TAPKV: ACCELERATING LONG-HORIZON SPECULATIVE DECODING WITH TARGET-AWARE DRAFT KV CACHES OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The speedup of speculative decoding hinges on generating draft proposals at low cost while maintaining high draft–target agreement. In long-horizon speculative decoding, where generation spans long decoding sequences, however, the growing draft KV cache incurs substantial memory traffic and increases drafting latency. KV-cache pruning provides a natural opportunity to reduce this overhead and accelerate drafting. However, existing attention-based importance metrics primarily identify influential KV entries for draft model performance, without explicitly capturing how their removal affects draft–target agreement and, ultimately, the speculative acceptance rate. We introduce *TapKV*, a draft-cache pruning method. We derive an *Attention-output Taylor* (AOT) score that combines the deletion-induced attention-output change with the loss gradient, capturing both value removal and attention renormalization. AOT guides cache-entry selection and global budget allocation across layers, while a paged-attention kernel efficiently supports the resulting unequal cache lengths. Experiments across diverse workloads show that TapKV achieves higher acceptance lengths and speedups than existing baselines, delivering an average speedup of $2.83\times$–$2.92\times$.

1 INTRODUCTION

Speculative decoding (SD) accelerates large language model (LLM) inference by using a lightweight draft model to propose tokens, which a larger target model then verifies in parallel. (Li et al., 2024b;c; Cai et al., 2024a; Ankner et al., 2024; Xia et al., 2023; Zhang et al., 2023a; Chen et al., 2024; Sun et al., 2024; Hu et al., 2025a;b). Meanwhile, many emerging generative workloads require LLMs to produce increasingly long output sequences. Examples include complex reasoning Guo et al. (2025), code generation Deng et al. (2025), agentic tasks Zhou et al. (2024a), and long-form content generation Wu et al. (2025), where a single request may involve thousands or even tens of thousands of generated tokens. Such workloads create a *long-horizon decoding* regime in which inference efficiency must be sustained throughout an extended generation process, rather than only over a short output sequence. However, in *long-horizon speculative decoding* with long generated sequences, the growing draft KV cache increases memory traffic and computational cost of attention, causing drafting latency to scale with generation length and eventually dominate total decoding time. To demonstrate this effect, we measure the latency breakdown of speculative decoding on the AIME dataset (Mathematical Association of America, 2025), using Qwen3-0.6B as the draft model and Qwen3-8B as the target model. As shown in Figure 1(a), as the decoding length increases, an increasingly large fraction of the SD latency is spent on drafting due to the growing draft KV cache. Consequently, the speedup of SD gradually diminishes and eventually approaches that of conventional autoregressive decoding.

This motivates periodic KV-cache pruning throughout draft generation. Conventional KV-cache pruning methods primarily estimate the importance of each KV entry from its attention statistics using key or value properties or reconstruction errors relative to full-cache execution (Xiao et al., 2024; Li et al., 2024a; Park et al., 2026; Cai et al., 2026; Chang et al., 2026; Kim et al., 2026b). While these methods have shown strong performance in preserving model accuracy under aggressive cache reduction, their objectives are **not directly aligned** with the goal of the draft model: closely approximating the target model's behavior to maximize draft–target agreement and acceptance length.

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.

Beyond designing a better KV-cache pruning criterion, efficient draft KV-cache pruning also requires deciding how the limited cache budget should be allocated across layers. While existing methods commonly assign a uniform budget to each layer (Cai et al., 2024b; Feng et al., 2026), an adaptive allocation can better account for the varying importance of KV entries across layers. However, realizing such non-uniform allocation also requires efficient system and kernel support, as conventional attention kernels often rely on uniform cache sizes and may lose the benefits of adaptive allocation through padding and redundant computation.

To address these limitations, we present *TapKV*, a target-aware framework for efficient draft KV-cache pruning in SD. TapKV formulates pruning as a system-level trade-off between target acceptance and drafting and pruning costs. To improve draft–target agreement under this tradeoff, TapKV uses a target-guided loss with first-order Taylor approximation to avoid heavy computation and estimates the impact of each KV deletion on this objective. The resulting Attention-output Taylor (AOT) score provides a common importance scale, enabling global allocation of the KV-cache budget across layers. TapKV periodically updates the scores and prunes KV cache to amortize pruning overhead and employs a layer-wise paged-attention backend to efficiently support non-uniform cache allocation. Our main contributions are summarized as follows:

- We derive a target-guided AOT score that combines the attention-output change caused by KV removal with the gradient of an agreement loss, directly estimating the impact of each KV-cache removal on SD efficiency.
- We develop a KV-cache budget allocation algorithm that globally ranks cache entries by target-guided importance and adaptively distributes the retention budget across layers.
- We develop a specialized paged-attention kernel that efficiently supports layer-specific KV-cache lengths, together with periodic score and cache updates to amortize the runtime overhead of target-aware pruning.
- Experiments show that TapKV improves throughput and preserves more accepted tokens than other baseline pruning algorithms, achieving $2.83\times$–$2.92\times$ overall SD speedup.

2 Related Work

2.1 Speculative Decoding

Speculative decoding reduces autoregressive inference latency by using a lightweight draft model to propose tokens and a larger target model for parallel verification (Leviathan et al., 2023; Chen et al., 2023). After the prefill stage, the draft model generates candidate tokens that the target model verifies in a single forward pass. Upon rejection, the accepted prefix and a corrected token, known as the *bonus token*, are committed to initiate the next drafting round. Longer accepted sequences therefore reduce the number of expensive target-model forward passes. However, these savings translate into end-to-end speedup only when the drafting process remains fast. Prior work has explored cheaper drafting mechanisms, including model-free drafting based on references or n-gram tables (Yang et al., 2023; He et al., 2023; Ou et al., 2024; Stewart et al., 2024), self-speculation through early exiting or layer skipping (Zhang et al., 2023a; Xia et al., 2024; Elhoushi et al., 2024; Liu et al., 2024a), and lightweight trainable drafters attached to the target model (Li et al., 2024b; Gao et al., 2024). A complementary line of work improves the efficiency of long-context drafting. TriForce introduces a retrieved sparse-cache intermediate model for hierarchical

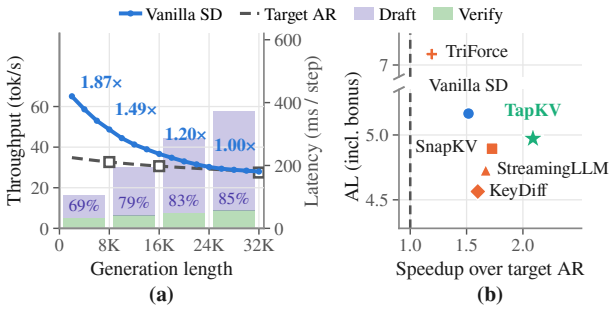

Figure 1: SD performance with Qwen3-0.6B (draft) and Qwen3-8B (target) on AIME 2025 using an A6000 GPU. In (a), bars show the latency breakdown across generation lengths, while lines show vanilla SD and AR throughput and SD speedup. (b) compares acceptance length (AL) and speedup of TapKV and baselines over AR decoding.

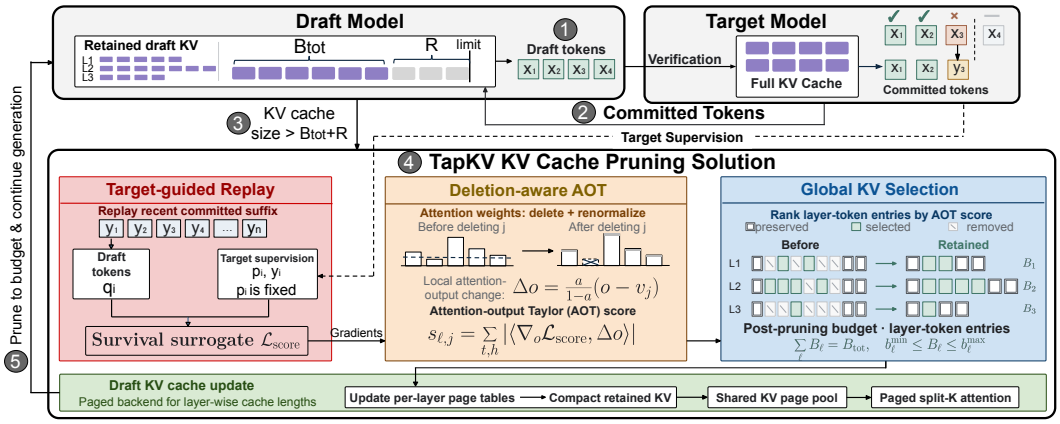

Figure 2: TapKV execution workflow during generation. All step numbers are shown in gray circles.

speculation (Sun et al., 2024), while MagicDec adapts the sparse draft cache according to context length and batch size (Sadhukhan et al., 2025). However, these approaches primarily target settings with long input contexts, rather than long-horizon decoding where the target model generates extended reasoning sequences.

2.2 KV Cache Importance Estimation

Multiple KV-cache pruning algorithms have been explored to improve LLM inference efficiency while preserving model accuracy. Their objective is to accurately estimate which cached entries will remain important for future decoding and retain them under a limited cache budget. StreamingLLM retains attention sinks and recent tokens (Xiao et al., 2024), while H_2O and SnapKV use accumulated or local attention statistics (Zhang et al., 2023b; Li et al., 2024a). KeyDiff exploits key similarity, R-KV combines attention importance with key redundancy, and Value-Aware Stochastic KV Cache Eviction incorporates value magnitude and stochastic selection (Park et al., 2026; Cai et al., 2026; Chang et al., 2026). Expected Attention estimates attention importance under a distribution of future queries (Devoto et al., 2025). KVzip measures context reconstruction error, while KVzap and Fast KVzip predict retention decisions from hidden states or reconstruction-based supervision (Kim et al., 2026b; Jegou & Jeblick, 2026; Kim et al., 2026a). Despite their different designs, these methods share two limitations when applied to draft KV-cache pruning. First, their importance scores are designed to preserve the pruned model's own accuracy rather than its agreement with the target model, creating an objective mismatch. Second, they do not explicitly capture the impact of KV-entry deletion, including value removal and attention renormalization, creating an operation mismatch. TapKV addresses both limitations via target-aware importance estimation.

Most KV-cache pruning methods use fixed per-layer or per-head budgets (Xiao et al., 2024; Zhang et al., 2023b; Li et al., 2024a), while recent methods adapt budgets using predefined layer patterns or attention importance (Cai et al., 2024b; Feng et al., 2026). However, these criteria are independent of the target model and do not directly optimize draft–target agreement. TapKV instead globally ranks KV entries using a common target-guided importance score, allowing layer-wise budgets to emerge from their relative importance.

3 Methodology

We first use a simple analytical model to characterize how draft KV-cache pruning affects the overall efficiency of speculative decoding. In each speculation round, the draft model proposes a window of γ tokens, which are verified in parallel by the target model. Let $\alpha \in [0, \gamma]$ denote the average number of accepted tokens among the γ proposals, where γ is the window size. Since verification additionally commits one token generated by the target model, each round advances the sequence by an average of $\alpha + 1$ tokens. Let T_{verify} denote the average verification latency, T_{draft} the per-token drafting latency with a trimmed KV cache, and T_{prune} the amortized pruning operation overhead

per round. The resulting speculative decoding throughput β_{sd} can then be approximated as $\beta_{sd} \approx (\alpha + 1)/(T_{\text{verify}} + \gamma T_{\text{draft}} + T_{\text{prune}})$. According to this equation, an effective draft KV-cache pruning algorithm should satisfy three key criteria. First, it should achieve a high pruning ratio to substantially reduce T_{draft}. Second, it should retain the KV entries most critical to draft–target agreement to maximize α. Finally, the pruning procedure itself should be lightweight, with minimal T_{prune} overhead. In the following, we present TapKV design to jointly address these three objectives.

3.1 TapKV Execution Workflow

TapKV modifies only the KV-cache size of the draft model, while the target model always performs verification using its full KV cache. TapKV employs a *continual pruning process* to bound the draft KV-cache size throughout speculative decoding. Figure 2 highlights the TapKV KV workflow. Let B^l_{tot} denote the target retention budget of layer l for the draft KV cache. During drafting, the draft model continuously generates new tokens (Step ①), which are then verified by the target model. The verified tokens are subsequently appended to the draft KV cache, further increasing its size (Step ②). Once the draft KV-cache size reaches $B^l_{\text{upper}} = B^l_{\text{tot}} + R$ (Step ③), where B^l_{upper} denotes the maximum allowed KV-cache size, pruning is triggered to remove low-importance KV entries using the mechanism described in Section 3.2 (Step ④). This reduces the cache size back to B^l_{tot} and substantially accelerates subsequent drafting (Step ⑤). During initialization, if the draft KV-cache size already exceeds B^l_{upper}, pruning is triggered immediately. Otherwise, the first pruning event occurs once the cache grows to $B^l_{\text{upper}} = B^l_{\text{tot}} + R$. By keeping the draft model's KV-cache size bounded between B^l_{tot} and B^l_{upper}, TapKV ensures efficient and scalable drafting throughout long-horizon generation. Meanwhile, pruning is performed only after approximately R new tokens have been verified. Since each speculation round commits $\alpha + 1$ tokens on average, a pruning event occurs approximately once every $R/(\alpha + 1)$ rounds, effectively amortizing the pruning overhead T_{prune} across multiple speculation rounds.

Upon reaching the pruning threshold B^l_{upper}, TapKV evaluates the importance of each KV-cache entry to determine which entries should be retained. As discussed in Section 1, although various KV-cache pruning methods have been proposed, their importance metrics, such as attention-based scores (Michel et al., 2019), are not directly aligned with the primary objective of speculative decoding: enabling the draft model to closely approximate the target model's token distribution and thereby maintain high draft–target agreement. To address this mismatch, TapKV introduces a *Attention-output Taylor (AOT) Scoring* to evaluate the importance of the KV entries. Specifically, TapKV replays the most recent W verified tokens through the draft model and obtains the probabilities q_i assigned by the draft model to the corresponding verified tokens y_i, where $1 \le i \le W$. Let p_i denote the probabilities assigned to the same tokens by the target model, and further define $\mathbf{p} = \{p_i\}_{1 \le i \le W}$ and $\mathbf{q} = \{q_i\}_{1 \le i \le W}$. To evaluate the importance of each KV-cache entry $(k_{t'}, v_{t'})$ associated with token t', we measure how its removal affects the loss $\mathcal{L}(\mathbf{p}, \mathbf{q})$, which can be defined using KL divergence. We estimate this impact using the AOT scoring method, described in detail in Section 3.2. Finally, TapKV can also be extended to tree-based drafting, as it only constrains the draft KV cache while leaving the proposal structure, verification rule unchanged.

3.2 Attention-Output Taylor Scoring for KV-Entry Selection

In this section, we evaluate the impact of removing each KV entry on the model output and use this impact to quantify its importance for KV-cache pruning. Specifically, TapKV derives the local change in the attention output induced by each KV-entry deletion and applies a first-order Taylor approximation to estimate its effect on the target-aware loss.

The KV entries associated with each token $t \in \mathcal{T}$ are evaluated at the granularity of individual layers and attention heads. For simplicity of notation, we omit the layer index in the following derivation. Let $\mathcal{T}$ denotes the set of W replayed tokens used for scoring. Let $\mathbf{q}^h_t$ denote the query vector of token t at attention head h. For another token t' whose KV entry is retained in the current cache, its corresponding attention weight $a^h_{t,t'}$ with respect to query token t at head h is given by: $a^h_{t,t'} = \frac{\exp(\mathbf{q}_t^\top \mathbf{k}_{t'}/\sqrt{d_{\text{head}}})}{\sum_{t'} \exp(\mathbf{q}_t^\top \mathbf{k}_{t'}/\sqrt{d_{\text{head}}})}$, where d_{head} denotes the query/key head dimension. The attention output of head h for token t is then computed as $\mathbf{o}^h_t = \sum_{t'} a^h_{t,t'} \mathbf{v}^h_{t'}$. A natural way to quantify the importance of the KV entry $(\mathbf{k}^h_{t'}, \mathbf{v}^h_{t'})$ is to measure how its removal changes the corresponding attention output

$\mathbf{o}_t^h$. Specifically, the deletion-induced change in the attention output can be expressed as:

$$\Delta \mathbf{o}_{t,t'}^h = \frac{\mathbf{o}_t^h - a_{t,t'}^h \mathbf{v}_{t'}^h}{1 - a_{t,t'}^h} - \mathbf{o}_t^h \tag{1}$$

where $\Delta \mathbf{o}_{t,t'}^h$ denotes the change in the attention output $\mathbf{o}_t^h$ induced by removing the KV entry $(\mathbf{k}_{t'}^h, \mathbf{v}_{t'}^h)$. In Equation 1, the term $a_{t,t'}^h \mathbf{v}_{t'}^h$ captures the direct contribution of the removed value vector to the original attention output value, while the factor $1 - a_{t,t'}^h$ accounts for the renormalization of the remaining attention weights after the KV entry of token t' is removed. Given the above analysis, we then evaluate the impact of the KV entry removal on the loss $\mathcal{L}(\mathbf{p}, \mathbf{q})$. Specifically, this can be estimated using first-order Taylor expansion:

$$\Delta \mathcal{L}_{t'} \approx \sum_h \sum_{t \in \mathcal{T}} \left\langle \nabla_{\mathbf{o}_t^h} \mathcal{L}(\mathbf{p}, \mathbf{q}), \Delta \mathbf{o}_{t,t'}^h \right\rangle \tag{2}$$

where $\Delta \mathcal{L}_{t'}$ denotes the change in the target-aware loss induced by removing all KV entries associated with token t' across all attention heads of one layer. Here, $\langle \cdot, \cdot \rangle$ denotes the dot product. To quantify the overall sensitivity of the loss to deleting token t', TapKV aggregates the magnitudes of the contributions across replayed positions and attention heads, avoiding cancellation between positive and negative effects. This yields the *Attention-output Taylor* (AOT) score:

$$s_{t',l}^{\mathrm{AOT}} = \sum_h \sum_{t \in \mathcal{T}} \left| \left\langle \nabla_{\mathbf{o}_t^h} \mathcal{L}(\mathbf{p}, \mathbf{q}), \Delta \mathbf{o}_{t,t'}^h \right\rangle \right| \tag{3}$$

KV entries with larger AOT scores are assigned higher retention priority. In addition, motivated by the previous study Xiao et al. (2024), TapKV always preserves the initial attention-sink tokens, a designated recent window, and the complete committed suffix used for replay. These protected positions count toward the overall cache budget, while AOT scores are used to rank the remaining candidate entries. Within each layer, the retention decision for each token is shared across all KV heads.

3.3 Global KV-Cache Budget Allocation

Uniform per-layer budgets can retain lower-scoring KV entries in one layer while discarding higher-scoring entries in another. To avoid this inefficiency, TapKV allocates the cache budget globally based on AOT scores. AOT provides a common first-order sensitivity measure across layers. Because each score is derived from the gradient of the same target-aware loss, AOT scores across different layers are expressed on a common loss-based scale and can therefore be directly compared for global ranking. Let B_{tot} denote the average per-layer retention budget, so that the total number of retained positions after pruning, counted separately within each layer and summed across all N_{layer} layers, is $\sum_l B_{\mathrm{tot}}^l = N_{\mathrm{layer}} B_{\mathrm{tot}}$, where B_{tot} is the average KV cache budget across layers. For each layer l, we constrain the number of retained KV entries to lie between $b_l^{\min}$ and $b_l^{\max}$.

TapKV performs global budget allocation in two stages. First, all preserved KV entries, including the attention-sink and recent-window tokens, are retained. For each layer with fewer than $b_l^{\min}$ retained KV entries, TapKV selects the highest-scoring KV entries from that layer until the lower bound is satisfied. It then allocates the remaining budget globally by considering all unselected candidates across layers in descending order of their AOT scores. A candidate is retained if its layer has not reached $b_l^{\max}$. Otherwise, it is skipped. This process continues until a total of $N_{\mathrm{layer}} B_{\mathrm{tot}}$ KV entries have been retained across all layers. Finally, the retained KV entries within each layer are restored to their original sequence order. Given fixed AOT scores and feasible constraints, this procedure maximizes the sum of AOT scores of the selected unprotected KV entries subject to the global budget and per-layer retention constraints. Appendix A.3 formalizes the optimization problem, describes the selection of the per-layer bounds, and provides a proof of this result.

3.4 TapKV System Implementation

As described in Section 3.3, TapKV further improves efficiency by adaptively allocating different KV-cache budgets across layers. However, CUDA Graph execution requires fixed-shape buffers at fixed memory addresses to keep the draft step graph-capturable NVIDIA (2025); Kwon et al. (2023).

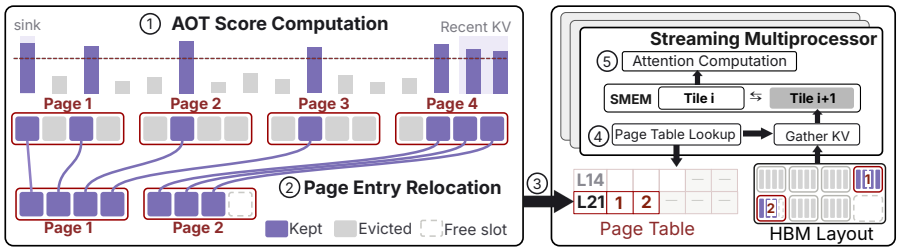

Figure 3: System implementation of TapKV. The step numbers are shown in circles.

A straightforward static allocation would therefore need to provision every layer for the maximum cache budget and mask unused entries during computation. This padding eliminates much of the memory and computational savings enabled by adaptive budget allocation.

To address this issue, we store the draft KV cache in fixed-size pages, each containing P tokens, with a separate page table for each layer (Figure 3, left). This organization changes only the physical layout of the cache, while preserving the logical semantics described in Section 3. Specifically, for layer ℓ, the retained set occupies the first $\lceil B^{\ell}_{tot}/P \rceil$ pages, where P is the page entry size. The recent buffer is organized similarly but grows dynamically as decoding proceeds: newly committed tokens are appended to the next available slot, and a new page is allocated from the page pool whenever the current tail page becomes full. Consequently, each layer occupies only the pages required by its allocated budget, avoiding padding to the maximum layer budget and eliminating computation on unused cache entries. Meanwhile, because the page pool and page tables remain at fixed memory addresses, the draft step remains CUDA-graph capturable.

To better preserve acceptance length under a fixed cache budget, we perform token-level pruning directly over the paged KV cache, allowing individual tokens to be retained or removed within each page. For each layer ℓ, once its KV-cache size reaches B^{ℓ}_{upper}, we first compute an importance score for each KV entry and evict low-scoring entries from their respective pages (Step ①). The surviving entries are then compacted in temporal order into the leading pages (e.g., Pages 1 and 2), leaving the remaining pages completely empty (Step ②). These empty pages are returned to the shared page pool and can be reused by other layers. Finally, the page table is updated in place to reflect the new physical layout, reducing the number of active pages from four (Pages 1–4) to two (Pages 1–2).

The paged KV-cache layout and SD workflow require an attention kernel that (i) accesses KV entries directly through the page tables and (ii) efficiently supports decoding workloads, where each step contains only one or a few query tokens and the attention computation is dominated by GEMV or small-GEMM operations. Given this workload, following prior work Dao et al. (2022); Kwon et al. (2023), we adapt Flash-Decoding to operate directly over the paged KV cache (Figure 3, right). The workflow proceeds as follows: After each pruning event, the page tables are updated to reflect the current physical locations of the retained KV entries (Step ③). To expose sufficient parallelism for the short-query decoding workload, attention is partitioned along the sequence dimension in addition to the batch and head dimensions, such that each thread block processes a subset of the KV pages. Each thread block first accesses the corresponding page-table entries (e.g., Pages 1 and 2 for Layer 21) and loads the associated K/V tiles from HBM into shared memory (SMEM). These loads are double-buffered: while the current tile is being processed, the next tile is prefetched, helping hide the latency of non-contiguous memory accesses (Step ④). Each block then computes a partial attention result using an online softmax, after which the partial results are combined using the corresponding online-softmax rescaling. This design enables direct attention over the paged KV cache while exposing substantial sequence-level parallelism for efficient decoding (Step ⑤). The same kernel also supports tree-based drafting: the query becomes a Q-row block rather than a single row, while the tree structure is captured solely by an additive mask that is preallocated and indexed by logical KV positions. Algorithm 1 in Appendix A.1 summarizes the TapKV algorithm details.

4 Evaluations

We evaluate draft KV-cache pruning methods across four draft–target model pairs: Qwen3-0.6B+Qwen3-8B (Yang et al., 2025a), Qwen3-1.7B+Qwen3-14B, DS-R1-Distill-1.5B+DS-R1-Distill-14B (Guo et al., 2025), and Qwen3-0.6B+Phi-4-14B (Abdin et al., 2024). Since Qwen3-

Table 1: Evaluation of TapKV on Greedy Speculative Decoding.

		vanilla SD		TriForce		SnapKV		StreamingLLM		KeyDiff		**TapKV**	
Draft+Target	Benchmark	AL	SP	AL	SP	AL	SP	AL	SP	AL	SP	AL	SP
Qwen3 0.6B+8B	AIME 2025	5.17	1.52	7.14	1.19	4.89	1.73	4.73	1.67	4.51	1.61	5.05	**2.09**
	LiveCodeBench	4.65	1.37	7.17	1.16	4.36	1.55	4.23	1.51	4.04	1.45	4.36	**1.90**
	GPQA	4.62	1.40	7.24	1.11	4.25	1.51	4.26	1.51	3.96	1.42	4.26	**1.85**
	LongProc	6.83	1.01	6.99	1.46	4.49	1.73	4.64	1.79	4.18	1.64	4.73	**2.15**
	Avg	5.32	1.32	7.13	1.23	4.50	1.63	4.46	1.62	4.17	1.53	**4.60**	**2.00**
Qwen3 1.7B+14B	AIME 2025	5.29	1.72	7.14	1.25	5.00	1.89	4.80	1.82	4.50	1.71	5.07	**2.17**
	LiveCodeBench	4.83	1.58	7.17	1.19	4.50	1.70	4.35	1.66	4.09	1.56	4.52	**1.93**
	GPQA	4.50	1.52	7.26	1.17	4.05	1.54	4.10	1.56	3.82	1.46	4.26	**1.84**
	LongProc	7.64	1.41	7.04	1.54	5.84	2.28	5.81	2.27	5.42	2.12	5.89	**2.57**
	Avg	5.56	1.56	7.15	1.29	4.84	1.85	4.76	1.83	4.46	1.71	**4.93**	**2.13**
DS-R1-Distill 1.5B+14B	AIME 2025	5.79	1.98	7.20	1.36	5.55	2.24	5.50	2.25	5.36	2.20	5.56	**2.53**
	LiveCodeBench	5.47	1.85	7.26	1.30	5.19	2.13	5.38	2.21	4.92	2.03	5.32	**2.47**
	GPQA	4.60	1.64	7.34	1.24	4.38	1.80	4.31	1.78	4.27	1.76	4.31	**2.02**
	LongProc	6.03	1.23	7.07	1.57	5.10	2.20	4.51	1.97	4.64	2.02	5.32	**2.57**
	Avg	5.47	1.68	7.22	1.37	5.05	2.09	4.93	2.05	4.80	2.00	**5.13**	**2.40**
Qwen3-0.6B +Phi-4-14B	AIME 2025	3.89	1.37	6.33	0.78	3.76	1.43	3.73	1.42	3.52	1.33	3.78	**2.01**
	LiveCodeBench	3.91	1.42	6.35	0.78	3.77	1.48	3.98	1.57	3.64	1.43	3.94	**1.78**
	GPQA	4.76	1.72	6.48	0.79	4.52	1.73	4.54	1.75	4.12	1.58	4.45	**2.35**
	LongProc	7.15	1.47	6.90	0.88	6.13	2.62	6.19	2.65	5.34	2.18	6.33	**3.46**
	Avg	4.93	1.50	6.51	0.81	4.55	1.81	4.61	1.84	4.15	1.63	**4.62**	**2.40**

0.6B and Phi-4-14B use different vocabularies, we adopt the cross-tokenizer method from (Timor et al., 2025). We evaluate on four benchmarks: AIME 2025 (Mathematical Association of America, 2025), LiveCodeBench (Jain et al., 2025), GPQA (Rein et al., 2024), and LongProc (Ye et al., 2025) and a maximum generation length of 8K tokens. All experiments are conducted on an NVIDIA RTX A6000 GPU. We report SD results under both greedy decoding and speculative sampling (Leviathan et al., 2023; Chen et al., 2023) with a temperature of 0.6, and further extend our method to tree-based drafting Miao et al. (2023). For chain drafting, we use a window size $\gamma = 7$. For tree drafting, we follow the EAGLE-2 dynamic tree setup with depth 7, top-k of 8, and 64 verified nodes per step (Li et al., 2024c). We adopt an average retention budget of 1024 and B_{upper} of 2048.

We compare against conventional autoregressive decoding (AR), vanilla SD without KV-cache pruning (vanilla SD), and three draft-side KV-cache pruning methods: SnapKV Li et al. (2024a), StreamingLLM Xiao et al. (2024), and KeyDiff Park et al. (2026). Each method employs a different KV-cache pruning strategy, but all primarily aim to preserve the draft model's own accuracy rather than align its behavior with the target model. We also compare against TriForce Sun et al. (2024), a hierarchical SD method based on self-speculation, where the draft model retrieves and attends to only a subset of KV entries selected based on their importance to SD. All three pruning baselines are re-implemented under the same continuous-pruning framework as TapKV, using a same B_{tot} and B_{upper}. TriForce instead uses its own hierarchical SD framework and method-specific hyperparameters. We use its released configuration, which also performs best in our hyperparameter sweep. For evaluation metrics, we report *Acceptance Length* (AL), defined as the average number of tokens committed per verification step, i.e., accepted draft tokens plus one bonus token, and *Speedup* (SP).

4.1 Main Results and Discussion

Table 1 reports the results of all baselines on four long-generation datasets with deterministic decoding, namely the temperature is set to zero. Although vanilla SD and TriForce achieve high acceptance lengths, this does not translate into high speedup. For vanilla SD, as discussed earlier, the growing KV cache increases drafting overhead and offsets the benefits of SD. In the extreme case, Qwen3-0.6B+8B on LongProc achieves an AL of 6.83 but only a $1.01\times$ speedup, essentially no faster than autoregressive decoding. TriForce is designed for long-prefill, short-decode settings; therefore, in long-generation scenarios, its high acceptance length does not yield a corresponding speedup due to the overhead of self-speculation. Its speedup is highest on LongProc, which has long inputs, and even drops below $1\times$ with the heavier Phi-4 target.

Next, we compare TapKV against the three draft-side pruning baselines, including SnapKV Li et al. (2024a), StreamingLLM Xiao et al. (2024), and KeyDiff Park et al. (2026). Under the same draft-cache budget, TapKV consistently achieves the highest average acceptance length across all draft–target configurations, demonstrating better preservation of draft–target agreement. Combined with an efficient kernel tailored to layer-wise adaptive budgets, TapKV achieves the highest speedup across all settings. Overall, TapKV achieves an average speedup of 2.00–$2.40\times$, compared with 1.53–$2.09\times$ for the pruning baselines.

Table 2: Greedy tree SD with and without TapKV.

		AIME25		LiveCode		GPQA		LongProc	
Draft+Target	**Method**	AL	SP	AL	SP	AL	SP	AL	SP
Qwen3 0.6B+8B	Tree SD	7.25	1.52	7.08	1.57	6.76	1.53	7.69	0.83
	TapKV	7.05	**3.00**	6.82	**2.90**	6.36	**2.69**	6.14	**2.73**
Qwen3 1.7B+14B	Tree SD	7.35	1.92	7.18	1.96	6.75	1.89	7.85	1.10
	TapKV	7.07	**3.03**	6.90	**2.92**	6.46	**2.73**	6.86	**3.01**
DS-R1-Distill 1.5B+14B	Tree SD	7.60	1.99	7.43	1.94	6.72	1.80	6.99	1.08
	TapKV	7.49	**3.09**	7.34	**2.97**	6.41	**2.59**	6.51	**2.81**

TapKV constrains only the draft KV cache and therefore composes naturally with tree-based drafting, with the two techniques providing complementary benefits. Tree drafting increases the AL per verification round, while TapKV reduces the cost of generating each draft proposal. With TapKV, tree drafting achieves mean speedups of $2.83\times$, $2.92\times$, and $2.87\times$ across the three draft–target pairs, corresponding to a 1.20–$1.42\times$ improvement over TapKV with chain drafting. In contrast, without KV-cache pruning, tree drafting increases the mean AL by 1.7–1.9 tokens but improves speedup by at most $0.16\times$, as the growing draft KV cache limits the benefit of higher acceptance.

Table 3: Evaluation of TapKV with sampling-based SD.

		AIME25		LiveCode		GPQA		LongProc		Avg.	
Draft+Target	**Method**	AL	SP	AL	SP	AL	SP	AL	SP	AL	SP
Qwen3 0.6B+8B	vanilla SD	5.11	1.48	4.50	1.38	4.26	1.32	6.90	1.12	5.19	1.33
	SnapKV	4.81	1.69	4.28	1.51	3.91	1.39	4.98	1.93	4.50	1.63
	StreamingLLM	4.64	1.63	4.13	1.46	3.87	1.37	5.14	1.99	4.45	1.61
	KeyDiff	4.51	1.58	4.04	1.43	3.74	1.32	4.31	1.67	4.15	1.50
	TriForce	7.25	0.78	7.11	0.77	7.01	0.76	6.74	0.84	7.03	0.79
	TapKV	4.90	**1.83**	4.28	**1.61**	3.93	**1.49**	4.96	**2.03**	4.52	**1.74**
Qwen3 1.7B+14B	vanilla SD	5.42	1.79	4.92	1.71	4.49	1.59	7.58	1.53	5.60	1.66
	SnapKV	5.07	1.91	4.59	1.74	4.16	1.60	5.64	2.23	4.87	1.87
	StreamingLLM	4.87	1.84	4.44	1.69	4.04	1.55	5.73	2.27	4.77	1.84
	KeyDiff	4.62	1.75	4.27	1.63	3.94	1.52	5.26	2.09	4.52	1.75
	TriForce	7.13	1.23	7.17	1.19	7.24	1.15	7.03	1.49	7.14	1.27
	TapKV	5.14	**2.03**	4.65	**1.84**	4.17	**1.66**	5.74	**2.36**	4.93	**1.97**
DS-R1-Distill 1.5B+14B	vanilla SD	5.73	2.01	5.14	1.81	4.42	1.62	5.46	1.16	5.19	1.65
	SnapKV	5.44	2.21	4.86	1.99	3.98	1.63	4.15	1.76	4.61	1.90
	StreamingLLM	5.34	2.18	4.78	1.97	4.12	1.69	3.71	1.61	4.49	1.86
	KeyDiff	5.27	2.16	4.71	1.93	4.04	1.65	4.00	1.73	4.51	1.87
	TriForce	7.17	1.35	7.23	1.29	7.29	1.22	7.00	1.44	7.17	1.33
	TapKV	5.42	**2.48**	4.82	**2.24**	3.95	**1.84**	4.70	**2.20**	4.72	**2.19**

Table 3 reports TapKV under sampling-based SD at temperature 0.6. TapKV achieves the highest mean AL and speedup across all model pairs (1.74–$2.19\times$ versus 1.50–$1.90\times$ for the pruning baselines). While speculative sampling generally yields lower speedups than greedy decoding due to its stricter distributional agreement requirement (Leviathan et al., 2023), TapKV remains effective under stochastic verification.

4.2 Ablation Studies

4.2.1 Impact of the Pruning Budget

We fix the pruning interval at $R{=}1024$ and vary the average KV-cache budget B_{tot} from 256 to 2048 (Figure 4(a)). Increasing B_{tot} allows the draft model to retain more context, consistently improving AL across all methods. However, larger caches increase memory traffic and attention computation, creating a trade-off between draft–target agreement and drafting cost. This trade-off manifests differently across methods. StreamingLLM achieves its highest speedup at the smallest budget ($B_{\text{tot}}{=}256$, SP=$1.81\times$), where its contiguous sliding-window cache even provides a slightly higher AL than SnapKV. SnapKV similarly performs best at B_{tot} between 256–512, with a SP of $1.80\times$, whereas TapKV reaches its peak at a larger budget of $B_{\text{tot}}{=}1024$, with a SP of $2.09\times$. From the algorithmic perspective, TapKV consistently achieves the highest AL across all budgets, showing that its scoring better preserves draft–target agreement. From the system perspective, the layer-wise paged attention kernel reduces the drafting overhead associated with larger caches, allowing TapKV to benefit from a larger retention budget and ultimately achieve a higher peak speedup.

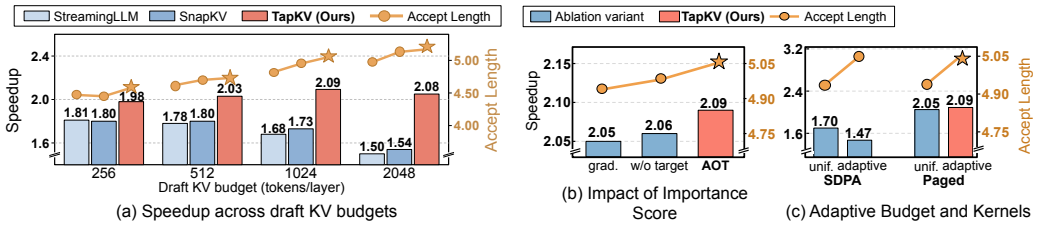

Figure 4: (a) SP across draft KV budgets (Qwen3 0.6B+8B, AIME 2025, γ=7). Bars show SP over AR baseline, the line shows AL. (b) Importance score: AOT versus gradient-only (grad.) and w/o-target variants. (c) Uniform versus adaptive per-layer budget under the SDPA and paged kernels.

4.2.2 Impact of the Adaptive KV Budget Allocation and Kernel Design

In Figure 4(c), we evaluate uniform and adaptive budget allocation under the same average cache size $B_{tot} = 1024, B_{upper} = 2048$, using different attention backends. We compare against uniform KV-budget allocation, where the same retention budget is assigned to every layer. Adaptive allocation improves AL by 0.11 over uniform allocation by distributing the KV-cache budget according to the importance of entries across layers.

However, with a static contiguous KV cache and the SDPA kernel widely used in Hugging Face, this AL improvement does not translate into higher speedup. Because SDPA expects uniform tensor shapes across layers, adaptive allocation requires padding each layer to the largest allocated cache size and masking the unused entries. The resulting redundant memory accesses and attention computation outweigh the benefit of higher AL, reducing speedup from $1.70\times$ to $1.47\times$. In contrast, our paged KV cache stores and processes only the KV entries allocated to each layer, while the specialized attention kernel directly supports these layer-specific cache lengths without padding. This design both reduces draft execution overhead and allows the higher AL from adaptive allocation to translate into actual end-to-end speedup.

4.2.3 Impact of the KV Scoring Method

In this section, we evaluate the contribution of the AOT scoring mechanism (Section 3.2). All experiments use Qwen3-0.6B+Qwen3-8B on AIME 2025, with all other configurations identical to those in Section 4.1. Figure 4(b) ablates one component of the AOT score (Equation 3) at a time. *w/o target* removes the target-guided gradient and instead scores each entry using only the attention-output change caused by its removal, $s^{\text{recon}}_{t',l} = \sum_h \sum_{t \in \mathcal{T}} \|\Delta \mathbf{o}^h_{t,t'}\|_2$. This objective favors preserving the draft model's original attention outputs rather than its agreement with the target, reducing AL by 1.36%. Moreover, taking the norm discards the direction of the output change: a removal that moves the draft prediction closer to the target is penalized similarly to one that moves it farther away. It also captures only the local attention-output change, without accounting for how that change propagates to the final draft–target agreement. In contrast, *grad.* removes the output change caused by KV removal and uses only the loss gradient, $s^{\text{grad}}_{t',l} = \sum_h \left(\|\nabla_{\mathbf{k}^h_{t'}} \mathcal{L}(\mathbf{p}, \mathbf{q})\|_1 + \|\nabla_{\mathbf{v}^h_{t'}} \mathcal{L}(\mathbf{p}, \mathbf{q})\|_1 \right)$. While the gradient captures the sensitivity of the target-guided loss to each KV entry, it does not capture the magnitude of the actual perturbation caused by removing that entry, resulting in a 2.06% reduction in AL. For example, an entry can have a large gradient but receive little attention, such that removing it produces only a small change in the attention output. By combining the target-guided gradient with the actual attention-output change caused by KV removal, AOT measures first-order, target-guided sensitivity to each cache deletion, yielding the highest measured AL and speedup in this ablation.

5 Conclusion

We present TapKV, a target-aware draft KV-cache pruning framework for long-horizon speculative decoding. TapKV uses AOT scoring and global budget allocation to preserve draft–target agreement, while a paged-attention backend efficiently supports layer-specific cache lengths. Across diverse workloads, TapKV improves acceptance and throughput, achieving up to $2.92\times$ speedup.

AI USE STATEMENT

We used large language model tools to assist with parts of the code implementation and with manuscript preparation, including organization, English editing, literature search and reference checking, notation and mathematical exposition, interpretation of existing experimental results, and figure and document preparation. The authors are responsible for the implementation, scientific claims, and final manuscript.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. *arXiv preprint arXiv:2402.05109*, 2024.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024a.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024b.

Zefan Cai, Wen Xiao, Hanshi Sun, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, et al. R-kv: Redundancy-aware kv cache compression for reasoning models. *Advances in neural information processing systems*, 38:60980–61005, 2026.

Ting-Yun Chang, Harvey Yiyun Fu, Deqing Fu, Chenghao Yang, Jesse Thomason, and Robin Jia. Value-aware stochastic kv cache eviction for reasoning models. *arXiv preprint arXiv:2606.03928*, 2026.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*, 2024.

Daewon Choi, Seunghyuk Oh, Saket Dingliwal, Jihoon Tack, Kyuyoung Kim, Woomin Song, Seojin Kim, Insu Han, Jinwoo Shin, Aram Galstyan, Shubham Katiyar, and Sravan Babu Bodapati. Mamba drafters for speculative decoding. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 21404–21418, 2025.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: Kv cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*, 2025.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *Advances in Neural Information Processing Systems*, 38:113152–113188, 2026.

Xiangxiang Gao, Weisheng Xie, Yiwei Xiang, and Feng Ji. Falcon: Faster and parallel inference of large language models through enhanced semi-autoregressive drafting and custom-designed decoding tree. *arXiv preprint arXiv:2412.12639*, 2024.

Raghavv Goel, Mukul Gagrani, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. Direct alignment of draft model for speculative decoding with chat-fine-tuned llms. *arXiv preprint arXiv:2403.00858*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv:2311.08252*, 2023.

Yunhai Hu, Zining Liu, Zhenyuan Dong, Tianfan Peng, Bradley McDanel, and Sai Qian Zhang. Speculative decoding and beyond: An in-depth survey of techniques. *arXiv preprint arXiv:2502.19732*, 2025a.

Yunhai Hu, Tianhua Xia, Zining Liu, Rahul Raman, Xingyu Liu, Bo Bao, Eric Sather, Vithursan Thangarasa, and Sai Qian Zhang. Dream: Drafting with refined target features and entropy-adaptive cross-attention fusion for multimodal speculative decoding. *arXiv preprint arXiv:2505.19201*, 2025b.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025.

Simon Jegou and Maximilian Jeblick. Kvzap: Fast, adaptive, and faithful kv cache pruning. *arXiv preprint arXiv:2601.07891*, 2026.

Jang-Hyun Kim, Dongyoon Han, and Sangdoo Yun. Fast kvzip: Efficient and accurate llm inference with gated kv eviction. *arXiv preprint arXiv:2601.17668*, 2026a.

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. *Advances in Neural Information Processing Systems*, 38:167563–167591, 2026b.

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

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

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. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024c.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*, 2024a.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *International Conference on Machine Learning (ICML)*, 2024b.

Mathematical Association of America. American invitational mathematics examination (aime) 2025. https://maa.org/math-competitions/aime, 2025.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. *arXiv preprint arXiv:2305.09781*, 2023.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

NVIDIA. CUDA C++ programming guide: CUDA graphs. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-graphs, 2025.

Jie Ou, Yueming Chen, and Wenhong Tian. Lossless acceleration of large language model via adaptive n-gram parallel decoding. *arXiv preprint arXiv:2404.08698*, 2024.

Junyoung Park, Dalton Jones, Matthew Morse, Raghavv Goel, Mingu Lee, and Christopher Lott. Keydiff: Key similarity-based kv cache eviction for long-context llm inference in resource-constrained environments. *Advances in Neural Information Processing Systems*, 38:5983–6019, 2026.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *Conference on Language Modeling (COLM)*, 2024.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *International Conference on Learning Representations*, volume 2025, pp. 6835–6850, 2025.

Lawrence Stewart, Matthew Trager, Sujan Kumar Gonugondla, and Stefano Soatto. The n-grammys: Accelerating autoregressive inference with learning-free batched speculation. *arXiv preprint arXiv:2411.03786*, 2024.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024.

Nadav Timor, Jonathan Mamou, Daniel Korat, Moshe Berchansky, Gaurav Jain, Oren Pereg, Moshe Wasserblat, and David Harel. Accelerating llm inference with lossless speculative decoding algorithms for heterogeneous vocabularies, 2025. URL https://arxiv.org/abs/2502.05202.

Junxiong Wang, Daniele Paliotta, Avner May, Alexander M. Rush, and Tri Dao. The mamba in the llama: Distilling and accelerating hybrid models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yuhao Wu, Ming Shan Hee, Zhiqiang Hu, and Roy Ka-Wei Lee. Longgenbench: Benchmarking long-form generation in long context llms. In *International Conference on Learning Representations*, volume 2025, pp. 6851–6872, 2025.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3909–3925, 2023.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. Swift: On-the-fly self-speculative decoding for llm inference acceleration. *arXiv preprint arXiv:2410.06916*, 2024.

Tianhua Xia, Mugilan Ganesan, Yifei Feng, Haiyu Wang, Maximilian Egger, and Sai Qian Zhang. Acceptance-aware draft model training for speculative decoding. *arXiv preprint arXiv:2609.24150*, 2026.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. Inference with reference: Lossless acceleration of large language models. *arXiv preprint arXiv:2304.04487*, 2023.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025b. URL https://arxiv.org/abs/2412.06464.

Xi Ye, Fangcong Yin, Yinghui He, Joie Zhang, Howard Yen, Tianyu Gao, Greg Durrett, and Danqi Chen. Longproc: Benchmarking long-context language models on long procedural generation. *arXiv preprint arXiv:2501.05414*, 2025.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*, 2023a.

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

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024a.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations (ICLR)*, 2024b.

APPENDIX OVERVIEW

This appendix has two parts:

- Appendix A (method details): the complete TapKV algorithm, notation, and exact budget allocation.
- Appendix B (additional experiments): tree-drafting results, pruning cost, sensitivity to operating conditions, compatibility with hybrid and fine-tuned drafts, analyses of the pruning pattern and the per-layer budget allocation, and TriForce details.

A METHOD DETAILS

A.1 TAPKV ALGORITHM

Algorithm 1 TapKV: speculative decoding with target-aware draft KV-cache pruning.

Require: draft M_q, target M_p, prompt $x_{1:n}$; draft length γ; average per-layer budget B_{tot}; recent-buffer size R; replay window W; per-layer bounds $b_l^{\min}, b_l^{\max}$

Prefill $x_{1:n}$ into draft cache $\mathcal{C}^d$ and target cache $\mathcal{C}^p$; store the target next-token distributions
if $|\mathcal{C}^d| \geq B_{\text{upper}}$ **then**
  $\mathcal{C}^d \leftarrow \textsc{Prune}(\mathcal{C}^d, x_{1:n})$
end if
while not finished **do**
  Draft $\hat{y}_{1:\gamma}$ with M_q over $\mathcal{C}^d$ ▷ cost bounded by B_{upper}
  Verify $\hat{y}_{1:\gamma}$ with M_p over the full $\mathcal{C}^p$; commit $y_{1:n_{\text{acc}}+1}$ and store their target distributions
  Append $y_{1:n_{\text{acc}}+1}$ to x, and their KV entries to $\mathcal{C}^d$ and $\mathcal{C}^p$
  if $|\mathcal{C}^d| \geq B_{\text{upper}}$ **then** ▷ about once every $R/(\alpha+1)$ rounds
    $\mathcal{C}^d \leftarrow \textsc{Prune}(\mathcal{C}^d, x)$
  end if
end while

function $\textsc{Prune}(\mathcal{C}^d, x)$
  $\mathcal{T} \leftarrow$ the W most recent committed tokens
  Replay $x_{\mathcal{T}}$ through M_q over $\mathcal{C}^d$; record $a^h_{t,t'}$, $\mathbf{o}^h_t$, and $\mathbf{q}$ for $t \in \mathcal{T}$
  Compute $\mathcal{L}(\mathbf{p}, \mathbf{q})$; backpropagate once to get $\nabla_{\mathbf{o}^h_t}\mathcal{L}$ ▷ M_q and $\mathbf{p}$ frozen
  for each layer l and cached token t' **do**
    $\Delta\mathbf{o}^h_{t,t'} \leftarrow$ Eq. (1) for all h and $t \in \mathcal{T}$
    $s^{\text{AOT}}_{t',l} \leftarrow \sum_h \sum_{t\in\mathcal{T}} \left|\langle \nabla_{\mathbf{o}^h_t}\mathcal{L}, \Delta\mathbf{o}^h_{t,t'}\rangle\right|$ ▷ AOT, Eq. (3)
  end for
  $\mathcal{K}_l \leftarrow F_l$ (sink, recent-window, and replayed tokens of layer l), for all l
  for each layer l **do** ▷ stage 1: lower bounds
    Add top-scoring positions of layer l to $\mathcal{K}_l$ until $|\mathcal{K}_l| = b_l^{\min}$
  end for
  for unselected (t', l) in descending $s^{\text{AOT}}_{t',l}$ **do** ▷ stage 2: global fill
    if $\sum_l |\mathcal{K}_l| = B_{\text{sum}}$ **then break** ▷ $B_{\text{sum}} = N_{\text{layer}} B_{\text{tot}}$ when feasible, App. A.3
    if $|\mathcal{K}_l| < b_l^{\max}$ **then** $\mathcal{K}_l \leftarrow \mathcal{K}_l \cup \{t'\}$
  end for
  Compact layer l of $\mathcal{C}^d$ to $\mathcal{K}_l$ in sequence order; free emptied pages, for all l
  return $\mathcal{C}^d$
end function

A.2 NOTATION

Table 4 summarizes the notation used in the main text and the appendix.

Symbol	Meaning
Speculative decoding	
M_p, M_q	Target and draft models
γ	Draft window size, i.e., tokens proposed per round
α	Average number of accepted draft tokens per round
T_{verify}, T_{draft}, T_{prune}	Mean verification latency, per-token drafting latency, and amortized pruning overhead per round
β_{sd}	Approximate speculative decoding throughput
x, n_{acc}	Token sequence (prompt and committed tokens), and number of draft tokens accepted in the current round
AL, SP	Acceptance length and speedup over target autoregressive decoding
Draft cache and budget	
$\mathcal{C}^d$, $\mathcal{C}^p$	Draft and target KV caches
N_{layer}, l, h	Number of draft layers, layer index, and attention head
$\mathcal{V}_l$, F_l, $\mathcal{U}_l$	Live, protected (sink, recent-window, and replay), and candidate cache positions in layer l, with $\mathcal{U}_l = \mathcal{V}_l \setminus F_l$
B_{tot}, B^l_{tot}	Average per-layer retention budget, and allocated budget of layer l, with $\sum_l B^l_{\text{tot}} = N_{\text{layer}} B_{\text{tot}}$
$b^{\min}_l$, $b^{\max}_l$	Lower and upper bounds on the retained size of layer l
$m_{\min}$, $m_{\max}$	Multipliers defining the layer-budget bounds
R, B_{upper}, B^l_{upper}	Recent-buffer size, average per-layer cache size at the pruning trigger $B_{\text{tot}} + R$, and its per-layer value $B^l_{\text{tot}} + R$
P	Number of tokens per KV-cache page
$z_{l,t'}$, $\mathcal{K}_l$	Binary decision to retain token t' in layer l, and the retained set of layer l
Target-guided scoring	
W, $\mathcal{T}$	Replay window size and set of replayed tokens
y_i, p_i, q_i	The ith verified token in the replay window, and the probabilities assigned to it by the target and draft models
$\mathbf{p}$, $\mathbf{q}$	$\{p_i\}_{i=1}^W$ and $\{q_i\}_{i=1}^W$
$\mathcal{L}(\mathbf{p}, \mathbf{q})$	Target-aware scoring loss
t, t'	Replayed (query) token and cached token
H_q, H_{kv}, $\pi(h)$	Query-head count, KV-head count, and query-to-KV head mapping
$\mathbf{q}^h_t$, $\mathbf{k}^h_{t'}$, $\mathbf{v}^h_{t'}$	Query of token t and cached key and value of token t' at head h, of dimension d_{head}
$\mathcal{I}_t$	Cached positions visible to query t under the causal mask
$a^h_{t,t'}$, $\mathbf{o}^h_t$	Attention weight of token t on token t', and attention output of token t, at head h
$\Delta\mathbf{o}^h_{t,t'}$	Change in $\mathbf{o}^h_t$ after deleting the KV entry of token t'
$\Delta\mathcal{L}_{t'}$	First-order change in $\mathcal{L}(\mathbf{p}, \mathbf{q})$ from deleting token t' in one layer
$s^{\text{AOT}}_{t',l}$	AOT score of cached token t' in layer l

Table 4: Notation for layer-wise draft-cache selection.

A.3 Layer Caps and Exact Allocation

Let $m_{\min}$ and $m_{\max}$ be dimensionless budget multipliers satisfying $0 \le m_{\min} \le m_{\max}$. With all layer sums taken over $l = 1, \dots, N_{\text{layer}}$, define

$$b^{\min}_l = \min(|\mathcal{V}_l|, \max(|F_l|, \text{round}(m_{\min} B_{\text{tot}}))) \,, \tag{4}$$

$$b^{\max}_l = \max\big(b^{\min}_l, \min(|\mathcal{V}_l|, \text{round}(m_{\max} B_{\text{tot}}))\big) \,, \tag{5}$$

$$B_{\text{sum}} = \min\left(\sum_l b^{\max}_l, \max\left(N_{\text{layer}} B_{\text{tot}}, \sum_l b^{\min}_l\right)\right) . \tag{6}$$

The total B_{sum} is $N_{\text{layer}} B_{\text{tot}}$ whenever that value is feasible. The layer bounds respect the protected positions and available cache entries, while clamping the nominal total to their sums ensures feasibility. All experiments use $m_{\min} = 0.5$ and $m_{\max} = 2$, i.e., per-layer bounds of 512 and 2048 entries at $B_{\text{tot}} = 1024$.

For candidate $t' \in \mathcal{U}_l$, let $z_{l,t'} = 1$ if it is retained and $z_{l,t'} = 0$ otherwise. The retained size of layer l is $B^l_{\text{tot}} = |F_l| + \sum_{t' \in \mathcal{U}_l} z_{l,t'}$. TapKV solves

$$\begin{aligned} \max_{z_{l,t'} \in \{0,1\}} \quad & \sum_{l=1}^{N_{\text{layer}}} \sum_{t' \in \mathcal{U}_l} s^{\text{AOT}}_{t',l} z_{l,t'} \\ \text{s.t.} \quad & \sum_{l=1}^{N_{\text{layer}}} B^l_{\text{tot}} = B_{\text{sum}}, \\ & b^{\min}_l \le B^l_{\text{tot}} \le b^{\max}_l, \qquad l = 1, \dots, N_{\text{layer}}. \end{aligned} \tag{7}$$

Protected entries do not appear in the objective because they are always retained, but they consume the budget through B^l_{tot}.

The two-stage procedure of Section 3.3 solves this problem exactly. Within a layer, exchanging a selected lower-scoring entry for an unselected higher-scoring entry preserves feasibility and cannot decrease the objective. An optimum can therefore be chosen to contain the highest-scoring entries needed for every minimum quota. After these quotas are met, every remaining entry consumes one budget slot, and the global ordering selects the highest scores allowed by the independent layer caps. This gives an exact solution to the fixed-total additive objective in Equation 7. The claim concerns the allocation for fixed scores and does not make the first-order AOT approximation exact. An approximate procedure that estimates and clamps layer budgets independently need not preserve the same total.

B Additional Experiments

B.1 Tree Drafting

Table 5 extends the tree-drafting comparison of Table 2 to all draft-side pruning baselines under greedy verification; TriForce is omitted because it does not support tree drafting. TapKV achieves the highest mean speedup for every pair, 2.83–$2.92\times$ compared with 1.80–$2.22\times$ for SnapKV, StreamingLLM, and KeyDiff and 1.36–$1.72\times$ for vanilla SD, while also retaining the highest mean AL among the pruning methods (6.59–6.94 versus 6.46–6.89).

Table 5: Greedy tree SD with draft-side KV-cache pruning.

		vanilla SD		SnapKV		StreamingLLM		KeyDiff		**TapKV**	
Draft+Target	Benchmark	AL	SP	AL	SP	AL	SP	AL	SP	AL	SP
Qwen3 0.6B+8B	AIME 2025	7.25	1.52	7.08	1.95	6.92	1.91	6.90	1.88	7.05	**3.00**
	LiveCodeBench	7.08	1.57	6.81	1.87	6.75	1.86	6.71	1.85	6.82	**2.90**
	GPQA	6.76	1.53	6.45	1.78	6.36	1.76	6.27	1.71	6.36	**2.69**
	LongProc	7.69	0.83	5.98	1.82	6.10	1.87	5.96	1.79	6.14	**2.73**
	Avg	7.19	1.36	6.58	1.85	6.53	1.85	6.46	1.80	**6.59**	**2.83**
Qwen3 1.7B+14B	AIME 2025	7.35	1.92	7.04	2.28	6.95	2.25	6.88	2.17	7.07	**3.03**
	LiveCodeBench	7.18	1.96	6.89	2.21	6.79	2.18	6.74	2.11	6.90	**2.92**
	GPQA	6.75	1.89	6.40	2.05	6.36	2.04	6.26	1.95	6.46	**2.73**
	LongProc	7.85	1.10	6.81	2.34	6.81	2.32	6.80	2.29	6.86	**3.01**
	Avg	7.28	1.72	6.78	2.22	6.73	2.20	6.67	2.13	**6.82**	**2.92**
DS-R1-Distill 1.5B+14B	AIME 2025	7.60	1.99	7.52	2.40	7.47	2.39	7.46	2.39	7.49	**3.09**
	LiveCodeBench	7.43	1.94	7.29	2.26	7.23	2.25	7.30	2.27	7.34	**2.97**
	GPQA	6.72	1.80	6.43	1.99	6.49	2.01	6.52	2.02	6.41	**2.59**
	LongProc	6.99	1.08	6.12	2.12	5.87	2.03	6.28	2.18	6.51	**2.81**
	Avg	7.19	1.70	6.84	2.19	6.76	2.17	6.89	2.22	**6.94**	**2.87**

B.2 Amortizing Scoring Cost

Following the performance model in Section 3, we amortize the cost of each continual pruning event, including the replay forward and backward pass required by AOT scoring, over the speculation

rounds between consecutive events, which yields the per-round term T_{prune}. With the default buffer $R = 1024$, TapKV adds only 0.40 ms per round, or 0.58% of the round latency.

Table 6 sweeps the recent-buffer size R, which sets the pruning frequency, and compares TapKV with two strong draft-side pruning baselines, SnapKV and StreamingLLM, at a fixed $B_{\text{upper}} = 2048$ (Qwen3-0.6B drafting for Qwen3-8B on AIME 2025). For each method we report the acceptance length (AL), the speedup over target autoregressive decoding (SP), and the pruning overhead (ovh), i.e. the pruning time amortized over speculation rounds, as a percentage of the round latency. A buffer of R leaves a retained budget of $B_{\text{tot}} = 2048 - R$ and triggers $1000/R$ pruning events per 1000 committed tokens (ev./1k). All rows were run on the same GPU. TapKV runs on the layer-wise paged kernel, which supports its layer-specific budgets. SnapKV and StreamingLLM use a uniform per-layer budget, so their caches have the same length in every layer and run on a dense SDPA cache without padding.

The sweep reveals a trade-off in the choice of buffer size. Because B_{upper} is fixed, a smaller buffer leaves a larger retained budget, which generally raises AL, but it also triggers pruning more often and keeps a larger draft cache, which lowers SP. A larger buffer reduces the retained budget, so the AL of the three methods decreases and converges, and the speedup also drops. $R = 1024$ balances the two and gives TapKV its highest SP.

Across the whole range, the pruning overhead of TapKV stays small, reaching at most 4.59% at the most frequent setting ($R = 128$). Even with the backward pass required by AOT scoring, pruning is therefore never the bottleneck of speculative decoding.

Table 6: Buffer sweep on AIME 2025 ($B_{\text{upper}} = 2048$). ovh: pruning share of decode time (%).

		TapKV			SnapKV			StreamingLLM		
Buffer	ev./1k	AL	SP	ovh	AL	SP	ovh	AL	SP	ovh
256	3.91	5.12	2.05	2.41	5.05	1.68	0.65	4.88	1.64	0.12
512	1.95	5.05	2.00	1.18	5.03	1.68	0.32	4.89	1.65	0.06
1024†	0.98	5.05	2.09	0.58	4.89	1.73	0.15	4.73	1.67	0.03
1536	0.65	4.78	1.99	0.40	4.78	1.64	0.10	4.72	1.60	0.02

†Default buffer; AL and SP repeat the AIME 2025 entries of Table 1.

B.3 Sensitivity to Speculation Length and Generation Length

Table 7 sweeps the speculation length γ around the default setting; the budget sweep is shown in Figure 4(a). Increasing γ raises AL, and the speedup of TapKV peaks at $\gamma = 7$. We also extend the generation length of TapKV from 8K to 32K tokens: its AL stays at 5.03–5.05, while its speedup rises from $2.09\times$ to $2.23\times$, consistent with its bounded drafting cost.

Table 7: Speculation-length sweep on AIME 2025 around the default setting (†), whose row repeats the AIME 2025 entries of Table 1.

	TapKV		SnapKV		StreamingLLM	
Setting	AL	SP	AL	SP	AL	SP
Speculation length γ						
3	3.23	1.86	3.21	1.65	3.18	1.64
5	4.22	2.05	4.16	1.73	4.09	1.71
7†	5.05	2.09	4.89	1.73	4.73	1.67
9	5.57	2.04	5.53	1.65	5.31	1.59

B.4 Compatibility with Hybrid and Linear-Attention Drafts

TapKV also applies to hybrid drafts that combine linear attention with full attention Wang et al. (2024); Choi et al. (2025).

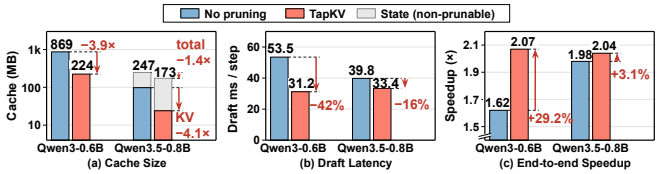

Figure 5: **TapKV on a dense and a hybrid draft on AIME 2025.** **(a)** Draft cache memory. The grey outline marks the hybrid's recurrent state, which TapKV does not prune. **(b)** Draft time per SD step. **(c)** Speedup over target autoregressive decoding.

As a representative hybrid draft, we use the Qwen3.5 family Qwen Team (2026) (0.8B drafting for 9B), which interleaves Gated DeltaNet Yang et al. (2025b) and full attention in a 3:1 ratio. Only 6 of the draft's 24 layers therefore hold a KV cache that grows with the context. TapKV prunes only these layers and leaves the constant-size recurrent state of the Gated DeltaNet layers unchanged. Figure 5 compares this hybrid pair with the dense Qwen3-0.6B+8B pair under the default setting ($B_{\text{tot}} = R = 1024$, $\gamma = 7$, greedy decoding, 8k generated tokens), using the same attention kernel for the pruned and unpruned runs of each pair.

TapKV reduces the draft KV cache by a similar factor on both architectures ($3.9\times$ for the dense draft and $4.1\times$ for the hybrid, or $1.4\times$ including the recurrent state), but the speedup gain differs: SP rises from $1.62\times$ to $2.07\times$ for the dense pair and from $1.98\times$ to $2.04\times$ for the hybrid pair. The gain depends on how much of the draft step is spent attending over the draft KV cache. The dense draft spends 53.5 ms of its 86.9 ms step there, and pruning reduces this to 31.2 ms. The hybrid draft spends only 39.8 ms there, as most of its layers carry no growing cache, so pruning shortens its step by 8.7%, a saving that is largely offset by its lower acceptance length. Bounding the draft cache therefore helps most when that cache dominates the drafting cost.

A control run confirms that the gain comes from the bounded cache rather than from the cache layout: raising the hybrid's budget to 4096, at which the pruned draft retains as much KV as the unpruned one, restores the draft step to 40.2 ms and the throughput to within 0.8% of the unpruned run.

B.5 Compatibility with Fine-Tuned Drafts

Weight fine-tuning Zhou et al. (2024b); Liu et al. (2024b); Goel et al. (2024) and draft-side KV pruning are compatible (Table 8): fine-tuning improves draft–target agreement through the draft's weights, whereas TapKV reduces the cost of each draft step by bounding its KV cache. For draft fine-tuning, following DistillSpec Zhou et al. (2024b) and Xia et al. (2026), we use a KL loss for greedy SD and a TV loss for speculative sampling. The two gains do not conflict: the fine-tuned draft with TapKV achieves the best speedup under both greedy decoding and $T{=}0.6$.

Table 8: SFT drafts on AIME 2025 (Qwen3-0.6B+8B).

	Greedy (KL)		**$T{=}0.6$** (TV)	
KV cache	AL	SP	AL	SP
Base draft				
Full	5.29	1.62	5.11	1.53
TapKV	5.02	2.17	4.80	2.04
Fine-tuned draft				
Full	5.38	1.68	5.18	1.60
TapKV	5.19	**2.21**	4.97	**2.10**

The two act on different terms: fine-tuning raises AL by 0.07–0.09 with the full cache and by 0.17 with TapKV at an unchanged draft cost, whereas TapKV trades 0.19–0.31 AL for a much cheaper draft step. SP nevertheless rises with TapKV in all four paired comparisons, by 0.50–0.55. Fine-tuning a draft is therefore not an alternative to bounding its cache, and the AL gain from fine-tuning is preserved when the cache is pruned.

B.6 Pruning Pattern Analysis

To understand which KV entries TapKV retains, we record the draft KV cache after each pruning event for one AIME 2025 problem (Qwen3-0.6B+8B, default setting of Section 4.1). Figure 6 visualizes the result for three representative layers.

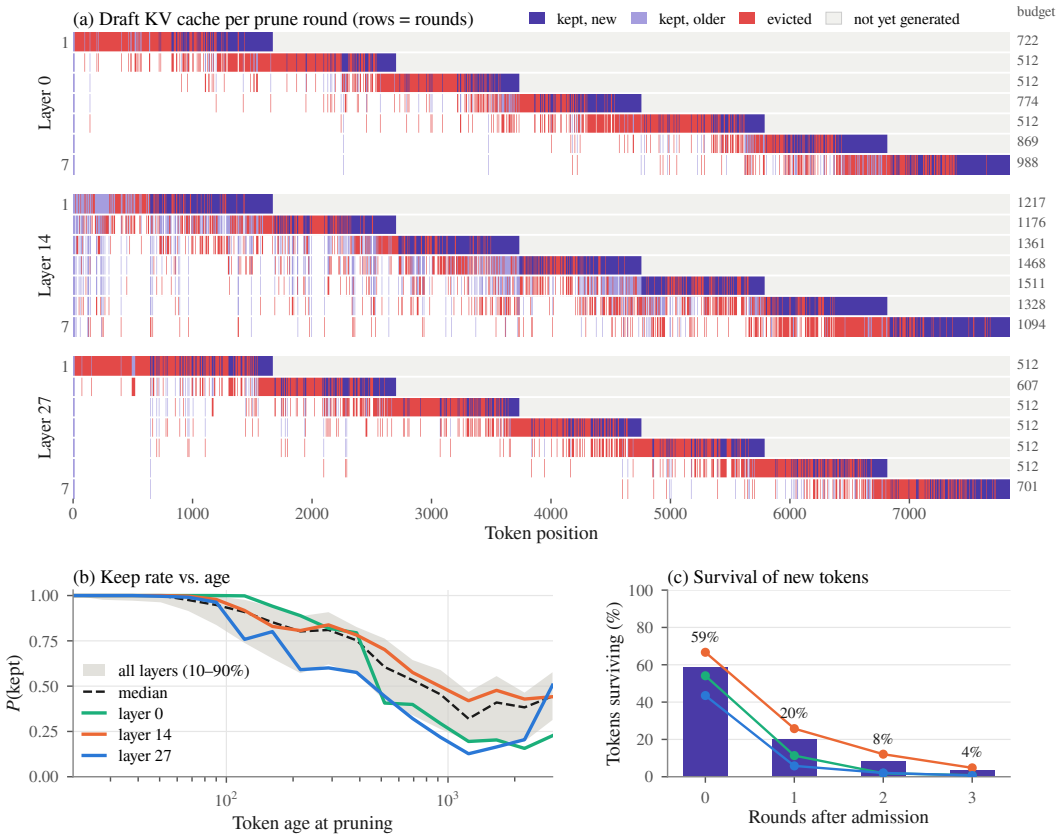

Figure 6: Draft KV-cache pruning pattern of TapKV on one AIME 2025 problem (Qwen3-0.6B+8B). (a) Cache content after each pruning event for layers 0, 14, and 27; each row is one event, and the number on the right is the number of KV entries that layer retains after the event. (b) Keep rate as a function of token age, i.e. the current decoding position minus the token position, at the time of pruning. (c) Fraction of the tokens generated between two events that survive k further events after their first pruning event.

The recent window supplies most of the retained budget. In Figure 6(a), most of the entries kept at each event are tokens generated since the previous event, while tokens from earlier rounds are mostly evicted. Only a few older tokens survive across several events, and they appear as sparse, isolated positions rather than contiguous spans. Retention is therefore largely driven by recency, with sparse long-range hits. The pattern is consistent across shallow, middle, and deep layers, although the per-layer counts differ: under the global allocation of Section 3.3, layer 14 retains 1094–1511 entries per event, while layers 0 and 27 often stay at the 512-entry lower bound.

Keep rate versus token age is U-shaped. Figure 6(b) groups the candidate entries by age. Tokens younger than about 100 steps are almost always kept, most entries are evicted soon after they leave the recent window, and the keep rate reaches its minimum at an age of roughly 10^3 steps. Beyond that, the keep rate rises again: the oldest tokens have already survived several events, so the ones still in the cache form a small set of entries that TapKV repeatedly identifies as important. Figure 6(c) gives the same view per admission cohort: 59% of newly generated tokens survive their first pruning event, but only 20%, 8%, and 4% survive one, two, and three further events.

These observations explain why recency-based heuristics such as StreamingLLM remain strong baselines, and also where TapKV improves on them: it spends most of the budget on recent context but retains the sparse long-range entries that a fixed recent window would discard.

B.7 Adaptive Budget Allocation Across Layers

Figure 7 shows how the global allocation of Section 3.3 distributes the KV budget over the 28 layers of the Qwen3-0.6B draft (Qwen3-8B target, AIME 2025), with an average budget of 1024 entries per layer and per-layer bounds of 512 and 2048.

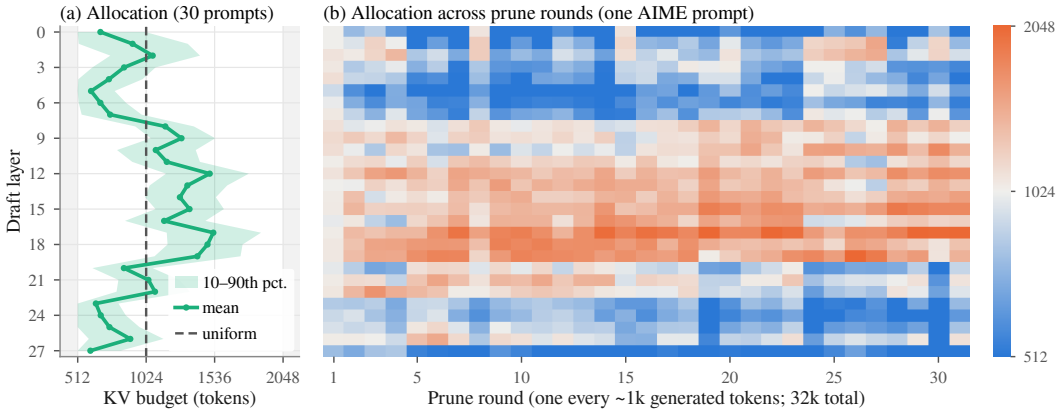

Figure 7: Per-layer KV budget assigned by TapKV. (a) Budget per layer, averaged over pruning events, for 30 AIME 2025 prompts; the dashed line is the uniform allocation. (b) Budget per layer at each pruning event for one prompt.

Middle layers receive more budget than shallow and deep layers. In Figure 7(a), the middle layers (roughly layers 8–19) are allocated above the uniform budget, while the shallow and deep layers are mostly allocated below it. The target-guided scores thus rate KV entries in the middle of the draft as more important for matching the target's output than those at either end.

A few layers are consistently more important. Within this trend, some layers stand out: layers 12 and 17–18 receive about $1.5\times$ the uniform budget on average. Figure 7(b) shows that this ranking is stable across pruning events within a single generation, with the same middle layers staying near the upper bound and the deepest layers near the lower bound. A uniform per-layer budget cannot express this preference, which is why adaptive allocation raises AL by 0.11 over uniform allocation (Section 4.2.2).

B.8 TriForce

TriForce Sun et al. (2024) speculates in a three-tier hierarchy. A small draft model with a StreamingLLM cache Xiao et al. (2024), which keeps a few attention-sink tokens and a window of recent tokens, proposes γ_1 tokens to an intermediate draft. The intermediate draft runs the target weights over a retrieval cache: the target KV cache is split into fixed-size chunks, and a fixed budget of chunks is selected by the attention between the current query and the average key of each chunk, with the leading and most recent chunks always retained. Once the intermediate draft has accumulated γ_2 tokens, the target verifies them with its full KV cache. The retrieval cache is rebuilt at a fixed stride of generated tokens. Table 9 lists these hyperparameters.

The official configuration is the released one, which also performs best in our sweep and is used in Table 1. The released TriForce builds the retrieval cache once after prefill and decodes at most 256 tokens, so it never rebuilds the cache; for long generation, we rebuild it every 256 tokens to match this horizon. The aligned configuration matches the speculation length of TapKV ($\gamma_2 = \gamma = 7$) and its update interval (1024 tokens), and sizes both the StreamingLLM cache (1552 tokens) and

the retrieval cache (1536 tokens) between $B_{\text{tot}} = 1024$ and $B_{\text{upper}} = 2048$. Table 10 compares the two on Qwen3-0.6B+Qwen3-8B under greedy decoding. The aligned configuration raises AL on every benchmark (7.47–7.91 versus 6.99–7.24) but lowers the speedup to 0.97–1.17×, below the 1.11–1.46× of the official configuration; we therefore report the official configuration in the main results.

Table 9: TriForce hyperparameters. Official: released configuration; aligned: matched to TapKV.

Tier	Hyperparameter	Unit	Official	Aligned	Meaning
Small draft (StreamingLLM)	`n_sink`	token	4	16	Attention-sink tokens kept in the cache
	`n_recent`	token	252	1536	Recent tokens kept in the cache
Intermediate draft (retrieval cache)	`chunk_size`	token	8	16	Tokens per KV chunk
	`budget_chunks`	chunk	512	96	Chunks in the retrieval cache
	`retr_sink_chunks`	chunk	1	1	Leading chunks always retained
	`retr_recent_chunks`	chunk	2	32	Most recent chunks always retained
Speculation	`gamma_in` (γ_1)	token	2	2	Tokens the small draft proposes to the intermediate draft
	`gamma_out` (γ_2)	token	6	7	Tokens verified by the target with its full cache
	`retr_refresh_every`	token	256[†]	1024	Generated tokens between retrieval-cache rebuilds

[†]The released TriForce retrieves once after prefill and decodes at most 256 tokens; we rebuild every 256 tokens to match this horizon.

Table 10: Official and aligned TriForce configurations on Qwen3-0.6B+Qwen3-8B (greedy).

	AIME 2025		LiveCodeBench		GPQA		LongProc	
Configuration	AL	SP	AL	SP	AL	SP	AL	SP
Official	7.14	**1.19**	7.17	**1.16**	7.24	**1.11**	6.99	**1.46**
Aligned	7.91	1.03	7.62	0.98	7.64	0.97	7.47	1.17