TOKENBUTLER: TOKEN IMPORTANCE IS PREDICTABLE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) rely on the Key-Value (KV) Cache to store token history, enabling efficient decoding of tokens. As the KV-Cache grows, reading it at every decoding step becomes a major memory-bandwidth bottleneck. However, only a small subset of tokens contribute meaningfully to each decoding step, which is an opportunity to alleviate this bottleneck. A key challenge in finding these *critical tokens* is that they are dynamic, and heavily input query-dependent. Existing methods evict tokens permanently, risking quality; retrieve coarse pages, failing on dense, context-rich tasks; or score every token in the full head dimension, at prohibitive cost. Additionally, many existing KV-Cache sparsity methods rely on inaccurate proxies for token importance. To address these limitations, we introduce **TokenButler**, a high-granularity, query-aware predictor that learns to identify these critical tokens. TokenButler predicts low-dimensional *importance queries* at a fixed depth stride, and combines them with a learned projection of the *real KV-cache keys* to score tokens cheaply, enabling dynamic per-token selection under a fixed budget while preserving the full KV cache. We train TokenButler by distilling the model's masked causal attention distributions, optimizing a lightweight predictor with minimal parameter overhead. On a novel synthetic small-context co-referential retrieval task, designed as a targeted stress test of eviction- and page-based methods, TokenButler achieves near-oracle accuracy where existing methods fail. On general-purpose long-context benchmarks (RULER, LongBench, LongMemEval), TokenButler achieves competitive or superior performance, up to $\approx 1.6\times$ on-GPU speedup using our proposed *prediction interval with neighbor fetching* that amortizes predictor cost while maintaining accuracy within $\approx$1.1%, and up to $7.6\times$ reduction in latency compared to Dense Attention with CPU offloading, enabling efficient decoding for long contexts in limited GPU memory systems. Code `is available`.

1 INTRODUCTION

As Large Language Models (LLMs) become more widely used (Thoppilan et al., 2022; Yuan et al., 2022; Wei et al., 2022; Zhang et al., 2024), recent advances have extended their context lengths to 64K–1M tokens. The Key-Value (KV) cache grows linearly with the context length, becoming a major memory and bandwidth bottleneck. Prior work addresses this via sparsity, quantization, efficient attention, or low-rank compression (Child et al., 2019; Choromanski et al., 2021; Katharopoulos et al., 2020; Shazeer, 2019; Pope et al., 2023; Sun et al., 2025; Akhauri et al., 2024b; Chen et al., 2025).

Token pruning methods fall into three categories: **(1)** *Static strategies* that cap the KV-Cache with fixed rules on removing tokens (StreamingLLM (Xiao et al., 2024), Sliding Window (Jiang et al., 2023)); **(2)** *Adaptive eviction* that permanently drops low-importance tokens (H_2O, SnapKV (Zhang et al., 2023; Li et al., 2024)); and **(3)** *Adaptive dynamic strategies* that preserve the full cache but selectively access a subset at decode time, reducing bandwidth at the cost of higher storage (Quest (Tang et al., 2024), TokenSelect (Wu et al., 2025b)).

Token importance is highly query-dependent (Tang et al., 2024) and each strategy has its limitations: static methods lack query-awareness, and adaptive eviction permanently discards tokens that may become relevant again in *co-referential* contexts (Vodrahalli et al., 2024). A context is co-referential when text introduced earlier is referenced again later, requiring accurate retrieval over the earlier mention. In such scenarios, token importance is non-monotonic: a critical entity may remain dormant for thousands of steps, only to become the most relevant token when a later query triggers a callback.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

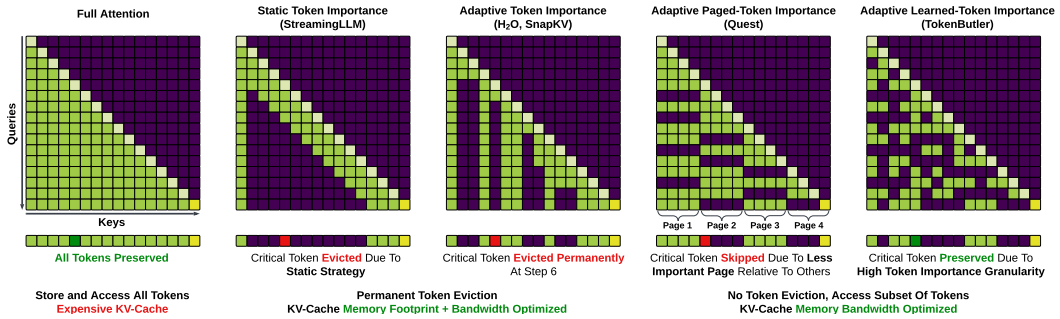


Figure 1: Full-Attention preserves all tokens, enabling access to the critical token (dark green) during the last decode step. Static strategies like StreamingLLM will not be able to access this token. Methods like H_2O may have evicted the token at an earlier decode step, if deemed unimportant. Paged-Token importance may cause a page-miss of a critical token in context dense tasks. **TokenButler** can effectively predict critical tokens, and can be leveraged by existing methods to offer both high-granularity and cheap importance estimation.

Adaptive dynamic strategies preserve the full cache, but trade granularity for efficiency: page-based methods score coarse *token groups* (Tang et al., 2024), and token-granular ones score in the full head dimension (Wu et al., 2025b). Figure 1 illustrates these failure modes. TokenButler addresses this failure mode directly: it keeps the full KV-cache and re-scores every token at each query with a predictor trained to recognize such dormant-then-critical tokens (Section 4.1).

Token importance can be quantified by several *metrics*: recency, aggregate attention scores, and others listed in Table 1, which sparsity methods use to guide eviction or retrieval decisions, with trade-offs tied to eviction or coarse token grouping. We instead propose a *learned* metric that gives fine-grained importance estimates, and a system, **TokenButler**, that makes use of it. Our contributions are summarized as:

- A lightweight predictor that learns token importance via attention distillation; *lightweight* refers jointly to $< 1\%$ parameter overhead, a 1K-token training context, and a frozen base LLM (no full-model training).
- A synthetic co-referential benchmark exposing failures of existing methods at contexts as small as 300 tokens; TokenButler achieves near-oracle accuracy.
- Competitive or superior results on RULER and LongBench at 64K context after using max length of 1K for distillation, and up to 7.6$\times$ latency reduction over Dense Attention with CPU offloading.
- A *prediction interval* scheme that amortizes predictor cost by $16\times$, with *neighbor fetching* to maintain accuracy within $\approx$1.1%, bringing on-GPU speedup over Dense Attention to $1.6\times$.

2 Related Work

Transformers exhibit strong contextual behavior: head and neuron importance depends heavily on the input query. Liu et al. (2023); Akhauri et al. (2024a) train small predictors to prune neurons and heads on a per-query basis using magnitude or gradient-based *metrics*. We draw the same distinction between importance *metrics* and the pruning *methods* that act on them.

This contextual behavior applies to token importance by design, as attention explicitly captures the tokens relevant to a query. However, pruning tokens is far more expensive than pruning heads: for a transformer with N layers, H heads and L past tokens, any importance *metric* must be evaluated for $N \times H \times L$ tokens at *every decode step*.

Table 1 summarizes how prior work *co-designs* metrics with methods of token sparsity. *Purely static strategies* such as StreamingLLM (Xiao et al., 2024) use *recency* as the metric, attending to a sliding local window plus initial anchor tokens. *Eviction* methods such as H_2O (Zhang et al.,

Table 1: Metrics for token importance

Method	Metric
StreamingLLM	Recency-based sliding window with initial attention-sink tokens
H_2O	Accumulated Attention Score for Token Eviction
SnapKV	Pooled Attention Score over a Fixed Window for Token Eviction
Quest	Per-Page Min-Max Key Bound on the Query-Key Product, for Page Loading
TokenSelect	Full-Dimensional Query-Key Dot Product for Token Loading
TokenButler	Predicted Importance for Fine-Grained Token Loading

2023) and SnapKV (Li et al., 2024) instead use attention scores to permanently evict low-importance tokens. This is a major limitation when a task requires synthesizing information distributed across the context (Vodrahalli et al., 2024), a failure mode documented directly by the KV-cache-centric benchmark SCBench (Li et al., 2025): a token that only becomes important later may already have been evicted. Moreover, accumulating attention scores over a window is biased towards frequently attended tokens and penalizes 'rare-event' tokens that have high utility but low aggregate attention. *Adaptive dynamic strategies* such as Quest (Tang et al., 2024) preserve all tokens and fetch a query-dependent subset at each step. To avoid computing full attention scores, Quest stores tokens in pages and scores each page by an upper bound derived from its elementwise min and max keys; this reduces bandwidth but not memory footprint, and its page-granular sparsity fails on the co-referential tasks we study. TokenSelect (Wu et al., 2025b) also preserves all tokens and selects them by query-key dot products, skipping the selection when consecutive queries are similar; however, scoring in the full head dimension is costly. We compare directly against TokenSelect in accuracy (Appendix F) and on-GPU throughput (Appendix J).

These attention-based metrics are tightly coupled to their methods, requiring eviction, paging, or expensive scoring. By contrast, *TokenButler* learns a lightweight predictor ($\sim$1% of the LLM) that cheaply approximates token-level attention logits via low-dimensional QK projections, preserving fine-grained per-token control.

Concurrent to our work, DeepSeek-V3.2 (DeepSeek-AI, 2025) introduces a *Lightning Indexer*, a lightweight FP8 scoring module that selects important tokens for sparse attention. While the high-level motivation is similar, the two approaches differ in important ways. First, DeepSeek Sparse Attention requires continued training of the full model ($\sim$944B tokens) to adapt to the sparse regime, whereas TokenButler trains only a small external predictor on a frozen off-the-shelf LLM. Second, the Lightning Indexer rescores at every decode step and layer, whereas TokenButler amortizes this cost over *prediction intervals* with neighbor fetching (Section 3.4). Recent published work also trains the token selector. LookaheadKV (Ahn et al., 2026) learns lookahead tokens on a frozen backbone that predict which prompt tokens the response will attend to, and prunes the prompt cache once during prefill; IndexMem (Yang et al., 2026) distills an indexer from attention logits max-pooled over queries and evicts permanently, approximating the discarded entries with a recurrent latent state. Both bound cache *capacity* by eviction; TokenButler distills the full per-query attention distribution and retains every entry, so a token passed over at one step can still be attended exactly at a later one.

3 Methodology

TokenButler (Figure 2) uses a single LLM layer's output to predict token importance across multiple subsequent layers ahead of time, trained by distilling the LLM's own attention distributions.

3.1 TokenButler Design

TokenButler operates at a fixed depth stride G (`producer_frequency`), designating *producer* layers at indices $0, G, 2G, \ldots$. Each producer takes hidden states $\mathbf{H} \in \mathbb{R}^{B\times L\times E}$ and predicts low-dimensional importance queries for the next G *consumer* layers; consumer layer ℓ uses slot $(\ell-1) \bmod G$.

Query prediction. A lightweight MLP produces G slot-specific query vectors per head:

$$\mathbf{Q}_{\text{imp}} = f_\theta(\text{LN}(\mathbf{H})) \in \mathbb{R}^{(BH)\times G\times L\times d'},$$

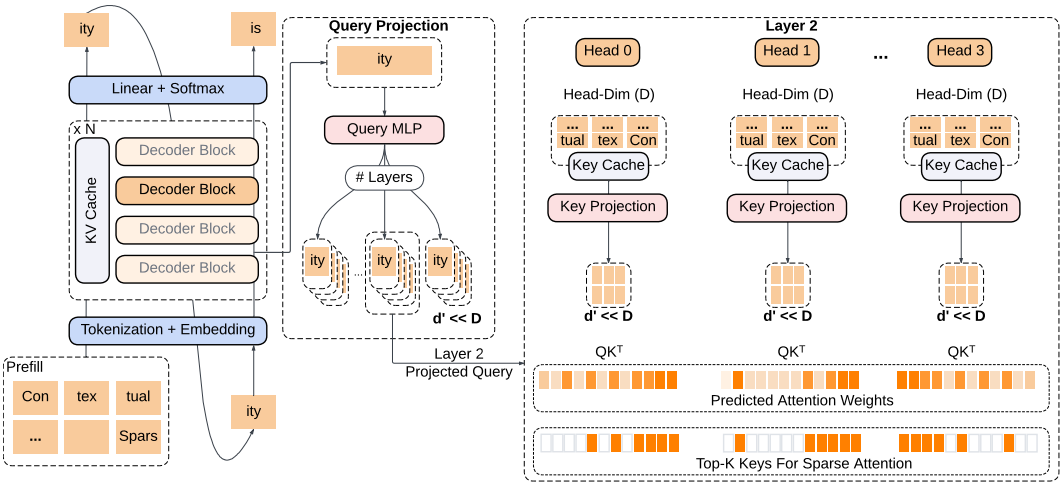


Figure 2: TokenButler predicts low-dimensional importance queries at fixed depth strides (producer layers) and combines them with a learned projection of the real KV-cache keys at each consumer layer to score and select tokens under a fixed budget. Training distills the masked causal attention distribution of the frozen LLM, and inference applies per-token selection while preserving sink tokens and local window tokens.

where $d' \ll E$ is the interaction dimension.

Key-cache projection. For each layer ℓ, we project the *real KV-cache keys* into the same d'-dimensional space via a learned matrix $\mathbf{W}_{\mathrm{K}}^{(\ell)} \in \mathbb{R}^{D \times d'}$:

$$\mathbf{K}_{\mathrm{imp}}^{(\ell)} = \mathbf{K}^{(\ell)} \mathbf{W}_{\mathrm{K}}^{(\ell)} \in \mathbb{R}^{B \times H_{\mathrm{kv}} \times L \times d'}.$$

This anchors scoring to the model's true cached keys while keeping computation low-dimensional.

3.2 Training Objective

We freeze the base LLM and train only TokenButler parameters (the query MLPs and key-cache projection matrices). Given an input sequence, we compute teacher attention logits from the frozen model for each layer and head, and compute TokenButler's student logits using predicted importance queries and projected KV-cache keys. Crucially, both teacher and student logits include the same causal/padding mask prior to normalization.

We distill the *masked causal attention distribution*. Let $\mathbf{A}_{\mathrm{true}}$ be the teacher logits (after adding the attention mask) and $\mathbf{A}_{\mathrm{pred}}$ be the corresponding TokenButler logits. We define:

$$\mathbf{P} = \mathrm{softmax}(\mathbf{A}_{\mathrm{true}}), \qquad \mathbf{Q} = \mathrm{softmax}(\mathbf{A}_{\mathrm{pred}}),$$

and minimize a cross-entropy distillation loss:

$$\mathcal{L}_{\mathrm{CE}} = -\mathbb{E}\left[\sum_k \mathbf{P}_k \log(\mathbf{Q}_k)\right].$$

To make training efficient at long context lengths, we subsample a fixed number R of query positions (loss rows) per sequence: we draw most loss rows from the late-context region and always include the final token, reducing the auxiliary loss cost from $\mathcal{O}(L^2)$ to $\mathcal{O}(RL)$ with $R \ll L$.

3.3 Inference Setup

Building on the critical role of initial and recent tokens (Xiao et al., 2024), we partition the KV-cache into three contiguous buffers: a **Sink Buffer** (first S tokens, always retained), a **Local Window Buffer** (circular buffer of the most recent N tokens), and an **Important Buffer** (dynamically populated by

TokenButler). This layout keeps attention kernel accesses contiguous, avoiding fragmented memory gathers.

To amortize the cost of projecting high-dimensional keys (D) into the d'-dimensional importance space, we exploit the temporal locality of the Local Window Buffer: since the attention kernel always attends to the local window densely, a newly generated token needs no importance score during its first N steps. We therefore defer its projection until it leaves the window, projecting the N most recent keys in a single batch each time the circular buffer wraps around, rather than one key per decode step. Predictor operations are batched similarly: once a producer layer's hidden state is ready, query prediction, importance scoring, and the migration of selected KV pairs into the Important Buffer are performed for all of its consumer layers at once. Batching these operations uses high-bandwidth memory more efficiently and significantly improves throughput.

3.4 Prediction Interval and Neighbor Fetching

While TokenButler's predictor is lightweight, its per-step cost (query prediction, scoring against all projected keys, and KV gather) accumulates over long generations. *Prediction interval with neighbor fetching* reduces this overhead by up to $N\times$ with minimal accuracy loss.

Prediction interval (i=N). Instead of running the predictor at every decode step, we invoke it once every N steps: prediction steps execute the full pipeline (predictor forward pass, scoring, top-B selection, and KV gather into the Important Buffer), while the intervening $N-1$ steps reuse the *stale* selection. The Local Window Buffer is still updated every step, so the most recent W tokens are always available. This reduces predictor invocations over a generation of T tokens from T to approximately T/N.

Neighbor fetching. Reusing stale selections risks missing tokens whose importance has shifted since the last prediction. Since important information often spans consecutive tokens (e.g., multi-token entities or reasoning chains), we expand each selected token with a spatial neighbor using a cluster-aware rule: consecutive selected indices form clusters, and each neighbor is placed just past its cluster boundary to maximize coverage. This yields $2B$ unique positions per prediction step (the sparse buffer is sized accordingly) and protects against importance drift between refreshes, as the neighbors of previously selected tokens are likely to cover newly important positions. All results with $i > 1$ employ neighbor fetching.

4 Experiments

Across all experiments, we use the same predictor architecture: a producer every G=4 layers, interaction dimension d'=16, and a two-layer MLP with hidden size 512. The stride G bounds the producer-to-consumer distance; since recall improves the closer a predictor sits to the layers it scores (Appendix B), G=4 keeps every consumer within a few layers of a producer while adding few producers. d'=16 is a deliberately small interaction dimension ($d' \ll E$) that affects only which tokens are selected, not the attention computation. The hidden size of 512 keeps the predictor at 0.3-0.9% of the base parameters (Table 6); larger predictors trade overhead for recall (Appendix C). The base LLM is frozen; only the predictor is trained ($\texttt{lr}$=10^{-3}) on a mixture of web, code, and long-context QA data at sequence length 1K. Since the predictor projects the key-cache directly, it generalizes to 64K contexts without long-context fine-tuning. Full training details are in Appendix A.

Token Selection Policy. TokenButler converts its per-layer importance scores into a keep/drop decision under a fixed budget: the S sink tokens and the W most recent tokens are always retained, and among the remaining *candidate* tokens we retain the top-x by predicted importance ($\texttt{xtok}$). As is standard for decode-time KV-sparsity methods (Tang et al., 2024), prefill and the first decode token are computed densely, and pruning is applied from the subsequent decode steps onward.

4.1 Evaluation On a Synthetic Task for Token Retrieval

As a targeted stress test of eviction-based and page-based methods (RULER, LongBench, and LongMemEval in Section 4.2 serve as the general-purpose evaluation), we evaluate TokenButler on a

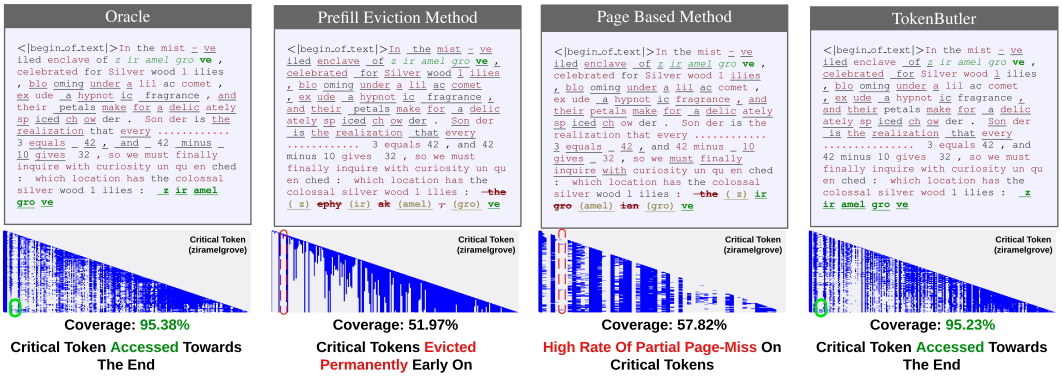


Figure 3: Sample behavior of different KV-Sparsity methods on our synthetic co-reference resolution task. TokenButler outperforms prefill eviction and page-based methods that have clear failure modes due to permanently dropping tokens, or fetching tokens with page-size granularity respectively.

Synthetic Benchmark Template and Sample

```
<contextual lead> <location>
<philosophical statement>
<culinary statement> <math
problem> <location prelude>
«location»
```

```
Shrouded in fog, ....  place
is:  wraithspire.  In the ...
wisdom to sprout.  Savor the
...  home-cooked love.  If we
compute ...  gives us 16.  Which
location ...  up the shore?
«wraithspire»
```

Table 2: Accuracy and coverage (%) of different KV-sparsity methods on our synthetic dataset. TokenButler outperforms eviction and page-based methods, and approaches Oracle performance.

Llama	Oracle		Token Eviction		Page-Based		TokenButler	
	Acc.	Cov.	Acc.	Cov.	Acc.	Cov.	Acc.	Cov.
1B	49.00	84.32	1.00	32.50	0.00	19.78	**48.94**	**84.02**
3B	81.00	95.38	10.00	51.97	6.00	57.82	**80.20**	**95.23**
8B	77.00	93.47	3.00	37.50	0.00	46.98	**76.17**	**92.59**

difficult synthetic task inspired by Multi-Round Co-reference Resolution (Vodrahalli et al., 2024), using concise sequences (< 512 tokens). The model must recall a fictional location mentioned in a *contextual lead*, then referenced again after several distracting statements. By the time the location needs to be mentioned again after the location prelude, several tokens may have intervened, making it likely that the location tokens may have been evicted. Coarse-grained retrieval schemes risk not finding the entire location as it may be split across pages. This setup mimics conversation-like scenarios. It is especially challenging for token sparsity methods, since prematurely discarding or overlooking the location tokens can irreversibly break the final reference, leading to incorrect or incomplete retrieval of the location name. Our data generation procedure is further detailed in Appendix H

Since every head may evict tokens based on their importance, we present the attention map for the first head of the 3rd layer (a random choice) in Figure 3. We observe there as well as in Table 2 that (i) prefill eviction methods, e.g. H_2O, have low accuracy because they permanently evict older tokens (the location name) once new context is being decoded. (ii) page-based methods, e.g. Quest, **very often** lose part of the location name if it straddles a page boundary in this context-dense example. *Coverage* counts the fraction of correctly-predicted location tokens (e.g., 3/4 tokens correct yields 0.75 coverage but 0 accuracy). Eviction and page-based methods recover 30-50% of tokens but rarely all of them, leading to low accuracy (Table 2).

4.2 Long Context Evaluation

We evaluate TokenButler under a fixed decode-time KV-access budget corresponding to an effective $8\times$ compression at 64K context. Specifically, at each decode step we always retain a small set of *sink* prefix tokens (128) and a *sliding-window* of most-recent tokens (local window = 256), and allocate

Table 3: Comparing TokenButler with KV-Cache sparsity, quantization and low-rank compression methods from prior research on LongBench. K.V. BW ($\times \downarrow$) is the controlled KV-cache access-bandwidth budget under which each method is evaluated, not a measured speedup. Bold marks the best non-dense method in each column.

Method	K.V. BW ($\times \downarrow$)	NarQA	Qasper	MFQA	HotpotQA	2WikiMQA	Musique	GovRpt	QMSum	MultiNews	SAMSum	PassRet	Avg.
Llama-3.1-8B-Instruct													
Dense	1.00	30.64	46.64	55.42	58.17	47.76	28.23	34.51	25.61	25.43	23.51	100.00	43.27
MiniCache	1.30	18.11	20.41	27.51	38.80	21.37	20.19	21.09	20.85	19.60	22.75	98.00	29.88
StreamingLLM	8.00	26.25	29.32	34.56	49.99	42.47	20.92	26.29	21.90	22.63	23.87	93.50	35.61
SnapKV	8.00	30.34	45.56	53.62	**57.92**	47.50	26.96	28.99	**25.33**	23.95	24.91	99.50	42.23
PyramidKV	8.00	31.83	44.36	52.56	56.38	48.35	27.23	28.63	25.06	23.49	24.80	99.50	42.02
KIVI	7.10	29.84	41.49	51.28	57.27	42.54	27.54	33.28	25.01	23.90	25.88	98.17	41.47
Single SVD	8.40	15.52	38.36	37.14	30.04	19.70	22.06	19.52	22.35	18.91	22.82	85.00	30.13
xKV	8.03	**32.85**	45.62	54.69	51.74	36.38	28.33	31.32	24.49	22.61	26.26	**100.00**	41.30
TokenButler	8.00	30.73	**48.86**	**56.45**	57.77	**50.82**	**28.34**	**34.34**	25.30	**26.91**	**29.22**	**100.00**	**44.43**
Qwen2.5-7B-Instruct-1M													
Dense	1.00	29.21	43.78	48.58	60.92	53.29	33.68	33.23	23.24	23.53	43.21	100.00	44.79
MiniCache	1.30	7.98	16.47	9.42	21.78	15.46	10.55	8.83	7.94	3.84	11.32	5.00	10.78
StreamingLLM	8.00	24.13	34.99	27.47	47.55	45.13	21.44	26.76	19.34	21.15	44.11	29.00	31.01
SnapKV	8.00	28.68	44.51	47.43	**60.88**	51.91	32.17	29.95	23.05	21.79	**45.67**	**100.00**	44.19
PyramidKV	8.00	**29.93**	41.29	47.21	60.16	51.98	32.79	27.21	22.88	19.86	45.00	**100.00**	43.48
KIVI	7.10	26.43	35.78	38.40	46.17	45.08	23.79	24.62	22.41	16.98	39.52	63.50	34.79
Single SVD	8.40	27.88	41.84	47.66	52.73	48.76	28.46	28.71	22.24	21.18	33.18	65.00	37.97
xKV	8.03	28.78	42.98	47.43	58.79	52.42	32.69	33.02	**23.67**	23.18	38.62	98.00	43.60
TokenButler	8.00	29.39	**47.27**	**52.50**	59.67	**56.01**	**34.58**	**33.32**	23.14	**24.04**	35.09	**100.00**	**45.00**

Table 4: RULER evaluation on Llama-3.1-8B-Instruct at 64K context length. K.V. BW ($\times \downarrow$) is the controlled reduction in key-value cache access bandwidth under which each method is evaluated, not a measured speedup. Bold marks the best non-dense method in each column.

Method	K.V. BW ($\times \downarrow$)	N-S1	N-S2	N-MK1	N-MK2	N-MQ	N-MV	QA-1	QA-2	VT	FWE	Avg.
Llama-3.1-8B-Instruct												
Dense	1.00	100.00	100.00	98.96	97.92	98.96	97.66	83.33	59.38	97.29	85.42	91.89
KVzip	8.00	**100.00**	**100.00**	8.33	70.83	52.60	78.12	62.50	**58.33**	73.96	75.35	68.00
SnapKV	8.00	**100.00**	**100.00**	98.96	94.79	**100.00**	**97.66**	**83.33**	**58.33**	**95.00**	68.75	89.68
Quest	8.00	90.75	90.63	96.88	87.50	94.27	85.42	**83.33**	57.29	77.71	**81.94**	84.57
xKV	8.03	**100.00**	96.88	97.92	**97.92**	96.09	96.62	78.13	56.25	86.67	78.47	88.50
TokenButler	8.00	**100.00**	**100.00**	**100.00**	96.88	98.96	94.27	**83.33**	57.29	91.46	77.08	**89.93**

the remaining budget to a query-dependent, per-token selection from the rest of the context so that the total number of tokens accessed per layer is capped at $\approx$ 8K.

Table 4 shows the results of the RULER benchmark (Hsieh et al., 2024). TokenButler achieves competitive performance on Llama-3.1-8B-Instruct, outperforming baselines like Quest and KVzip (Kim et al., 2025) while matching stronger methods like SnapKV and xKV. We also evaluate the `Qwen2.5-7B-Instruct-1M` model in Table 5, where TokenButler achieves the best average score among all methods. Both evaluations are done at a context-length of 64K.

LongBench (Bai et al., 2024). We evaluate the `Llama-3.1-8B-Instruct` and `Qwen2.5-7B-Instruct-1M` models across a wide range of baselines that alleviate KV-Cache bandwidth requirements. Specifically, we evaluate KV-compression methods such as PyramidKV (Cai et al., 2025), Single SVD (Chang et al., 2026), MiniCache (Liu et al., 2024a) and xKV (Chang et al., 2026), quantization with KIVI (Liu et al., 2024b) and sparsity with StreamingLLM (Xiao et al., 2024), SnapKV (Li et al., 2024) and TokenButler. We find that TokenButler offers better accuracy than prior methods from Table 3.

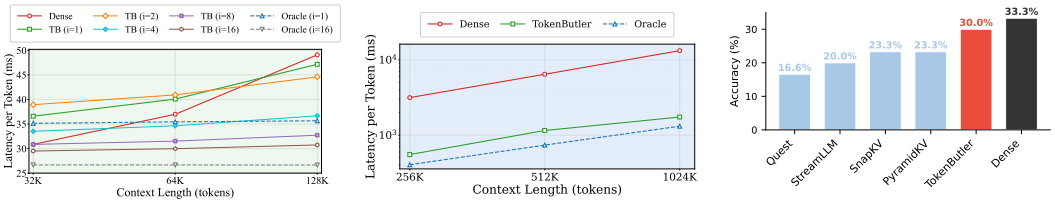
LongMemEval (Wu et al., 2025a). To test, on real conversational data, the retrieval of information that becomes relevant only long after it was introduced, we evaluate `Llama-3.1-8B-Instruct` with TokenButler at an 8K token budget on LongMemEval (using the cleaned release[1], which removes noisy history sessions that interfere with answer correctness). Table 10 in Appendix G shows that TokenButler stays close to Dense on every question type (26.6 vs. 27.4 overall).

Reasoning models. The emphasis on long decode has increased, as reasoning models must emit long chain-of-thought (CoT) traces before arriving at an answer. These generated CoTs can signifi-

[1] https://huggingface.co/datasets/xiaowu0162/longmemeval-cleaned

Table 5: RULER evaluation on Qwen2.5-7B-Instruct-1M at 64K context length: TokenButler vs. methods which sparsify or compress the KV-Cache. Bold marks the best non-dense method in each column.

Method	K.V. BW (× ↓)	N-S1	N-S2	N-MK1	N-MK2	N-MQ	N-MV	QA-1	QA-2	VT	FWE	Avg.
Qwen2.5-7B-Instruct-1M												
Dense	1.00	100.00	100.00	100.00	100.00	100.00	95.83	84.38	60.42	90.63	86.81	91.81
MiniCache	1.30	26.04	0.00	0.00	0.00	0.00	0.00	12.50	14.58	0.42	3.47	5.70
Single SVD	8.40	**100.00**	97.92	96.88	**98.96**	97.40	**91.15**	64.58	56.25	73.75	61.46	83.84
xKV	8.03	**100.00**	**100.00**	**100.00**	**98.96**	**100.00**	90.63	80.21	58.33	82.08	81.94	89.22
StreamingLLM	8.00	15.63	12.50	12.50	9.38	14.84	17.71	46.88	43.75	13.13	**89.24**	27.56
SnapKV	8.00	**100.00**	96.88	97.92	31.25	95.31	83.07	**84.38**	**59.38**	**91.25**	80.56	82.00
PyramidKV	8.00	**100.00**	93.75	96.88	16.67	90.37	80.73	**84.38**	**59.38**	89.17	76.39	78.77
KIVI	7.10	0.00	2.08	3.13	13.54	0.00	0.78	48.96	43.75	36.46	40.63	18.93
TokenButler	8.00	**100.00**	**100.00**	**100.00**	96.88	99.47	90.63	**84.38**	58.33	89.17	82.64	**90.15**



(a) Latency, KV cache resident in GPU memory. TB = TokenButler, i = prediction interval.

(b) Latency, KV cache offloaded to CPU.

(c) AIME24 accuracy.

Figure 4: (a, b) Decoding efficiency on `Llama-3.1-8B-Instruct` with a sparse token budget of 8K. (c) AIME24 accuracy on `DeepSeek-R1-Distill-Llama-8B` under a fixed KV access budget (Section 4.2).

cantly slow down decoding by stressing memory bandwidth with excessive KV-token loading. We train TokenButler on `DeepSeek-R1-Distill-Llama-8B` (Guo et al., 2025) and evaluate on AIME24 (Mathematical Association of America, 2024) with a decode budget of 8K tokens. Across all methods, we fix a maximum KV access budget of 2179 tokens per step (32 sink tokens and 128 local window tokens). Since most AIME24 problems require fewer than 160 prefill tokens, this setting is decode-heavy with a relatively small prefill. Figure 4c summarizes accuracy: TokenButler attains 30.0% accuracy, approaching dense decoding (33.3%) and outperforming other KV-sparsity baselines under the same budget. Since AIME24 contains only 30 problems, we additionally report a lower-variance multi-sample metric: with sampling (temperature 0.6, top-p 0.95), Avg@8 over seeds 0-7 is $33.75 \pm 7.22\%$ for TokenButler and $33.75 \pm 7.44\%$ for Dense, i.e., TokenButler stays on par with dense decoding in this decode-heavy regime.

4.3 Efficiency

We measure per-token decode latency on `Llama-3.1-8B-Instruct` (Nvidia A6000, budget = 8K tokens) using prediction interval with neighbor fetching (Section 3.4). For contexts exceeding 128K we employ CPU offloading; we also report an Oracle (zero-overhead selection at i=1) as a lower bound.

Accuracy impact. We evaluate the accuracy-efficiency tradeoff on RULER at 64K context length. As shown in Table 8 (Appendix D), increasing the prediction interval from every step (i=1) to every 16 steps (i=16) reduces average accuracy by only 1.1% (89.9% → 88.8%), while reducing predictor compute by 16×. Even at i=8, accuracy (88.5%) remains competitive with xKV (88.5%) from Table 4.

On-GPU efficiency. Figure 4a shows the on-GPU latency impact. The baseline TokenButler (i=1) incurs predictor overhead comparable to Dense Attention, but with prediction interval this overhead is amortized. At 128K context, i=16 achieves 30.7ms per token, a 1.6× speedup over Dense (49.1ms).

Notably, prediction interval with neighbor fetching achieves ≈1.4× speedup over the Oracle (i=1), which re-selects at every decode step similar to DeepSeek Sparse Attention (DeepSeek-AI, 2025).

CPU-offloading efficiency. In the CPU-offloading regime (≥256K), the bottleneck shifts to KV data transfer from host memory rather than predictor compute. Consequently, TokenButler with i=1 already approaches Oracle performance, and increasing the prediction interval does not yield significant additional gains. As shown in Figure 4b, by drastically reducing the volume of transferred KV data, TokenButler achieves 7.6× lower latency than Dense Attention at 1M context. At 256K, this translates from ≈3.2s per token (Dense) to ≈0.6s (TokenButler), enabling real-time long-context inference on GPU memory-limited systems.

Comparison to TokenSelect and serving-stack integration. We note that TokenButler reduces decode-time KV access *bandwidth*, the dominant cost of memory-bound decoding, and not only compute; the on-GPU gains above are obtained with the cache fully resident in GPU memory. Appendix J compares on-GPU throughput against TokenSelect (Wu et al., 2025b), which scores tokens with full-dimensional query-key dot products: TokenButler tracks Oracle throughput and achieves up to 3.1× higher throughput because it scores in d'=16 dimensions, while also being more accurate on RULER across context lengths (Appendix F). Finally, to verify feasibility in a production serving stack, we integrated a prototype of TokenButler into SGLang (Zheng et al., 2024), whose hierarchical KV-cache system HiSparse[2] offloads inactive entries to host memory, keeping a hot device buffer of frequently accessed KV regions. Without further performance tuning, the prototype reaches accuracy parity with dense decoding (1.00 on RULER NIAH at a 12.5% KV budget) and a 1.3× decode speedup at 96K context for `Llama-3.1-8B-Instruct` on a single A6000 GPU.

5 Limitations

Capacity vs. accuracy. TokenButler deliberately sits on the accuracy side of the eviction trade-off: retaining the full KV-cache means no token that may matter later is discarded, and decode-time KV bandwidth and compute are reduced, but the KV-cache *capacity* footprint is not. Capacity-bound deployments must pair it with CPU offloading (Section 4.3) or a hierarchical KV-cache system such as SGLang HiSparse. **Training and integration.** A predictor is trained once per backbone (under 10 hours on one A6000, Table 6); this one-time cost is small but, unlike training-free heuristics, not zero, and the serving stack must support sparse per-token KV access. Prefill and the first decode step remain dense, so TokenButler targets decode-heavy workloads. **Small budgets.** Recall degrades at very small budgets: on `Llama-3.1-8B-Instruct`, Recall@10% is ≈47% and Recall@5% is ≈42% (Appendix C reports Recall@50% as a function of predictor size). **Hyperparameter sensitivity.** We fix G=4, d'=16, and a hidden size of 512 across all backbones. Appendices B and C characterize the effect of predictor depth and capacity, and Appendix E the sink and window sizes, but a joint sweep of G and d' is left for future work. **Stress test vs. general evaluation.** The synthetic co-reference task (Section 4.1) is constructed to expose the failure modes of eviction and paging; the size of that gap is not a general advantage, which is instead measured by RULER, LongBench, and LongMemEval.

6 Conclusion

We present TokenButler, a lightweight predictor that estimates token importance at fine granularity with minimal overhead. Our co-reference stress test shows that eviction and page-based strategies lose critical tokens that TokenButler preserves with near-oracle accuracy, and on RULER, LongBench, and LongMemEval it matches or outperforms prior methods. With prediction interval and neighbor fetching, predictor cost is amortized 16× within 1.1% accuracy, yielding up to 1.6× speedup over Dense on the GPU and 7.6× with CPU offloading. Because the predictor is trained on a frozen backbone in under ten GPU-hours, TokenButler can be added to existing models without retraining them, and our SGLang prototype shows that it composes with production serving stacks. TokenButler demonstrates that query-aware, fine-grained token selection can replace coarse heuristics with a learned, efficient alternative. Natural next steps are studying how the training data mixture affects the learned metric, evaluating on further reasoning benchmarks, and extending prediction to the prefill phase, where prompt-side methods such as LookaheadKV (Ahn et al., 2026) are complementary.

[2] https://www.lmsys.org/blog/2026-04-10-sglang-hisparse/

AI USE STATEMENT

In this work, we used generative AI tools to *generate synthetic data sets* and to *implement methods*. OpenAI's `gpt-4o-mini` produced the pools of short fictional passages from which our synthetic co-reference benchmark is assembled, following the item categories and sequence template specified in Appendix H; the test sequences themselves are composed by randomly drawing from these pools rather than generated by a model, and `gpt-4o-mini` is not among the models we evaluate. We also used AI coding assistants to help implement portions of the code for some of our experiments. Tasks concerning mathematical claims and proofs are not applicable to this work, which contains none, and neither is translation. Additionally, we used generative AI tools to edit the manuscript to improve its readability and to assist with formatting.

We have reviewed all AI-assisted work. All claims and citations were checked against the underlying results. The generated benchmark passages serve as fictional distractor content: the quantity we score is retrieval of the injected location name, which is fixed by the benchmark construction rather than by any model output. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work proposes a method for improving the efficiency of LLM inference and does not involve human subjects or the collection of personal data. All datasets and benchmarks used (C4, FineWeb-Edu, CodeParrot-Clean, BABILong, WikiText2, RULER, LongBench, LongMemEval, AIME24) are publicly available, and the synthetic co-reference benchmark contains only fictional, machine-generated content. We do not foresee ethical concerns specific to this work beyond those generally associated with large language models.

REPRODUCIBILITY STATEMENT

The TokenButler architecture and training objective are described in Sections 3.1 and 3.2, with full training details (hyperparameters, datasets, and per-model training cost) in Appendix A. The inference setup, token selection policy, and prediction interval with neighbor fetching are specified in Sections 3.3, 4, and 3.4. The generation procedure for the synthetic co-reference benchmark is given in Appendix H, and the efficiency measurement setup in Section 4.3 and Appendices I and J. Source code is available at `https://anonymous.4open.science/r/TokenButler-0B6F`.

REFERENCES

Jinwoo Ahn, Ingyu Seong, Akhil Kedia, Junhan Kim, Hyemi Jang, Kangwook Lee, and Yongkweon Jeon. LookaheadKV: Fast and accurate KV cache eviction by glimpsing into the future without generation. In *The Fourteenth International Conference on Learning Representations*, 2026.

Yash Akhauri, Ahmed F. AbouElhamayed, Jordan Dotzel, Zhiru Zhang, Alexander M. Rush, Safeen Huda, and Mohamed S. Abdelfattah. ShadowLLM: Predictor-based contextual sparsity for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 19154–19167, Miami, Florida, USA, 2024a. Association for Computational Linguistics.

Yash Akhauri, Safeen Huda, and Mohamed S. Abdelfattah. Attamba: Attending to multi-token states. *arXiv preprint arXiv:2411.17685*, 2024b.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 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)*, pp. 3119–3137, Bangkok, Thailand, 2024. Association for Computational Linguistics.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Second Conference on Language Modeling*, 2025.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Hung-Yueh Chiang, Yash Akhauri, Xilai Dai, Huiqiang Jiang, Yucheng Li, Luis Ceze, Kai-Chiang Wu, and Mohamed S. Abdelfattah. xKV: Cross-layer KV-cache compression via aligned singular vector extraction. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2026.

Yuzong Chen, Xilai Dai, Chi-chih Chang, Yash Akhauri, and Mohamed S. Abdelfattah. The power of negative zero: Datatype customization for quantized large language models. *arXiv preprint arXiv:2501.04052v1*, 2025.

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

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Łukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with Performers. In *International Conference on Learning Representations*, 2021.

CodeParrot. CodeParrot dataset cleaned (`codeparrot/codeparrot-clean`). https://huggingface.co/datasets/codeparrot/codeparrot-clean, 2022.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 1286–1305, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction. In *Advances in Neural Information Processing Systems*, volume 38, pp. 167563–167591. Curran Associates, Inc., 2025.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In *Advances in Neural Information Processing Systems*, volume 37, pp. 106519–106554. Curran Associates, Inc., 2024.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV cache-centric analysis of long-context methods. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970. Curran Associates, Inc., 2024.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. MiniCache: KV cache compression in depth dimension for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 139997–140031. Curran Associates, Inc., 2024a.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. Deja vu: Contextual sparsity for efficient LLMs at inference time. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 22137–22176. PMLR, 23–29 Jul 2023.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 2024b.

Mathematical Association of America. American invitational mathematics examination (AIME) 2024. https://huggingface.co/datasets/AI-MO/aimo-validation-aime, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, volume 37, pp. 30811–30849. Curran Associates, Inc., 2024.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling Transformer inference. In *Proceedings of Machine Learning and Systems*, volume 5, pp. 606–624, 2023.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57355–57373. PMLR, 2025.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911. PMLR, 2024.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Agüera y Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. LaMDA: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

Kiran Vodrahalli, Santiago Ontanon, Nilesh Tripuraneni, Kelvin Xu, Sanil Jain, Rakesh Shivanna, Jeffrey Hui, Nishanth Dikkala, Mehran Kazemi, Bahare Fatemi, Rohan Anil, Ethan Dyer, Siamak Shakeri, Roopali Vij, Harsh Mehta, Vinay Ramasesh, Quoc Le, Ed Chi, Yifeng Lu, Orhan Firat, Angeliki Lazaridou, Jean-Baptiste Lespiau, Nithya Attaluri, and Kate Olszewska. Michelangelo: Long context evaluations beyond haystacks via latent structure queries, 2024.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Wei Wu, Zhuoshi Pan, Kun Fu, Chao Wang, Liyi Chen, Yunchu Bai, Tianfu Wang, Zheng Wang, and Hui Xiong. TokenSelect: Efficient long-context inference and length extrapolation for LLMs via dynamic token-level KV cache selection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21264–21281, Suzhou, China, 2025b. Association for Computational Linguistics.

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

Xintong Yang, Hao Gu, Binxing Xu, Lujun Li, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Yike Guo, and Sirui Han. IndexMem: Learned KV-cache eviction with latent memory for long-context LLM inference. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2026.

Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. Wordcraft: Story writing with large language models. In *27th International Conference on Intelligent User Interfaces*, pp. 841–852, 2022.

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B. Hashimoto. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024.

A Training Details

We keep the TokenButler predictor architecture fixed across backbones: we place a *producer* every $G = 4$ transformer layers (i.e., `producer_frequency=4`), predict low-dimensional importance queries with interaction dimension $d' = 16$, and use a two-layer MLP with hidden size 512; the base LLM is frozen and we train only TokenButler parameters with `lr` $= 10^{-3}$. Unless otherwise noted, predictor training uses training sequence length of 1024 tokens: the training data is a concatenation of (i) general web text from C4 (RealNewsLike, 90k examples) (Dodge et al., 2021), (ii) educational web text from FineWeb-Edu (`sample-10BT`, 90k examples) (Penedo et al., 2024), (iii) code from CodeParrot-Clean (90k streamed examples) (CodeParrot, 2022), and (iv) long-context QA-style sequences from BABILong (Kuratov et al., 2024) (contexts $\{2\text{k}, 4\text{k}, 8\text{k}, 16\text{k}\}$ across tasks `qa1-qa10`), where each BABILong example is flattened into a single text field (`input + question + target`) and then tokenized into fixed-length training windows (padding with `eos` if needed). To make distillation efficient, we compute the cross-entropy loss between teacher and student *masked causal attention distributions* on a subsampled set of query rows: when row-subsampling is enabled we draw most loss rows from the late-context "tail" (by default, the final fraction of positions) and always include the final token, reducing auxiliary attention-loss cost from $\mathcal{O}(L^2)$ to $\mathcal{O}(RL)$ with $R \ll L$. We train separate TokenButler predictors for each model discussed in this paper.

A.1 Training cost

Training TokenButler does *not* fine-tune the base LLM. On a single A6000 GPU, the end-to-end predictor training time (including data loading and optimization) is shown in Table 6.

Table 6: TokenButler predictor training time on a single A6000 GPU.

Model	TokenButler params		Training time	Param. Percentage
	(M)	(exact)	(hh:mm)	(%)
Llama-3.2-1B	8.93	8,929,280	04:43	0.893
Llama-3.2-3B	19.77	19,769,344	06:45	0.659
Llama-3.1-8B-Instruct	29.43	29,425,664	09:08	0.368
DeepSeek-R1-Distill-Llama-8B	29.43	29,425,664	09:33	0.368
Qwen2.5-7B-Instruct-1M	20.92	20,923,392	08:37	0.299

B Effect of Predictor Attachment Depth

We ablate the layer at which TokenButler consumes hidden states by training five predictors (each ≈54.6M parameters) on `Llama-3.2-3B`, attached at layers $\{0, 4, 8, 16, 24\}$. For target layers 25–27, we evaluate recall across a budget sweep (Recall@$k\%$); the resulting curves are shown in Fig. 5. Plotted markers correspond to the measured Recall@$k\%$ values (e.g., 10/30/50), and lines provide a simple linear interpolation. We find that later attachment increases recall across budgets (predictor @24 is best), but layers $< k$ must then become dense, reducing the achievable sparsity budget. In practice, there is a tradeoff such that we (i) attach at a moderate depth to balance recall and sparsity, or (ii) when memory allows, use multiple lightweight predictors (e.g., every 4 layers) to approach the accuracy of attaching at later layers, to retain more sparsity.

C Predictor Scaling Study

We study how TokenButler's parameter count affects token-importance recovery. All predictors are trained with the same protocol on `Llama-3.2-3B` and evaluated on WikiText2 (Merity et al., 2017). Table 7 reports Recall@50%, i.e., the fraction of ground-truth high-importance tokens recovered when keeping the predictor's top-50% predictions (averaged over heads and sparse layers).

We observe a smooth scaling trend: increasing the predictor size from 3.48M to 287M improves Recall@50% by +13.6 points (67.38% → 81.02%), providing a convenient accuracy/overhead trade-off for different deployment budgets.

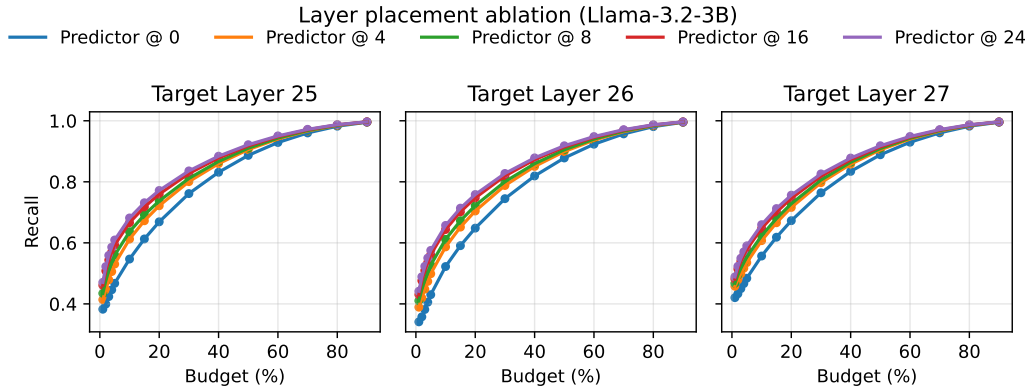


Figure 5: **Layer placement ablation (Llama-3.2-3B).** Recall vs. budget for target layers 25–27. Each curve corresponds to a predictor attached at layers {0, 4, 8, 16, 24}. Markers denote the measured Recall@$k\%$ points (e.g., 10/30/50). Later attachment (e.g., predictor @24) consistently yields higher recall across budgets, but leaves fewer layers for sparse execution.

Table 7: **Predictor size scaling (Llama-3.2-3B).** Larger predictors yield higher Recall@50%.

Predictor size (M params)	3.48	5.06	12.40	39.66	144.52	287.00
Recall@50% (%)	67.38	70.18	71.90	73.90	79.70	81.02

D Prediction Interval Ablation

Table 8 reports per-task RULER accuracy on `Llama-3.1-8B-Instruct` at 64K context as the prediction interval i grows from 1 to 16 with neighbor fetching enabled (Section 3.4); the averages are discussed in Section 4.3.

E Sink and Local Window Size Sensitivity

We verified that accuracy is insensitive to the sink size S and the local window size W around our defaults (S=128, W=256). On a RULER subset (FWE, QA-2, VT) with `Llama-3.1-8B-Instruct`, sweeping $S \in \{64, 128, 256\}$ moves the three-task average by at most 0.2 points (84.51-84.70), and sweeping $W \in \{128, 256, 512\}$ likewise stays within 0.2 points, with both sweeps attaining their highest value at our chosen defaults. For neighbor fetching we use one neighbor per selected token ($2B$ buffer slots); Table 8 shows this keeps accuracy within 1.1% of the per-step baseline at i=16.

F Accuracy Comparison to TokenSelect

TokenSelect (Wu et al., 2025b) is the closest query-aware, per-token, full-cache baseline: it selects tokens by full-dimensional query-key dot products rather than a learned low-dimensional predictor. Table 9 compares RULER accuracy on `Llama-3.1-8B-Instruct` at context lengths from 4K to 32K. TokenButler is more accurate at every length, and the gap widens with context (15.1 points on average). The corresponding on-GPU throughput comparison is in Appendix J.

G LongMemEval Results

Table 10 reports accuracy per question type on LongMemEval (Wu et al., 2025a) (cleaned release) for Dense attention and TokenButler with an 8K token budget on `Llama-3.1-8B-Instruct`. TokenButler tracks Dense across question types, including the multi-session and temporal-reasoning categories that require recalling information introduced long before it is needed.

Table 8: RULER evaluation on Llama-3.1-8B-Instruct at 64K context length with prediction interval and neighbor fetching. $i{=}N$ runs the predictor every N steps with neighbor fetching enabled (sparse budget $8K$). At $16\times$ predictor amortization accuracy is within 1.1% of the per-step baseline ($i{=}1$); the largest drop over the sweep is 1.4% at $i{=}8$.

Config	Pred. Interval	N-S1	N-S2	N-MK1	N-MK2	N-MQ	N-MV	QA-1	QA-2	VT	FWE	Avg.
$i{=}1$	every step	100.00	100.00	100.00	96.88	98.96	94.27	83.33	57.29	91.46	77.08	89.93
$i{=}2$	every 2	100.00	100.00	100.00	94.79	99.22	96.88	81.25	59.38	92.50	73.96	89.80
$i{=}4$	every 4	100.00	100.00	100.00	94.79	97.14	93.75	81.25	57.29	92.71	72.57	88.95
$i{=}8$	every 8	100.00	100.00	100.00	94.79	89.32	94.53	81.25	57.29	92.29	75.69	88.52
$i{=}16$	every 16	100.00	100.00	100.00	94.79	92.19	93.49	82.29	56.25	93.33	75.69	88.80

Table 9: RULER accuracy (%) of TokenSelect and TokenButler on `Llama-3.1-8B-Instruct` at different context lengths.

Method	4K	8K	16K	32K	Avg.
TokenSelect	93.73	82.92	71.92	65.38	78.49
TokenButler	**95.91**	**94.21**	**91.13**	**93.12**	**93.60**

H Synthetic Co-reference Benchmark

To rigorously evaluate token sparsity methods under retrieval-intensive scenarios, we developed a synthetic co-reference benchmark utilizing OpenAI's `gpt-4o-mini` model. The benchmark consists of 100 unique fictional location names, 100 paired location introductions and tieback questions, 100 philosophical reflections, 100 culinary descriptions, and 100 short math problems. Each data sample is constructed by randomly selecting one location introduction along with its corresponding tieback question, one location name, one philosophical statement, one culinary description, and one math problem. The resulting sequence is structured such that the location is introduced early in the context, followed by distractor content, and concludes with a prelude statement that prompts the retrieval of the original location name.

This modular generation approach allows for the creation of up to $100^4 = 10^8$ unique sequences by combining different components, ensuring extensive diversity. When a specific number of samples are requested, they are dynamically generated by randomly drawing from the respective pools of location introductions, location names, philosophical statements, culinary descriptions, and math problems. This on-the-fly sampling methodology ensures that each test instance presents a distinct retrieval challenge, effectively simulating real-world conversational dynamics where important tokens may reappear unpredictably after various interleaved topics. By designing the benchmark in this manner, we specifically target the capability of token sparsity methods to accurately retain and retrieve critical tokens between substantial contextual noise, thereby providing a robust assessment of their effectiveness in maintaining model performance on co-referential tasks.

I Timing Breakdown

We analyze the time taken to do different operations in the case of Dense Attention and in case of TokenButler. As shown in Figure 6, The Attention Kernel in TokenButler takes almost a constant time as the context length increases while it is the main growth in latency factor in Dense Attention. On the other hand, the most growing operation in TokenButler with context length is the Importance Score Computation because that one has to go through all existing tokens but at a smaller dimension than the original attention. It can be seen also that some of the constant times like gathering of keys and values and the Attention Kernel consume more time when the Sparse Token Budget (K) is higher.

J Additional Efficiency Evaluation

We provide additional evaluation of the end-to-end performance. We integrate TokenButler with a `Llama-3.2 1B` model and measure the end-to-end decode throughput under different context lengths in Figure 7. The evaluation utilizes TokenSelect (Wu et al., 2025b) code base where we

Table 10: LongMemEval accuracy (%) by question type. TokenButler uses an 8K token budget.

Question type	Dense	TokenButler
single-session-user	40.6	40.6
single-session-assistant	51.8	46.4
single-session-preference	17.3	17.0
multi-session	15.7	15.7
knowledge-update	50.0	44.4
temporal-reasoning	16.5	18.1
abstention	3.3	6.7
overall	27.4	26.6

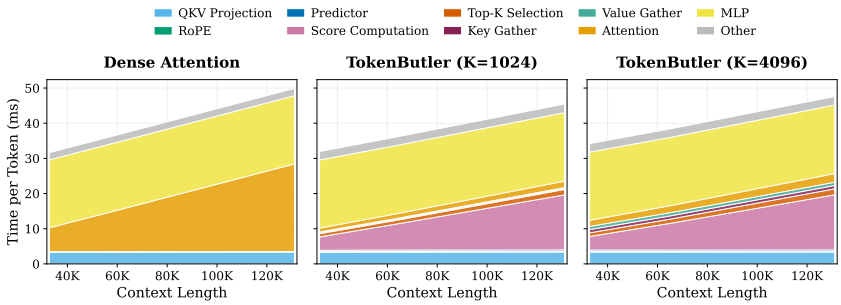


Figure 6: Breakdown of time taken in different operations for `Llama-3.1-8B-Instruct`.

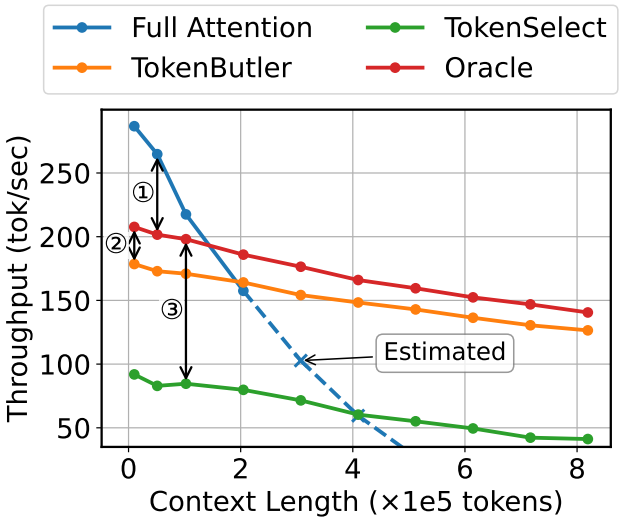


Figure 7: Performance of TokenButler vs. Dense Attention and TokenSelect at 1024 token budget on an H100 GPU. ①: Sparse Attention Overhead. ②: TokenButler Overhead. ③: TokenSelect Overhead.

replace their method by a different version of TokenButler that predicts the importance per token per layer removing the head dimension from the predictions to match the token retrieval method of the system. Full attention throughput drops as the context length increases, eventually giving an error. Token sparsity methods like TokenButler are needed to counter that. TokenButler throughput is close to the oracle performance and TokenButler is more efficient than TokenSelect (up to 3.1× higher throughput) as our predictor is very lightweight and does not need to do the dot product with the full original embedding dimension E between Q and K.