. The learned policy shows the largest gains at small cache sizes (< 96), converging to the baseline as the cache size increases.

As the cache size grows (> 128), the "Window" region captures most local dependencies naturally, reducing the relative impact of the "Remote" policy. The method is most effective for **high-compression scenarios**.

Latency Overhead Analysis

A key concern with dynamic compression is the cost of the decision-making policy.

Policy Architecture: 3-Layer MLP ($d_{model} \to 256 \to 1$).

- **Policy FLOPs:** $\approx 2 \times 10^6$ operations per step.
- **Attention FLOPs (Saved):** $\approx 2 \times d_{model} \times (T_{full} - T_{cache})$.

For a sequence length of $T = 2048$ and cache size 32, the policy overhead is **< 0.1%** of the total inference time. The method provides "smart" compression with effectively zero runtime penalty compared to static heuristics.

Conclusion

This project demonstrates that **learned KV retention** is a viable alternative to static heuristics for streaming language models.

- **Dynamic Selection:** We successfully trained a policy to rank remote tokens based on utility rather than just recency.
- **Performance:** The method matches or exceeds StreamingLLM accuracy, particularly in low-memory regimes (e.g., Cache=32), validating the hypothesis that adaptive mechanisms can improve efficiency.
- **DPO Effectiveness:** Preference optimization consistently outperformed supervised behavioral cloning, demonstrating the value of ranking-based training.

Future work could focus on applying this method to larger models and exploring end-to-end differentiable cache compression.

References

- Xiao et al., "Efficient Streaming Language Models with Attention Sinks" (2024).
- Lin et al., "REFRAG: Rethinking RAG with RL" (2025).
- Rae et al., "Compressive Transformers for Long-Range Sequence Modelling" (2019).

{adamkuca, peizhou, chaoyz, yxiangz}@umich.edu