# 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.

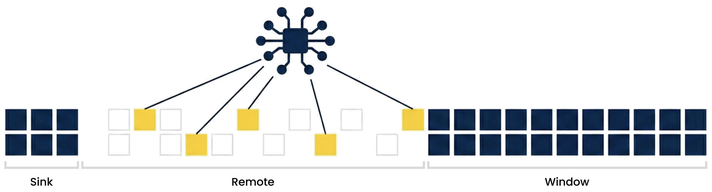



Figure 1. Cache architecture. The policy dynamically ranks and selects tokens from the Remote region.

## 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.

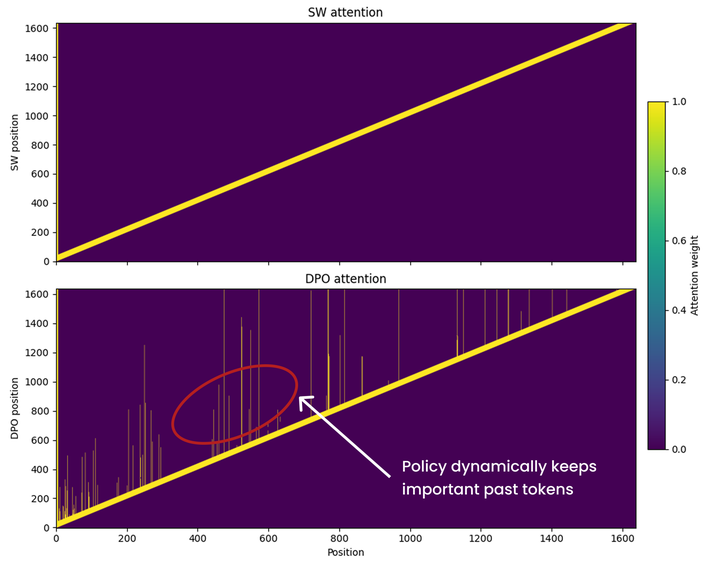



Figure 2. **Top:** StreamingLLM uses a fixed diagonal pattern. **Bottom:** The policy learns dynamic retention, preserving specific sequences in the remote history that correspond to important context.

## 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.

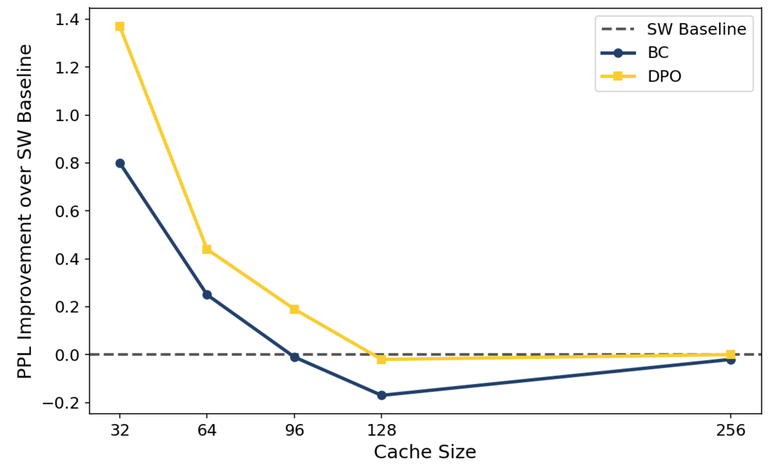



Figure 3. Perplexity improvement (Baseline - Ours) across cache sizes 32, 64, 96, 128, 256. Higher values are better. The learned policy shows the largest gains at small cache sizes ($< 96$), converging to the baseline as the cache size increases.

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