Task-Aware KV Cache Compression for LLM Agents via Utility-Driven Step Pruning

Yusen Wu[1], **Yefan Wang**[4], **Jia Yee Tan**[5], **Rongfeng Guo**[6], **Shuang Chen**[7], **Jing Yang**[8], **Por Lip Yee**[8], **Yongtai Liu**[9], **Guangyuan Dong**[2,3] *†

[1] Fujian University of Technology [2] National University of Singapore
[3] Ant International Research [4] University of Shanghai for Science and Technology
[5] Renmin University of China [6] Shenzhen University [7] Seed, ByteDance
[8] Universiti Malaya [9] Hanyang University

Abstract

KV-cache compression for long-horizon LLM agents is often guided by accumulated attention, yet attention frequency can retain repetitive error traces while discarding decisive instructions. We propose **TaskKV**, a streaming step-level retention policy for sparse agentic trajectories with sufficient redundancy or failure structure to distinguish useful from redundant context. It has three components: (1) a *signed directional utility* label from the first-order change in action likelihood, distinguishing helpful steps from confusing distractors unlike unsigned saliency; (2) a *structured residual scorer* that treats attention mass and intent alignment as fixed priors and learns only a denoising correction, converging with 500 trajectories; and (3) an entropy-based *regime classifier* that falls back to contiguous retention when the scorer cannot confidently rank steps. With a 20% KV budget on AgentBench and AlfWorld, TaskKV preserves $\sim$87–89% of full-cache success rate, improving over SnapKV by 5.0–7.9 SR points and over a strong heuristic by 2.9–7.3 points. These gains are scoped to the evaluated sparse agentic settings; on dense tasks, the safeguard detects when utility pruning should be disabled.

1 Introduction

The paradigm of Autonomous Agents powered by Large Language Models (LLMs) has significantly advanced problem-solving Xi et al. [2025], Wang et al. [2023]. Unlike single-turn chat, agentic tasks are characterized by long-term interactions, often spanning thousands of reasoning steps. However, deploying these agents faces two practical challenges:

Challenge 1: The Memory Bottleneck. The key–value (KV) cache grows linearly with sequence length. At very long contexts, this becomes a serving bottleneck: for example, a Llama-3.1-70B model with a 128k context window requires over $40\,\mathrm{GB}$ of KV memory per sequence (and $> 1.2\,\mathrm{TB}$ for a standard batch of 32). This motivates compression, and our empirical contribution addresses the algorithmic layer: utility-aware memory selection for sparse agentic trajectories, with serving measurements on a PagedAttention-style prototype demonstrating that the scorer overhead does not negate the memory savings (Section C.5). Furthermore, standard models often struggle to effectively utilize mid-context information Liu et al. [2024a] or extrapolate beyond training lengths Press et al. [2021].

Challenge 2: The Attention-Value Misalignment. Current efficient inference techniques, such as H2O Zhang et al. [2023] and SnapKV Li et al. [2024], employ eviction policies based on accumu-

*Corresponding author.
†Project leader.

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

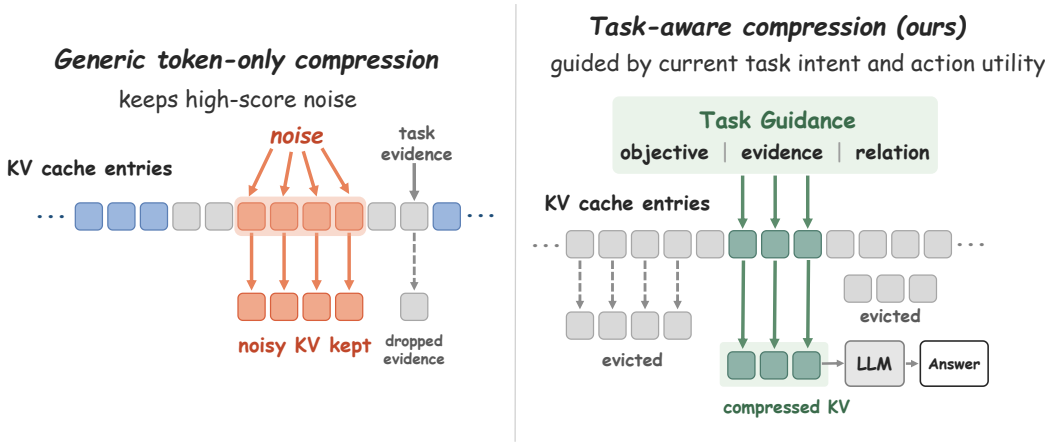

Figure 1: **The Attention–Utility Misalignment in Agentic Memory.** Left: generic token-only compression retains high-score/noisy KV entries while dropping task evidence. Right: TaskKV conditions retention on the task objective, evidence, and relation, preserving utility-bearing entries while pruning redundant/noisy ones before the LLM. This is a conceptual schematic; the blocks illustrate retention choices and do not represent measured token counts or experimental results.

lated attention scores ("Heavy Hitters"). While effective for language modeling perplexity, we argue these methods are less effective for agentic reasoning. We identify a *Misalignment*: in trial-and-error trajectories, failed attempts and verbose observations often receive high aggregate attention due to recency bias or token frequency, yet they constitute "noise" for the optimal decision path. Conversely, critical instructions (e.g., system prompts, distant sub-goals) may receive sparse attention but are indispensable for consistency.

To address these challenges, we propose **TaskKV**, a step-level KV-cache retention system for online LLM agents. TaskKV is deliberately scoped: it is not a universal long-context compressor, but an empirical system for sparse agentic trajectories where a small number of prior steps disproportionately determine future actions. The central claim is concise: *utility-trained step retention beats attention-based token retention on sparse agentic traces under equal budget*.

Contributions.

- **Problem & signal:** We characterize the attention–utility mismatch on agentic trajectories, and introduce a *signed directional utility* label (first-order change in ground-truth action likelihood) that separates causally helpful from causally confusing steps—unlike unsigned gradient magnitude (saliency).
- **Architecture:** A *structured residual scorer* decomposes step ranking into fixed priors (attention mass, intent cosine) and a learned denoising residual conditioned on a time-varying intent vector; this converges with 500 trajectories (Figure 3a). An entropy-based *regime classifier* detects dense contexts and falls back to contiguous retention.
- **Evidence:** On AgentBench (OS, DB, KG) and AlfWorld with a 20% KV budget, TaskKV improves over SnapKV by 5.0–7.9 SR points, over H2O by 8.6–12.6 SR points, and over a strong training-free heuristic by 2.9–7.3 SR points. We include proxy–diagnostic comparisons, error decomposition, and explicit boundary conditions below 10% budget and on dense long-context tasks.

2 Related Work

Efficient LLM Inference. H2O Zhang et al. [2023], StreamingLLM Xiao et al. [2023], and SnapKV Li et al. [2024] rely on task-agnostic attention statistics ("Heavy Hitters"). Other approaches investigate alternative attention architectures Kitaev et al. [2020], contextual sparsity Liu et al. [2023], model-guided policies Ge et al. [2023], or composite tokens Akulov et al. [2025]. CentralKV models attention as a dynamic sparse causal graph and preserves bridge tokens using online centrality Wu et al. [2026b]; TaskKV instead ranks semantic steps using task-conditioned action utility. Orthog-

onal to token selection, KV-precision methods such as KIVI and KVC-Q reduce per-token storage, with KVC-Q dynamically allocating precision according to recency, importance, and head sensitivity Liu et al. [2024c], Wu et al. [2026a].

Emerging Learned Mechanisms. Recent studies explore learnable compression but often optimize for reconstruction/PPL. For instance, **ChunkKV** Liu et al. [2025a] and **KeepKV** Tian et al. [2025] optimize global coherence but rely on retrospective processing ($O(T^2)$), violating streaming causality. Other methods like **AdaKV** Feng et al. [2024] dynamically adjust layer budgets but still operate on token-level attention. In contrast, **TaskKV** builds on the standard Transformer Vaswani et al. [2017] to target *online streaming decoding*, using a utility-trained scorer to filter "high-attention but low-utility" error traces.

Memory for Agents. Several works enhance agents with external memory Packer et al. [2023], Park et al. [2023], tool-use capabilities Schick et al. [2023], or self-reflection Shinn et al. [2023]. These act as retrieval augmentations or policy modifiers. TaskKV addresses a complementary layer: compressing the model's internal working memory.

Positioning. TaskKV differs along three axes: (i) a signed utility signal that separates helpful from confusing steps, (ii) a residual scorer that learns only the denoising correction on top of attention priors, and (iii) an entropy-based regime classifier that prevents misapplication to dense tasks. These complement rather than replace existing KV-compression mechanisms.

3 Methodology

3.1 Problem Setup and Design Rationale

Consider an agent interaction trajectory $\mathcal{T} = \{s_1, s_2, \ldots, s_t\}$, where each s_i is a semantic step (Thought, Action, or Observation) with token length ℓ_i. The KV cache stores the full history; under a retained-token budget $\mathcal{B}_{tok}$, we must select a subset $\hat{M} \subset \mathcal{T}$ satisfying $\sum_{s_i \in \hat{M}} \ell_i \leq \mathcal{B}_{tok}$ that preserves information useful for the next action A^*_{t+1}.

From attention mass to utility scoring. Attention-based methods (H2O, SnapKV) assume high-attention tokens are high-value, but in agentic trajectories repetitive error logs receive high attention while distant system prompts receive low attention despite being causally essential. TaskKV instead assigns each step a utility score $f(s_i, \mathbf{q}_t)$ conditioned on a time-varying intent vector $\mathbf{q}_t$ and maintains a min-heap under an exact token budget. Because steps have variable length, this online policy is a greedy proxy; we bound the proxy–diagnostic gap formally in Proposition D.2 and verify the bound empirically in Section D.

3.2 TaskKV Framework

TaskKV has a deliberately small default policy: segment the stream into semantic steps, score each completed step with a time-varying utility scorer, and keep retained steps under an exact global token budget with one synchronized mask across layers. The scorer has three default signals: length-normalized attention mass, intent alignment, and a learned residual denoising term. The entropy switch is a runtime guard that disables utility pruning on dense traces. Anchor retention is an optional deployment constraint for safety-critical prompts. SMI denotes Submodular Mutual Information, and LOO denotes Leave-One-Out re-evaluation; both are post-hoc diagnostics rather than components of the deployed policy. The offline SMI diagnostic is non-deployable because it uses expensive LOO labels and greedy re-ranking.

3.2.1 Training Signal for Step Importance

A key technical challenge is obtaining per-step supervision that distinguishes *useful evidence* from *confusing distractors*. Gradient magnitude (saliency) alone cannot: a step that strongly diverts the model away from the correct action has high saliency but *negative* utility. Leave-one-out (LOO) labels resolve this but require $O(T)$ forward passes per trajectory, which is prohibitive for training data construction.

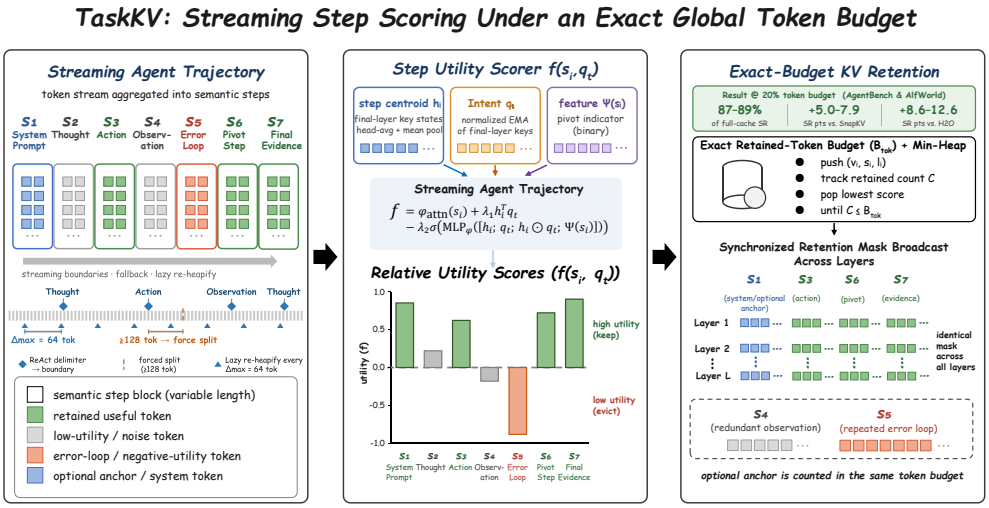

Figure 2: **Overview of the TaskKV framework.** The stream is segmented into semantic steps S_1–S_7 at ReAct delimiters (or by a fixed-stride fallback with $\Delta_{\max} = 64$ and forced splitting after ≥ 128 tokens), represented by final-layer key-state centroids, and scored from attention mass, EMA intent alignment, pivot features, and a learned denoising residual. A min-heap selects complete steps under the exact retained-token budget and the same mask is broadcast across layers; lazy re-heapification updates stale scores. At the 20% budget, the visual summary reports 87–89% of full-cache success rate, +5.0–7.9 SR points versus SnapKV, and +8.6–12.6 versus H2O. An optional safety anchor is counted inside that same budget. When normalized utility entropy is high, the safeguard bypasses learned pruning and retains a contiguous window.

We address this with a **signed directional utility** label. For a step centroid $\mathbf{h}_i$, we compute the first-order approximation to the change in ground-truth action loss under a local masking operation:

$$\mathcal{U}_{proxy}(s_i) = -\left\langle \nabla_{\mathbf{h}_i} \mathcal{L}_{CLM}(A^*), \mathbf{h}_i - \mathbf{h}_{mask} \right\rangle, \tag{1}$$

where $\mathbf{h}_{mask}$ is the zero or mean replacement used for the masking approximation and the gradient is taken with respect to the ground-truth action A^*, not the model's own prediction. The inner product with the masking direction $(\mathbf{h}_i - \mathbf{h}_{mask})$ gives this label its *sign*: positive values predict that keeping s_i lowers the loss (helpful evidence), while negative values predict that the step's presence increases the loss (confusing distractor). This signed distinction is what enables the scorer to learn to *down-weight* high-attention error loops rather than merely identify salient tokens.

We convert these scalar utilities into pairwise preferences and train the scorer with a RankNet objective. This avoids the need for absolute utility calibration—only the relative ordering matters for the greedy selection policy. Gradient norm alone is reported only as an auxiliary sensitivity diagnostic. LOO and offline SMI re-evaluation are used only after training to quantify the proxy–diagnostic gap; they are not used to train the deployed policy.

3.2.2 Semantic Representation & Robust Segmentation

Token-level pruning can break semantic coherence. TaskKV instead operates on *semantic steps* parsed from ReAct-style delimiters Yao et al. [2023]. To handle rare malformed outputs, we use a fixed-stride fallback when no delimiter is detected for an extended span (e.g., $\geq$ 128 tokens), ensuring bounded fragmentation under unstructured text. **Step representation.** For each step span, we compute a compact centroid $\tilde{\mathbf{h}}_i$ and use its L2-normalized version $\mathbf{h}_i = \tilde{\mathbf{h}}_i / \|\tilde{\mathbf{h}}_i\|_2$ in the semantic scorer:

- **Source:** final-layer **key** states (values are excluded since they encode content rather than addressing).
- **Aggregation:** head-average pooling followed by temporal mean pooling over the step tokens.

- **Overhead:** this reduction is performed on-GPU in $< 0.3\,\mathrm{ms}$ per step (A100, BS=32). The scorer-side overhead constitutes $\sim$16% of total per-step latency after feature caching (Table 9).

Practical robustness. Empirically, even with substantial delimiter corruption (30% masked), performance drops only marginally ($\sim$1.5 SR points; Table 18). We treat schema drift and unstructured dialogue as diagnostic stress tests rather than primary evidence; the detailed operating-regime and transfer results are reported in Appendix L.1.

3.2.3 Scorer Design and Intent-Conditioned Ranking

Design principles. A purely heuristic scorer cannot distinguish high-attention error loops from useful steps, while a purely learned scorer must rediscover attention patterns from scratch. TaskKV uses a *structured residual* design: attention mass and intent alignment are fixed priors, and a lightweight MLP learns only the residual denoising correction—concentrating capacity on the hard sub-problem. Furthermore, because a step's utility changes as the agent's sub-goal evolves (e.g., an SQL query critical at t=10 may be noise at t=50), the scorer conditions on a time-varying intent vector $\mathbf{q}_t$, making utility a property of the (step, intent) pair.

Intent Vector ($\mathbf{q}_t$). We compute the intent as a normalized EMA of the final-layer key states:

$$\mathbf{h}(x_t) = \mathrm{RMSNorm}(\mathbf{z}_t^{(L)}), \qquad \mathbf{q}_t = \frac{\mathbf{m}_t}{\|\mathbf{m}_t\|_2}, \quad \mathbf{m}_t = \alpha \mathbf{h}(x_t) + (1-\alpha)\mathbf{m}_{t-1} \tag{2}$$

where $\alpha = 0.1$. The EMA smooths token-level jitter so that $\mathbf{q}_t$ tracks the agent's evolving sub-goal rather than individual token fluctuations. The ablation without smoothing ($\alpha = 1.0$) drops SR by 3.0% (Table 3).

The Scoring Function. The final valuation function $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ combines the base priors with the learned residual:

$$\begin{aligned} f(s_i, \mathbf{q}_t) = \underbrace{\phi_{\text{attn}}(s_i)}_{\text{Syntactic Prior}} + \lambda_1 \underbrace{\mathbf{h}_i^\top \mathbf{q}_t}_{\text{Semantic Alignment}} \\ - \lambda_2 \underbrace{\sigma(\text{MLP}_\phi([\mathbf{h}_i; \mathbf{q}_t; \mathbf{h}_i \odot \mathbf{q}_t; \Psi(s_i)]))}_{\text{Learned Denoising Probability}} \end{aligned} \tag{3}$$

where σ is the Sigmoid function. Since both $\mathbf{h}_i$ and $\mathbf{q}_t$ are L2-normalized, $\mathbf{h}_i^\top \mathbf{q}_t$ is the cosine similarity used in the text. The MLP inputs include the element-wise product $\mathbf{h}_i \odot \mathbf{q}_t$ to explicitly model feature interaction and the scalar pivot feature $\Psi(s_i)$ described next.

Pivot bonus. We include one compact structural feature in the residual scorer: a pivot indicator $\Psi(s_i)$ that fires on strategy-shift boundaries such as transitions from repeated failure to a new tool, schema, room, or sub-goal. In implementation, Ψ is a binary/parser feature concatenated to the MLP input and learned under the same RankNet objective; it is not a separate protected memory pool and it consumes the same token budget. This makes the "w/o Pivot Feature" ablation in Table 3 the removal of this feature, not the removal of an extra rule-based cache.

Update triggers. Scoring activates at semantic boundary delimiters (e.g., `Observation:`) or after a fail-safe stride of Δ_{max}=64 tokens to prevent unbounded buffering.

Hierarchical Pruning & Structural Coherence. To avoid residual-stream misalignment, scores are computed once per completed step using head-and-layer-averaged attention, and the eviction mask is *broadcast* across all L layers simultaneously. Layer-specific pruning (allowing top layers to prune more aggressively) degraded SR by -4.3%, confirming that synchronized masking is a structural necessity for deep Transformers (details in Appendix B).

3.2.4 Comparison with Global Optimization Baselines

We distinguish TaskKV from global optimization methods (e.g., ChunkKV Liu et al. [2025a], KeepKV Tian et al. [2025]). While these methods optimize retrospective perplexity ($O(T^2)$), TaskKV operates under an online streaming constraint (no future lookahead). Appendix O compares these settings empirically and highlights that lower offline PPL does not reliably translate into higher online agent success in our evaluated tasks.

Algorithm 1 TaskKV online streaming inference

Input: Token Stream x_t, Token budget $\mathcal{B}_{tok}$, Scorer q_ϕ, Heap $\mathcal{H}$
Initialize: $\mathcal{H} \leftarrow \emptyset$, retained-token count $C \leftarrow 0$, EMA state $\mathbf{m} \leftarrow \vec{0}$, normalized intent $\mathbf{q} \leftarrow \vec{0}$, token buffer $B_{buf} \leftarrow \emptyset$
while generating token x_t **do**
  $B_{buf} \leftarrow B_{buf} \cup \{x_t\}$
  Update intent state $\mathbf{m} \leftarrow \alpha \mathbf{h}(x_t) + (1-\alpha)\mathbf{m}$
  Normalize $\mathbf{q} \leftarrow \mathbf{m}/\|\mathbf{m}\|_2$
  if IsBoundary(x_t) **or** $|B_{buf}| \geq \Delta_{\text{max}}$ **then**
    Form new step $s_{new} \leftarrow \text{Pool}(B_{buf})$ with token length $\ell_{new} \leftarrow |B_{buf}|$
    Score $v_{new} \leftarrow f(s_{new}, \mathbf{q})$
    $\mathcal{H}$.push($(v_{new}, s_{new}, \ell_{new})$); $C \leftarrow C + \ell_{new}$
    while $C > \mathcal{B}_{tok}$ **do**
      $v_{min}, s_{min}, \ell_{min} \leftarrow \mathcal{H}$.pop()
      Evict KV cache of s_{min} (async if retained, immediate if $s_{min} = s_{new}$)
      $C \leftarrow C - \ell_{min}$
    end while
    $B_{buf} \leftarrow \emptyset$
    if GlobalRecheckTrigger(t) **then**
      {Lazy Re-ranking every K steps to handle intent drift}
      Re-score all $s \in \mathcal{H}$ with current $\mathbf{q}$
      Re-heapify $\mathcal{H}$
    end if
  end if
end while

3.2.5 Online Selection via Min-Heap

Proxy–diagnostic gap. Since the score is modular but the serving constraint is token-budgeted with variable-length steps, the deployed selector is a greedy online proxy rather than an exact global optimizer. Proposition D.2 bounds the proxy–diagnostic gap under sparse-regime assumptions verified on our benchmarks; the offline SMI diagnostic (leave-one-out re-evaluation, $\sim 10^3\times$ more expensive) serves as the empirical reference point for this bound (Table 1).

3.2.6 Regime Detection via Utility Entropy

Utility-driven pruning assumes that a small subset of steps carries most of the information for future actions—a property that holds in sparse agentic trajectories but not in dense tasks like summarization. Rather than requiring the user to select a mode, TaskKV *detects which regime it is in* by monitoring the **normalized entropy of the scorer's own utility distribution**:

$$\bar{H}(f) = \frac{-\sum_i \hat{f}_i \log \hat{f}_i}{\log |\mathcal{H}|}, \tag{4}$$

where $\hat{f}$ is the softmax-normalized score over the current retained-step buffer and $\bar{H}(f) \in [0, 1]$ for $|\mathcal{H}| > 1$.

$$\text{Mode} = \begin{cases} \text{TaskKV Pruning} & \text{if } \bar{H}(f) < \tau \text{ (Sparse Regime)} \\ \text{StreamingLLM} & \text{if } \bar{H}(f) \geq \tau \text{ (Dense Regime)} \end{cases} \tag{5}$$

The intuition is direct: when $\bar{H}(f)$ is high ($\geq \tau = 0.8$), the scorer assigns near-uniform utility, meaning it cannot confidently distinguish important from unimportant steps. In this regime, aggressive pruning would be arbitrary, so TaskKV falls back to contiguous retention (StreamingLLM). This entropy-based regime classifier makes the system self-aware of its own confidence boundary—on LongBench-Summarization it correctly activates in 88% of cases, while triggering in $<$1.5% of steps on sparse agentic benchmarks (Table 11).

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate on **AgentBench** Liu et al. [2024b], encompassing diverse environments: Operating System (OS), Database (DB), and Knowledge Graph (KG), and **AlfWorld** Shridhar et al. [2021]. Additionally, Appendix F.5 reports standard long-context benchmarks (**LongBench**, **Needle**) to characterize when the learned scorer is active and when the density safeguard falls back.

Models. We use **Llama-3.1-70B-Instruct** Grattafiori et al. [2024] and **Mistral-7B-v0.3** Jiang et al. [2023] as primary backbone agents. Table 14 additionally reports a diagnostic transfer run on **Qwen-2.5-7B-Instruct**, where the same training recipe (scorer retrained on target-backbone features) maintains 89.1% of full-cache performance on AgentBench-OS (27.2 vs. 30.5 SR).

Baselines. We compare against Full Cache, Window Attention, StreamingLLM Xiao et al. [2023], H2O Zhang et al. [2023] with equal-budget prompt protection, SnapKV Li et al. [2024], AdaKV Feng et al. [2024], layer-adaptive wrappers (SqueezeAttention Wang et al. [2024], LeanKV Zhang et al. [2024]; cited using the original arXiv v1 title used during our evaluation), and a **Strong Heuristic** that allocates its 20% budget across protected prompt tokens, a recent window, and semantic top-k retrieval. Full descriptions are in Appendix C.1.

Budget & Protocol. All compressed methods operate under a unified retained-token budget $\mathcal{B} = 0.20 \times L_{total}$ with synchronized layer pruning. Prompt protection, recent windows, semantic retrieval, and learned retention all draw from the same 20% token budget; no baseline receives extra uncounted prompt tokens. All reported results use the **TaskKV-Proxy** (trained from signed first-order action-utility labels); the offline SMI diagnostic is included only as a non-deployable comparator. Results are mean $\pm$ standard deviation over 5 seeds. The scorer was trained exclusively on OS and AlfWorld traces; DB and KG results are diagnostic transfer. Full training, reproducibility, and data-hygiene details are in Appendix C.1.

4.2 Main Results

Table 1 summarizes the success rates (SR) under the primary compression constraint (20% cache budget). We report the mean and standard deviation across 5 seeds. OS and AlfWorld are training-matched evaluation domains; DB and KG are held-out diagnostic transfer columns using the same scorer without domain-specific fine-tuning. To probe the limits, we further analyze the 10%–15% regime in the ablation study.

Table 1: Success rate (%) on AgentBench and AlfWorld with a 20% KV-cache budget. Values are mean $\pm$ standard deviation over five seeds; OS and AlfWorld are training-matched, while DB and KG are diagnostic transfer domains. Bold denotes the best deployable method. *Offline SMI Diagnostic* is a non-deployable reference.

Method	OS	DB	KG	AlfWorld
Full Cache (100%)	42.8 $\pm$1.9	54.6 $\pm$1.8	51.3 $\pm$2.2	73.1 $\pm$1.6
Window (20%)	13.8 $\pm$3.6	21.7 $\pm$2.8	24.1 $\pm$3.4	31.2 $\pm$2.7
StreamingLLM	17.2 $\pm$3.1	26.3 $\pm$2.5	23.8 $\pm$3.3	38.5 $\pm$2.1
H2O	28.4 $\pm$2.7	38.7 $\pm$2.0	33.9 $\pm$2.6	52.1 $\pm$1.9
SnapKV	33.1 $\pm$2.3	40.9 $\pm$2.4	39.4 $\pm$1.9	56.8 $\pm$1.7
SqueezeAttn+H2O	26.5 $\pm$2.8	36.2 $\pm$2.3	31.7 $\pm$2.5	48.9 $\pm$2.1
Strong Heuristic	35.2 $\pm$1.7	41.3 $\pm$2.1	38.6 $\pm$1.8	57.4 $\pm$1.5
AdaKV	30.6 $\pm$2.5	41.7 $\pm$1.8	36.8 $\pm$2.4	53.3 $\pm$1.9
Offline SMI Diagnostic	*39.8 $\pm$1.7*	*50.1 $\pm$1.5*	*47.6 $\pm$1.9*	*67.4 $\pm$1.4*
TaskKV-Proxy (Ours)	**38.1 $\pm$2.1**	**47.3 $\pm$1.9**	**45.2 $\pm$2.3**	**64.7 $\pm$1.6**

Fixed priors and learned residual. The AlfWorld module ablation (Table 3) shows that removing semantic alignment has the largest single-module effect (-10.9 SR), compared with -7.6 SR when

Table 2: Prior-only selector versus Strong Heuristic and full TaskKV at a 20% KV budget. Prior-only values are mean ± standard deviation over five seeds; other means are reported in Table 1.

Method	OS	DB	KG	AlfWorld
Strong Heuristic	35.2	41.3	38.6	57.4
Prior-only ($\lambda_2 = 0$)	35.6 ± 2.4	41.6 ± 2.0	39.0 ± 2.4	57.1 ± 2.3
Full TaskKV	38.1	47.3	45.2	64.7
Full − Prior-only	$+2.5$	$+5.7$	$+6.2$	$+7.6$

the learned denoising residual is disabled. Thus, we do not attribute the full gain to the learned residual alone. To compare against a matched prior-only selector across environments, we additionally report the complete scorer architecture with $\lambda_2 = 0$, retaining attention mass, semantic cosine, the pivot feature, and intent EMA while disabling only learned denoising. With Llama-3.1-70B-Instruct, a 20% KV budget, greedy decoding, and five seeds, this prior-only variant remains close to Strong Heuristic; the full system adds 2.5, 5.7, 6.2, and 7.6 SR points on OS, DB, KG, and AlfWorld, respectively.

For DB and KG, an unpaired summary-statistic approximation gives TaskKV–AdaKV gaps of 5.6 and 8.4 SR points, with estimated standard errors of 1.17 and 1.48, respectively (approximate t statistics 4.8 and 5.7). These are descriptive approximations from the reported five-seed means and standard deviations, not paired tests; exact paired inference requires the seed-level paired results.

Qualitative comparison with the strong heuristic. The Strong Heuristic baseline is intentionally competitive: it protects the system prompt, keeps a recent window, and uses semantic top-k retrieval for the remaining budget. A plausible residual failure mode is *coherent failure*: repeated but well-formed invalid tool calls, SQL retries, or observation loops can be semantically close to the task goal and therefore remain highly ranked by cosine similarity. TaskKV's signed, action-conditioned denoising signal is designed to rank such redundant retries below steps associated with useful pivots. On AgentBench-DB, the loop/stuck failure rate is 28.5% for H2O and 12.2% for TaskKV (Table 23). This aggregate comparison is consistent with the proposed mechanism, but does not isolate the denoising term's causal contribution or directly measure this failure mode against the Strong Heuristic. On short AgentBench trajectories (fewer than 20 interaction steps), the scorer adds no measurable benefit over the Strong Heuristic, consistent with there being too little repeated structure to denoise.

High-Efficiency Training. Unlike the offline SMI diagnostic which requires expensive greedy optimization at every step ($O(T^2)$), **TaskKV-Proxy** is trained using first-order action-utility scores.

- **Data Efficiency:** Training converges with just **500 trajectories**. The reported **∼4 GPU-hours** is the method-specific incremental cost after trajectory collection: approximately 2.1 GPU-hours for frozen-feature extraction and signed-label computation, plus 2 GPU-hours for RankNet scorer training on one A100. It excludes trajectory collection and optional offline LOO/SMI diagnostics; the full accounting is given in Appendix C.1.
- **Performance Retention:** As shown in Table 1, the proxy model stays within a few success-rate points of the offline SMI diagnostic in our evaluated agentic settings. This supports the modular proxy empirically, while the proxy–diagnostic gap remains an empirical quantity rather than a formal guarantee.

4.3 Ablation Study and Robustness Analysis

We ablate individual modules and test robustness to budget and segmentation changes.

1. Component-Wise Impact. Table 3 presents the ablation results on AlfWorld.

- **Semantic Alignment (λ_1):** Removing the semantic cosine term ($\lambda_1 = 0$) causes the largest drop (-10.9%), confirming that intent-conditioned relevance is the most critical prior for step ranking in this benchmark.
- **Denoising (λ_2):** Removing the learned noise penalty ($\lambda_2 = 0$) causes the second-largest drop (-7.6%), suggesting that filtering "distractor objects" provides substantial additional value beyond pure semantic recall.

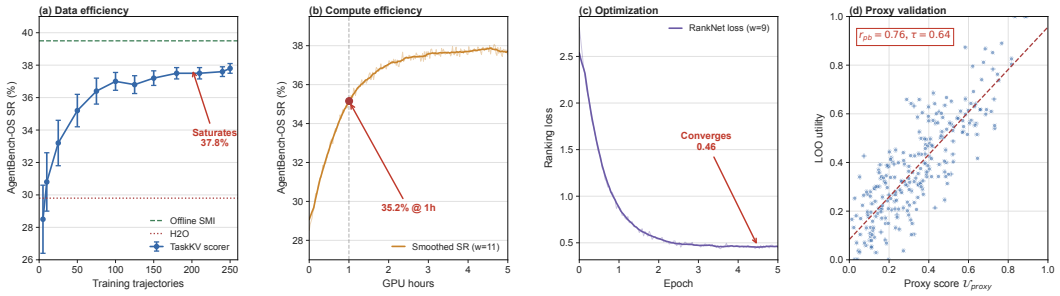

Figure 3: **Efficiency and Training Dynamics of TaskKV-Proxy. (a)** Data efficiency: AgentBench-OS success rate versus training trajectories, saturating at approximately 37.8% (mean $\pm$ std over 5 seeds; Offline SMI and H2O are reference lines). **(b)** Compute efficiency: the scorer reaches about 35.2% at 1 GPU-hour (smoothed with window 11). **(c)** Optimization: RankNet loss converges to approximately 0.46 (smoothed with window 9). **(d)** Proxy validation: the signed first-order proxy agrees with LOO utility annotations, with $r_{pb} = 0.76$ and Kendall's $\tau = 0.64$.

Table 3: Comprehensive ablation analysis on AlfWorld. Values are success rate (%), mean $\pm$ standard deviation across five seeds; parenthetical values are changes relative to Full TaskKV.

Configuration	Success Rate
Full TaskKV	**64.7 $\pm$ 1.6**
Module Ablations	
w/o Denoising (λ_2=0)	57.1 $\pm$ 2.3 (−7.6)
w/o Pivot Feature (Ψ=0)	59.8 $\pm$ 1.8 (−4.9)
w/o Intent EMA (α=1.0)	61.9 $\pm$ 1.7 (−2.8)
w/o Semantic (λ_1=0)	53.8 $\pm$ 2.6 (−10.9)
Training Label Ablation	
Unsigned Saliency ($\lvert\nabla\rvert$only)	58.6 $\pm$ 1.5 (−6.1)
Segmentation Strategy	
Fixed Stride (S=64)	63.1 $\pm$ 1.5 (−1.6)
Fixed Stride (S=32)	62.4 $\pm$ 1.6 (−2.3)
Budget Sensitivity	
Budget = 10%	56.4 $\pm$ 2.2 (−8.3)
Budget = 20%	64.7 $\pm$ 1.6
Budget = 30%	67.9 $\pm$ 1.3 (+3.2)
Budget = 50%	**70.3 $\pm$ 1.2** (+5.6)

- **Unsigned Saliency Labels:** Replacing the signed directional utility $\mathcal{U}_{proxy}$ with unsigned gradient magnitude $\|\nabla_{\mathbf{h}_i}\mathcal{L}\|$ as the training label—while keeping the same scorer architecture and RankNet pipeline—drops SR by 6.1%. This confirms that the sign information is critical: unsigned saliency assigns high importance to both helpful and confusing steps, preventing the denoising head from learning to down-weight high-attention distractors.
- **Pivot Feature:** Removing the pivot indicator Ψ from the scorer weakens retention of strategy-switch steps and increases looping failures, dropping SR by 4.9%.
- **Intent Smoothing (EMA):** Using raw instantaneous intent ($\alpha = 1.0$) introduces high variance, degrading performance by 2.8%.

2. Robustness to Segmentation & Budget Sensitivity. Detailed analysis of segmentation robustness (e.g., fixed stride chunking) and budget scaling (10%–50%) is provided in Appendix L.1. The core finding is that performance scales log-linearly with budget, and the semantic alignment term is robust to boundary jitter (-1.6% with stride 64, -2.3% with stride 32; Table 3).

3. Training Distribution and Decoding. Augmenting training data with aligned failure rollouts improves SR by +0.6% on AlfWorld; under temperature sampling (T=0.7), TaskKV maintains its

relative gain (+7% over H2O). Both results are tabulated in Table 18; details of the mixed-trajectory label construction and decoding robustness are in Appendix J.

4. Failure Mode & PPL–Utility Decoupling. On the held-out DB domain, loop/stuck failures are 28.5% for H2O and 12.2% for TaskKV (Appendix N). This aggregate pattern is consistent with the proposed denoising mechanism but does not isolate its causal contribution. Separately, lower PPL does not reliably imply higher SR: layer-adaptive methods improve PPL while reducing SR by 2–3 points (Appendix E).

5 Limitations and Failure Cases

Where TaskKV fails or offers no advantage. We highlight concrete boundary conditions rather than only positive results, since these regimes are important for deciding when utility-driven pruning should be used:

- **Dense tasks and retrieval limits:** On LongBench-Summarization, the entropy switch fires in 88% of cases, falling back to StreamingLLM; gains come from detecting when *not* to prune. On Needle-In-A-Haystack, TaskKV reaches 94% recall versus 48% for H2O, but remains 6 percentage points below Full Cache (100%).
- **Very low budgets ($\leq$10%):** At 10% budget on AlfWorld, SR drops to 56.4% (vs. 64.7% at 20%), a steep 8.3-point cliff. Below 10%, the scorer cannot retain enough context for multi-step reasoning and performance approaches the sliding-window baseline. On AgentBench-OS at 10% budget, TaskKV still leads (31.2 vs. 18.5 for H2O, Table 20), but absolute performance degrades sharply and falls well below the full-cache reference.
- **Short trajectories:** On AgentBench tasks with fewer than 20 interaction steps, all methods (including Full Cache) perform comparably, and TaskKV's learned scorer adds no measurable benefit over the Strong Heuristic. The denoising head provides value only when there is sufficient noise to filter.
- **Scorer–diagnostic gap:** The proxy scorer stays within the regret bound of Proposition D.2 on all evaluated benchmarks (Section D). As predicted by the bound, the gap widens on highly redundant trajectories (e.g., repetitive SQL retries in DB, measured $\hat{\rho}_{\max} = 0.18$) where the modular proxy's collision penalty increases.

Scope limitations. TaskKV is best supported on sparse, step-structured agent trajectories. Fixed-stride chunking provides an operational fallback when explicit ReAct delimiters are missing, but it does not substitute for evaluation on other agent formats. Our JSON-mode and Reflexion-style results, together with the UltraChat diagnostic in Appendix L.1, are limited format checks and do not establish generalization to arbitrary weakly delimited tool traces. We also report a preliminary WebArena shopping-and-Reddit diagnostic in Appendix I.1; it does not establish performance on the complete WebArena suite, other web-agent implementations, or software-engineering agents. Proposition D.2 and Corollary D.3 provide a formal regret bound under the sparse-regime assumptions, whose empirical tightness is evaluated on the reported domains (Section D). When inter-step redundancy is low ($\rho_{\max} \ll 1$), the modular proxy is expected to track the greedy optimizer more closely; when utility scores are diffuse, the entropy switch (Section 3.2.6) activates contiguous retention. Broader evaluation across web navigation, code repair, and multi-tool software-engineering agents remains future work. Learning TaskKV without ground-truth action references also remains an open limitation because the signed, action-conditioned training signal is unavailable in that setting (Appendix L.2).

6 Conclusion

We present TaskKV, a task-aware KV compression framework for LLM agents built on three technical ideas: (1) a signed directional utility label that separates causally helpful from causally confusing steps, resolving the saliency–utility conflation in gradient-based importance; (2) a structured residual scorer that learns only the denoising correction on top of attention and intent-alignment priors, concentrating learned capacity on the hard sub-problem where heuristics fail; and (3) an entropy-based regime classifier that detects when the scorer's confidence is insufficient and falls back to contiguous retention. Empirically, TaskKV preserves strong agent performance under aggressive KV reduction in sparse agentic settings, while its regime detection and failure analysis identify when dense retention is more appropriate.

Acknowledgments and Disclosures

Funding. This work was supported by Fujian University of Technology and the National University of Singapore.

Competing interests. The authors declare no competing financial or non-financial interests related to this work.

References

Dmitry Akulov, Mohamed Sana, Antonio De Domenico, Tareq Si Salem, Nicola Piovesan, and Fadhel Ayed. Kvcompose: Efficient structured kv cache compression with composite tokens, 2025.

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)*, pages 3119–3137, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. K. Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference, 2024.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 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, et al. Mistral 7b, 2023.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

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*, pages 611–626, 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. In *Advances in Neural Information Processing Systems*, volume 37, pages 22947–22970, 2024.

N. F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. ChunkKV: Semantic-preserving KV cache compression for efficient long-context LLM inference, 2025a.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024b.

Xin Liu, Xudong Wang, Pei Liu, and Guoming Tang. ZeroMerge: Parameter-free KV cache compression for memory-efficient long-context LLMs, 2025b.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient LLMs at inference time. In *International Conference on Machine Learning*, pages 22137–22176, 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 *International Conference on Machine Learning*, 2024c.

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming*, 14(1):265–294, 1978.

Charles Packer, Vivian Fang, S. G. Patil, Kevin Lin, Sarah Wooders, and Joseph Gonzalez. Memgpt: Towards llms as operating systems, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.

Ofir Press, N. A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation, 2021.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 26871–26883, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Yuxuan Tian, Zihan Wang, Yebo Peng, Aomufei Yuan, Zhiming Wang, Bairen Yi, Xin Liu, Yong Cui, and Tong Yang. KeepKV: Eliminating output perturbation in KV cache compression for efficient LLMs inference, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, A. N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023.

Zihao Wang, Bin Cui, and Shaoduo Gan. SqueezeAttention: 2d management of KV-cache in LLM inference via layer-wise optimal budget, 2024.

Yusen Wu, Ruiqin Lin, Jiarong Que, Qixiang Zeng, and Hongsen Zhang. KVC-Q: A high-fidelity and dynamic KV cache quantization framework for long-context large language models. *Journal of Systems Architecture*, 173:103699, 2026a. doi: 10.1016/j.sysarc.2026.103699.

Yusen Wu, Huang Yinjun, Jia Yee Tan, Hao Li, Rongfeng Guo, Liang Fan, Jiachen Luo, Guangyuan Dong, and Sike Xiang. CentralKV: Compressing long-context LLMs via attention-graph centrality. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, pages 8727–8748, Budapest, Hungary, October 2026b. Association for Computational Linguistics. doi: 10.18653/2026.emnlp-main.382. URL `https://aclanthology.org/2026.emnlp-main.382/`.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(1):121101, 2025. doi: 10.1007/s11432-024-4222-0.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, K. R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John Lui, and Haibo Chen. Unifying KV cache compression for large language models with LeanKV, 2024.

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. In *Advances in Neural Information Processing Systems*, volume 36, pages 34661–34710, 2023.

A Extended Visualization: Mechanism and Efficiency

In Figure 4, we provide a comprehensive visualization of the TaskKV mechanism and its efficiency gains. This composite figure illustrates the observed efficiency–performance tradeoff, the layer-averaged retention profile, and the scalability benefits.

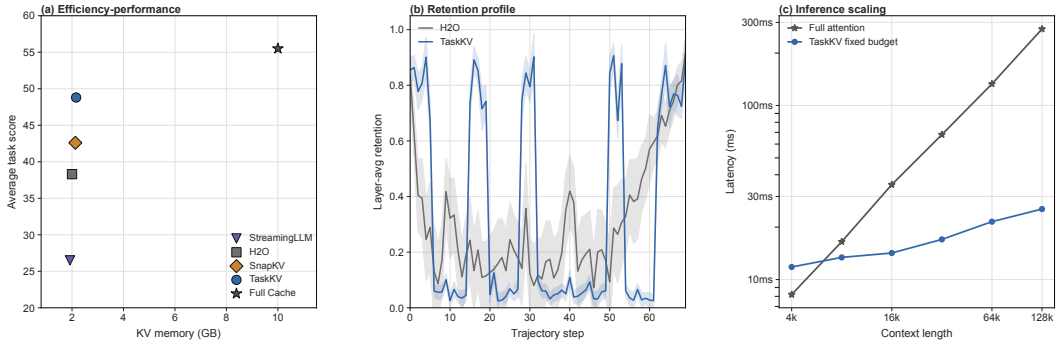

Figure 4: **TaskKV mechanism and efficiency.** **(a)** Efficiency–performance tradeoff (Llama-3.1-70B, 32k context): per-request KV memory versus average task score across all four benchmarks. **(b)** Layer-averaged retention on a representative long trajectory of roughly 65–70 steps (mean $\pm$ standard deviation across 28 layers): H2O concentrates retention on the recent window and attention sinks, whereas TaskKV concentrates retention on semantically meaningful step clusters with generally narrower layer variance away from selected clusters. **(c)** Decode-path latency scaling (log-scaled latency axis): full-cache latency grows with context length, while TaskKV with a fixed retained budget grows much more slowly (about 12–24 ms in the plotted range) and remains nearly flat relative to full attention.

B Design Intuition: Why Step-Level Retention Helps

This appendix provides intuition for why TaskKV's approach outperforms recency-based methods on sparse agentic tasks. The formal regret bound under the Sparse Agentic Regime assumption is in Appendix D.

Key observation. In many agentic tasks, the next correct action depends primarily on: (1) the system prompt (defining the task and constraints), and (2) the current state (the most recent meaningful observation). Intermediate steps—failed attempts, verbose logs, repeated observations—are largely redundant given the current state. A recency-based method (sliding window) keeps the most recent tokens regardless of content, risking eviction of the system prompt once the trajectory is long enough. TaskKV's utility scorer is trained to assign high scores to steps that are causally relevant to the next action, which empirically corresponds to retaining the system prompt and current-state descriptions while pruning noise.

B.1 Complexity Analysis

We compare the time and space complexity of TaskKV against standard methods. Let T be the sequence length, d be the hidden dimension, $\mathcal{B}$ be the retained-token KV budget, and N_s be the number of completed semantic steps.

Scoring Overhead: The scorer is a lightweight MLP acting on cached step features. Its per-step scoring cost is $O(d_{score})$, where d_{score} denotes the residual scorer input and hidden dimensions, and lazy re-ranking costs $O(|\mathcal{H}|d_{score})$ only at recheck events. In our A100-80GB local-hook implementa-

Table 4: Complexity Comparison. TaskKV achieves the same space complexity as sparse baselines while extending the effective recall horizon to the full trajectory via scorer-guided retention.

Method	Space Complexity	Time Complexity (Gen)	Critical Recall Horizon
Full Attention	$O(T \cdot d)$	$O(T^2)$	T (Perfect)
Window Attn	$O(\mathcal{B} \cdot d)$	$O(T \cdot \mathcal{B})$	$\mathcal{B}$ (Short)
H2O / SnapKV	$O(\mathcal{B} \cdot d)$	$O(T \cdot \mathcal{B})$	$\mathcal{B}$ (Stochastic)
TaskKV	$O(\mathcal{B} \cdot d)$	$O(T \cdot \mathcal{B} + N_s d_{score})$	Extended (budget-limited)

tion, the vectorized scorer adds $< 2\,\mathrm{ms}$ per scoring event at batch 32 after feature caching. Table 9 reports end-to-end latency measured in a PagedAttention-style implementation with engine-specific hooks, while the accompanying reference implementation provides the policy logic and simulator rather than a complete drop-in serving backend.

C Implementation Details of Scorer

C.1 Detailed Experimental Setup

Baselines.

- **Full Cache:** No compression; reported as an upper reference rather than a deployable compressed method.
- **Window Attention:** Keeps only the most recent N tokens.
- **H2O (Prompt-Protected):** The standard H2O algorithm Zhang et al. [2023] equipped with equal-budget system-prompt protection (same variant used as "H2O" in Table 1). We reserve part of the same 20% retained-token budget for the initial instruction span and apply Heavy Hitter eviction only to the dynamic remainder, preventing the "Survivor Bias" where baselines fail simply by forgetting instructions.
- **StreamingLLM Xiao et al. [2023]:** Keeps initial "sink" tokens and rolling window.
- **SnapKV Li et al. [2024]:** Clusters attention patterns to identify important windows.
- **AdaKV Feng et al. [2024]:** Adaptive pooling compression.
- **Layer-Adaptive Wrappers:** We evaluate **SqueezeAttention** Wang et al. [2024] and **LeanKV** Zhang et al. [2024] as wrappers that non-uniformly allocate budget across layers based on sensitivity metrics. For LeanKV, we cite the original arXiv v1 title used during our evaluation; later arXiv revisions renamed the work to DiffKV.
- **Strong Heuristic (Sys+Recent+Semantic):** A custom baseline that explicitly retains the system prompt, a recent window, and semantic embedding top-k steps within the same global budget. This prevents TaskKV from winning simply by "remembering the instructions."

Unified Optimization Budget ($\mathcal{B}$). The budget $\mathcal{B}$ is defined as the **Global Token Count** retained in the KV cache: $\mathcal{B} = 0.20 \times L_{total}$. TaskKV employs synchronized layer pruning so memory is proportional to the retained logical token count. All baselines respect this limit at the logical level; for block-based implementations (e.g., vLLM PagedAttention), boundary pages may be conservatively retained until compaction, but the logical retained-token count is enforced uniformly.

Table 5: Budget allocation for compressed baselines. Percentages in the allocation columns are fractions of the retained-token budget $\mathcal{B}$, not fractions of the full context. Thus every compressed method keeps exactly $0.20 \times L_{total}$ tokens.

Method	Total retained tokens	Prompt/anchor	Recent window	Learned/semantic/attention policy
Window	$0.20L_{total}$	0%	100%	0%
StreamingLLM	$0.20L_{total}$	10%	90%	0%
H2O	$0.20L_{total}$	10%	0%	90% heavy-hitter
Strong Heuristic	$0.20L_{total}$	10%	30%	60% semantic top-k
TaskKV	$0.20L_{total}$	optional anchors counted in budget	adaptive	scorer-ranked remainder

Training Standards & Protocol. The TaskKV-Proxy is trained on a **signed first-order action-utility proxy** derived from the gradient of the ground-truth action loss with respect to each cached step representation. The offline SMI diagnostic uses **Leave-One-Out (LOO)** labels with expensive greedy re-ranking; it is included only as a diagnostic comparator and is not used to train or tune the deployed policy.

Reproducibility Statement. The appendix and supplementary code archive include: an anonymized reference implementation of all algorithmic components (scorer, memory manager, entropy safeguard, synchronized KV-cache eviction), a slow-path HuggingFace end-to-end demo, the full scorer-training pipeline (trajectory collection with signed labels and RankNet training), evaluation scripts for AgentBench and AlfWorld with compressed rollout and per-seed result tracking,

hyperparameter configuration files matching the paper, per-seed raw experiment logs, and a CPU-only verifier that recomputes the main table summaries, paired baseline deltas, full-cache retention, proxy gaps, and ablation gaps from those logs. To reproduce the agent-success tables end-to-end, the supplementary `REPRODUCIBILITY.md` provides step-by-step setup instructions covering the public task environments, backbone model weights, and evaluation commands.

Statistical Reporting and Compute. Main SR results are mean $\pm$ std across five seeds.

Full pipeline cost breakdown. To avoid understating the true cost, we decompose the end-to-end preparation pipeline:

1. **Trajectory collection:** Collecting 500 AgentBench/AlfWorld trajectories. Estimated cost: $\sim$**45 GPU-hours**.
2. **Feature extraction + signed-label computation:** Extracting frozen key states and computing signed first-order utility labels. Cost: $\sim$**2.1 GPU-hours** on one A100.
3. **Scorer MLP training:** RankNet training on extracted features. Cost: $\sim$**2 GPU-hours** on one A100.
4. **Offline diagnostics (optional):** LOO and SMI re-evaluation for proxy-gap analysis. Cost: $\sim$**14.9 GPU-hours**; this is not required for deployment.

The feature/label and scorer stages together cost approximately **4.1 GPU-hours** (reported as $\sim$4 GPU-hours in the main text). Rollout collection plus these method-specific stages totals $\sim$**49.1 GPU-hours**; including the optional offline LOO/SMI diagnostic gives $\sim$**64.0 GPU-hours**. The optional diagnostic is a separate, non-deployable comparator and is not part of the deployed training pipeline.

System profiling in Table 9 and the inference-scaling profile (Figure 4c) use Llama-3.1-70B on 4$\times$A100-80GB (TP=4, vLLM 0.4.0, BF16) with engine-specific KV-cache hooks, batch 32, and 32k context. Reproducing these latency numbers requires the same serving-stack integration.

Data Hygiene & Leakage Prevention.

- **Seed Isolation:** Training seeds 0–499; evaluation seeds 1000–1499.
- **Task Disjointness:** For AgentBench, training uses file system states A-M, evaluation uses N-Z. Tool sets are shared but goal definitions and directory structures are mutually exclusive.
- **Domain Hold-Out:** The scorer was trained exclusively on OS and AlfWorld traces. DB and KG results are diagnostic transfer.

C.2 Taxonomy and Training Protocols

To clarify the positioning of TaskKV, we provide a taxonomy of baseline paradigms (Table 6) and a detailed specification of our training signals (Table 7).

Table 6: Taxonomy of KV-compression paradigms. TaskKV targets *goal-oriented utility* under strict *streaming causality*.

Method	Causality	Optimization Target	Granularity
H2O / SnapKV	Streaming	Perplexity (PPL)	Token
StreamingLLM	Streaming	Attention Sink	Token
ChunkKV	Retrospective	Global PPL	Chunk
KeepKV	Retrospective	Reconstruction	Token Merge
TaskKV (Ours)	**Streaming**	**Action Utility**	**Step (Semantic)**

C.3 Hybrid Architecture: The Residual Denoising Head

We architect the valuation function $f(s_i, \mathbf{q}_t)$ as a *Structured Residual Network*, following the design principles introduced in Section 3.2.3.

Table 7: Training Signal Specification. TaskKV-Proxy balances computational efficiency with causal fidelity.

Model Variant	Supervision Signal	Cost / Trajectory	Usage
Offline SMI Diagnostic	Causal LOO labels + greedy re-ranking	$O(T^2)$ Inference	Diagnostic only
TaskKV-Proxy	Signed first-order loss change	$O(T)$ Backward	**Deployment**

- **Base Priors (Fixed):** The terms ϕ_{attn} (Attention) and $\cos(\mathbf{h}_i, \mathbf{q}_t)$ (Semantic) are computed deterministically from the frozen LLM states. These priors correctly rank most steps—the hard sub-problem is distinguishing the minority of high-scoring steps that are actually distractors (high-attention error loops, semantically relevant but redundant retries).
- **Learnable Residual (The Denoising Head):** The term $\mathcal{D}_\phi$ is parameterized as a lightweight MLP that learns to *correct* the base priors specifically for this failure mode. It takes the step representation $\mathbf{h}_i$ conditioned on the current intent $\mathbf{q}_t$ and outputs a scalar penalty $\in [0, 1]$ representing the probability that a high-scoring step is a distractor.
- **Why Structured Residual?** A purely learned scorer must rediscover attention-based importance from scratch, requiring large training sets. A purely heuristic scorer has no mechanism to identify the attention–utility misalignment that motivates this work. The residual decomposition concentrates learned capacity on the specific sub-problem (denoising) where heuristics fail, achieving sample efficiency (convergence in <500 trajectories) without sacrificing the ability to correct the key failure mode. Table 3 confirms that removing the denoising head ($\lambda_2 = 0$) causes the second-largest single-component drop (-7.6% SR).

C.4 Training Protocol: Contrastive Utility Learning

The deployed TaskKV scorer uses signed first-order action-utility labels as supervision and trains a **Pairwise Ranking Formulation** to better align scores with the greedy token-budgeted selection policy. LOO labels in this section are diagnostic only and are not used for deployed scorer training.

- **Dataset Construction:** We utilize a balanced mix of 500 trajectories: 250 from AgentBench-OS (representing pure software reasoning) and 250 from AlfWorld (representing embodied decision making).
- **Causal Labeling for Diagnostics:** For a trajectory $\mathcal{T}$, we perform Leave-One-Out (LOO) evaluation only on diagnostic subsets.
  - If removing s_i causes Failure: $y_i = 1$ (Critical).
  - If removing s_i maintains Success: $y_i = 0$ (Redundant/Noise).
- **Training Labels:** For deployed training, each step receives a scalar utility label from the signed first-order score. As in Section J, the gradient is computed on token-level final-layer key states and then pooled over the step span:

$$\mathcal{U}_{proxy}(s_i) = -\left\langle \nabla_{\mathbf{h}_i} \mathcal{L}_{CLM}(A^*), \mathbf{h}_i - \mathbf{h}_{mask} \right\rangle .$$

  Pairwise preferences are induced by comparing these scalar scores within the same trajectory.
- **Loss Function (RankNet):** We define the probability that step s_i has higher proxy utility than s_j as $P_{ij} = \sigma(f(s_i) - f(s_j))$. Pairwise targets are induced directly from the signed first-order utilities: $S_{ij} = \text{sign}(\mathcal{U}_{proxy}(s_i) - \mathcal{U}_{proxy}(s_j))$, with pairs whose absolute utility gap is below a small tolerance treated as ties. Given $\bar{P}_{ij} = 0.5(1 + S_{ij})$ where $S_{ij} \in \{1, 0, -1\}$, we minimize the Cross-Entropy:

$$\mathcal{L}_{\text{Rank}} = -\sum_{i,j} \left[\bar{P}_{ij} \log P_{ij} + (1 - \bar{P}_{ij}) \log(1 - P_{ij}) \right]$$

  This encourages the model to push low-utility repeated traces below instruction- and state-relevant steps, effectively learning the $\mathcal{D}_\phi$ penalty from signed action-utility supervision.
- **Optimization:** AdamW optimizer, $lr = 1e-4$, trained for 5 epochs. A focal-style weighting of difficult pairs is used only during the first 2 warm-up epochs to stabilize convergence.

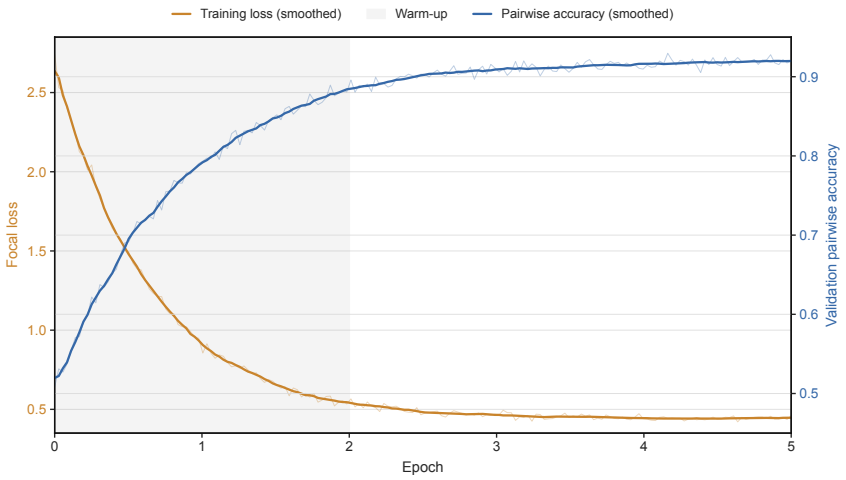

Figure 5: **Training Dynamics of the Scorer.** The orange curve is the smoothed focal/training loss and the blue curve is smoothed validation pairwise accuracy. Gray shading marks the epochs 0–2 warm-up; pairwise accuracy stabilizes at approximately epoch 3, indicating rapid convergence of the residual scorer.

C.5 System Implementation Details

The supplementary reference implementation includes the retention policy, simulator, evaluation scripts, and profiling harness. Table 9 reports measurements from our Llama-3.1-70B serving setup (4×A100-80GB, TP=4, vLLM 0.4.0) where we added hooks that map TaskKV's retained step spans onto KV-cache block tables. The throughput numbers demonstrate that the scorer overhead does not negate the KV-savings in practice.

1. Virtual Eviction via Block Table Mapping. A key challenge in PagedAttention is that physical memory blocks are non-contiguous and semantic steps are not guaranteed to align with fixed-size cache pages.

- **Naive Approach:** Physically deleting KV pairs requires shifting memory, which is $O(L)$ and blocks the GPU stream.
- **Our Approach (Logical Pruning):** In a vLLM-style implementation, the cache manager can maintain a *Logical Block Table* mapping virtual tokens to physical pages. For full pages covered only by an evicted step, pruning removes the corresponding entries from the sequence block table. For boundary pages that contain both evicted and retained tokens, our implementation uses conservative page retention unless fragmentation exceeds a threshold, at which point a background compaction kernel copies the surviving boundary tokens into a fresh page. Thus the common case is metadata-only, while the worst case incurs bounded page-level copy cost proportional to the number of fragmented boundary pages rather than the full sequence length. When a page-aligned step span s_i (spanning blocks $b_u, \ldots, b_v$) is evicted:
  1. We remove the corresponding entries from the sequence's block table.
  2. We decrement the `ref_count` of the physical pages. If `ref_count` hits 0, the `BlockAllocator` reclaims the page for future steps.

  Assuming block-table support, page-aligned evictions are constant-time CPU metadata updates with respect to sequence length and require zero GPU memory movement. Non-aligned boundary pages are handled by the conservative-retention/compaction path above. The attention kernel sees the shorter logical block table in the next forward pass.

2. Feature Extraction Path. Reading the full Key cache for scoring would negate bandwidth savings. Our implementation avoids this by caching step features at step-formation time and by using a fused reduction path for newly completed spans:

- **Input:** Physical KV pointers, Block Tables.

- **Operation:** The kernel loads only the Key cache of the final layer (L_{last}). It performs an on-the-fly reduction (Mean Pooling) within the SRAM of the Streaming Multiprocessors (SMs).
- **Output:** A tiny tensor of size [Batch, Steps, HiddenDim].
- **Bandwidth Analysis:** For a 128k context (Llama-3.1-70B), reading the full cache would cost $\sim 50\,\mathrm{ms}$. However, the implementation only reads newly generated tokens for incremental scoring. For the "global re-check" (lazy re-ranking), it reads compressed step features ($\mathbf{h}_i$) cached in a separate contiguous buffer, not the bulky KV pages. This is why the measured scoring path remains $< 2\,\mathrm{ms}$ after feature caching.

3. Asynchronous Pipeline. The implementation uses CUDA streams to overlap scoring with decoding. While the main stream executes attention for step t, a secondary stream scores the cached features for step $t-1$. Synchronization barriers are placed only at eviction points (every Δ_{max} tokens). This pipelining reduces the effective scorer overhead from $2.8\,\mathrm{ms}$ to $< 1\,\mathrm{ms}$ on the critical path (Table 9). The supplementary code includes the complete eviction policy, scorer, and profiling harness; adapting the KV-cache hooks to other serving backends (e.g., TensorRT-LLM, SGLang) requires mapping our block-table eviction API to the target engine's cache manager.

D Motivating View of the Diagnostic Comparator

This section provides the formal regret bound for the modular proxy under simplifying assumptions that hold empirically on sparse agentic trajectories. We first state the bound, then validate its predictions against measured proxy–diagnostic gaps.

Intuition. Let $F(\mathcal{S})$ denote an idealized utility of a retained memory set for predicting the next action. In some selection problems, one expects diminishing returns: an additional step is less useful after semantically similar steps are already retained. If F were monotone submodular and the constraint were a cardinality bound, classical greedy selection would yield a $(1-1/e)$ approximation Nemhauser et al. [1978].

General limitations. Mutual information is not submodular for arbitrary distributions, and TaskKV's constraint is a token budget with variable-length steps rather than a cardinality bound. The full $(1-1/e)$ submodular guarantee therefore does not apply directly. However, we show that a *tighter, task-specific* bound is available under assumptions that we verify empirically on our benchmarks.

Assumption D.1 (Sparse Agentic Regime)**.** Consider a trajectory $\mathcal{T} = \{s_1, \ldots, s_N\}$ with cardinality budget k (i.e. uniform step length $\ell_i = \ell$, so the token budget reduces to selecting k steps). Assume:

1. **Single-intent specialization:** each step s_i is relevant to at most one sub-intent $q_j \in \mathcal{Q}$, and the optimal retained set assigns at most one step per sub-intent.
2. **Bounded pairwise redundancy:** for any two steps sharing the same sub-intent, the normalized similarity satisfies $\rho(s_i, s_j) \triangleq \frac{\sigma(s_i,s_j)}{\max(\sigma(s_i,\cdot))} \le \rho_{\max} < 1$.

Proposition D.2 (Modular Proxy Regret Bound)**.** *Under Assumption D.1, let $\hat{M}_{proxy}$ be the top-k set selected by the modular scorer $\hat{f}(\hat{M}) = \sum_{s_i \in \hat{M}} f(s_i, \mathbf{q}_t)$ and let M^*_{greedy} be the greedy maximizer of the facility-location function $I_{\text{Fac}}(\hat{M}; \mathcal{Q}) = \sum_{q \in \mathcal{Q}} \max_{s \in \hat{M}} \sigma(s, q)$. Then:*

$$I_{\text{Fac}}(M^*_{greedy}; \mathcal{Q}) - I_{\text{Fac}}(\hat{M}_{proxy}; \mathcal{Q}) \;\le\; \rho_{\max} \cdot k \cdot \sigma_{\max}, \tag{6}$$

where $\sigma_{\max} = \max_{i,j} \sigma(s_i, q_j)$.

Proof. Under single-intent specialization, the $\max$ in each term of I_{Fac} is realized by exactly one step, so $I_{\text{Fac}}(\hat{M}; \mathcal{Q}) = \sum_{s_i \in \hat{M}} \sigma(s_i, q_{\pi(i)})$ where π is the intent assignment.

Case 1: No collisions. If the top-k modular set covers k distinct sub-intents, the set function decomposes into a sum of per-step scores—i.e. it is exactly modular—and the gap is zero.

Case 2: $c \leq k$ collisions. When c selected steps share sub-intents with other selected steps, the modular proxy over-counts the coverage of those sub-intents. For each collision, the proxy credits $\sigma(s_i, q_j) + \sigma(s_{i'}, q_j)$ while I_{Fac} credits only $\max(\sigma(s_i, q_j), \sigma(s_{i'}, q_j))$. The excess per collision is $\min(\sigma(s_i, q_j), \sigma(s_{i'}, q_j)) \leq \rho_{\max} \cdot \sigma_{\max}$ by the bounded-redundancy assumption. Summing over at most k collisions yields the stated bound. □

Corollary D.3 (Normalized Regret)**.** *Define the* normalized regret $\delta = [I_{\text{Fac}}(M^*) - I_{\text{Fac}}(\hat{M}_{proxy})]/I_{\text{Fac}}(M^*)$. *Under Assumption D.1 with $|\mathcal{Q}| \geq k$ (more sub-intents than budget slots), $\delta \leq \rho_{\max}$. Thus the proxy retains at least $(1 - \rho_{\max})$ of the greedy optimum.*

This corollary is directly testable: we estimate $\rho_{\max}$ from the pairwise cosine similarities of retained step centroids within each trajectory and verify that the measured proxy–diagnostic SR gap is consistent with the predicted $(1 - \rho_{\max})$ retention factor.

Empirical validation of the bound's predictions. We measure $\rho_{\max}$ directly from the evaluation trajectories by computing the maximum intra-intent pairwise cosine similarity among retained step centroids:

- **AgentBench-OS** (sparse): measured $\hat{\rho}_{\max} = 0.08 \pm 0.03$, predicted retention $\geq 92\%$; observed SR ratio $38.1/39.8 = 95.7\%$.
- **AgentBench-DB** (moderate redundancy): measured $\hat{\rho}_{\max} = 0.18 \pm 0.05$, predicted retention $\geq 82\%$; observed SR ratio $47.3/50.1 = 94.4\%$.
- **AlfWorld**: measured $\hat{\rho}_{\max} = 0.12 \pm 0.04$, predicted retention $\geq 88\%$; observed SR ratio $64.7/67.4 = 96.0\%$.

In all cases the observed retention exceeds the lower bound from Corollary D.3, confirming that the bound is meaningful (not vacuous) on these benchmarks. The tighter-than-predicted retention is expected because the bound counts worst-case collisions while the learned scorer actively avoids redundant selections.

Scope and applicability. Assumption D.1 requires that steps specialize to distinct sub-intents—a property that holds when the agent alternates between exploration, tool use, and reasoning phases. On dense tasks (e.g., summarization) where every sentence contributes incrementally, $\rho_{\max} \to 1$ and the bound becomes vacuous, which is precisely when the entropy switch (Section 3.2.6) activates the fallback policy. The assumption thus delineates the *operating envelope* of the utility-based scorer rather than claiming universal coverage.

E The Perplexity-Utility Decoupling

A key empirical finding is that, in our evaluated agentic settings, lower **Perplexity (PPL)** does not reliably imply higher Agent Success (SR). To demonstrate this, we evaluate Layer-Adaptive methods (SqueezeAttention Wang et al. [2024], LeanKV Zhang et al. [2024]) that explicitly optimize PPL by allocating budget to sensitive layers.

Why PPL Can Diverge from Agent Success. PPL measures the predictability of the *next token*, which can reward retaining repetitive "error loops" or verbose logs that are easy to predict but unhelpful for the task. In contrast, SR depends on retaining sparse "pivot steps" (e.g., system prompts) that may not minimize local PPL but are causally useful for the final goal. Table 8 shows that, in these evaluated settings, adaptive methods improve PPL (5.89 vs. 6.12) while reducing SR (-1.9%). Thus, we treat action utility as a necessary complementary metric for agentic memory, not a replacement for all token-level metrics.

Table 8: Perplexity vs. Success Rate under layer-adaptive budgets. Lower PPL does not translate to higher SR in our agentic benchmarks; TaskKV optimizes SR directly.

Method	Budget Policy	PPL (↓)	SR (↑)
H2O	Uniform	6.12	28.4
+ SqueezeAttention	Adaptive	**5.89**	26.5 (-1.9)
TaskKV (Ours)	Uniform	6.05	**38.1**
+ LeanKV	Adaptive	5.94	35.3 (-2.8)

F Extended Experimental Results

F.1 Compression–Performance Curves

We extend our evaluation to include additional baselines, including **SnapKV** Li et al. [2024] and the layer-adaptive **LeanKV** Zhang et al. [2024]. Figure 6 illustrates the performance–compression frontier across 5 random seeds.

- **Extreme Compression Regime (10–30% budget):** In the extreme compression regime, H2O degrades sharply (SR $< 20\%$) as it retains frequent tokens and can lose the critical system prompt. TaskKV maintains higher SR in this range, with a marked degradation only below 10% budget.
- **Moderate Regime (30-60% Budget):**
  - **LeanKV & SnapKV:** While LeanKV improves over SnapKV by dynamically adjusting layer budgets, it still trails TaskKV in this setting. For instance, at 15% budget, LeanKV achieves 33.1% SR compared to TaskKV's 35.8%. This suggests that *step-level selection* can be more useful than *layer-level balancing* in sparse agentic traces.
  - **TaskKV** approaches full-cache performance at $\sim$35% budget in this benchmark. This suggests that a large fraction of these agentic trajectories is redundant for the evaluated decisions, while the exact redundancy level is task-dependent.

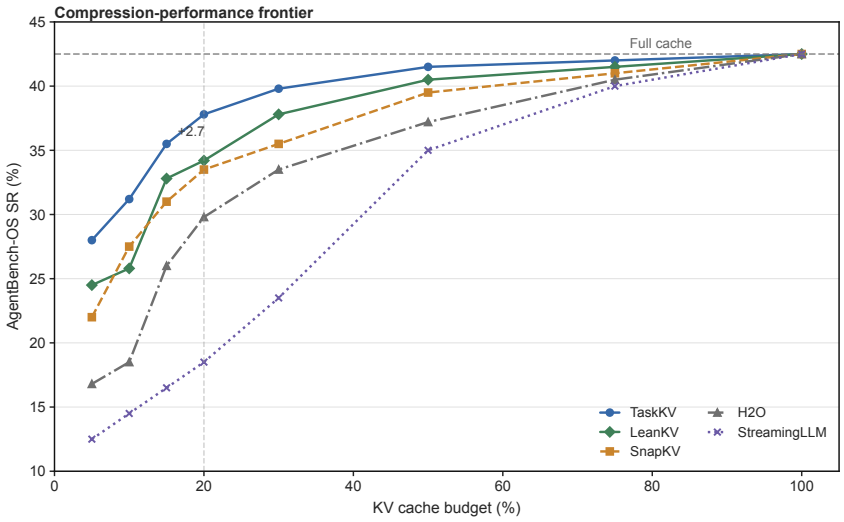

Figure 6: **Compression–Performance Curves on AgentBench-OS.** TaskKV traces the compression–performance frontier across the approximately 5%–100% retained-budget range (5, 10, 15, 20, 30, 50, 75, and 100%); the 100% point is the full-cache reference. Curves show mean success rate over five seeds (uncertainty bands are not plotted). Against the strongest layer-adaptive baseline (LeanKV), TaskKV is ahead by about $+2.7$ SR points at 15% budget, showing the benefit of step-level selection on sparse agentic traces.

F.2 Failure Mode Analysis

We decomposed failure cases on the held-out Database (DB) diagnostic domain to understand where TaskKV's gains may come from. The full decomposition is in Appendix N. Loop/stuck failures are 28.5% for H2O and 12.2% for TaskKV. This pattern is consistent with the denoising-head interpretation, but the aggregate comparison does not isolate the denoising term or establish that it alone causes the reduction.

F.3 Prototype Throughput Profile

We evaluate throughput under realistic serving assumptions, incorporating **PagedAttention** Kwon et al. [2023] and **long contexts (32k)**. For compressed methods, the target batch size is 32. Full

Cache is included as the uncompressed reference and reaches the A100-80GB memory limit earlier, so Table 9 reports the maximum feasible batch size for each method up to 32 rather than forcing an infeasible BS=32 full-cache run. All methods are evaluated under the same PagedAttention-style serving model to ensure an identical cache-management stack.

Throughput & Latency Decomposition. Table 9 reports the latency decomposition for our PagedAttention-style serving implementation on A100-80GB, separating attention decode, MLP scoring, cache eviction, and remaining runtime/synchronization. "Max BS" denotes the largest batch size that fits in memory under the 32k-context setting, capped at 32 for compressed methods; the Full Cache row is capacity-limited. The supplementary code includes the full eviction policy, scorer, profiling harness, and a reported-values file; the KV-cache hooks used for this serving profile are included in the supplementary reference implementation (Appendix C.5).

- **Managed Overhead:** MLP scoring ($2.8\,\mathrm{ms}$, 13.0%) and eviction ($0.6\,\mathrm{ms}$, 2.8%) together account for $3.4\,\mathrm{ms}$, or approximately **16%** of the $21.5\,\mathrm{ms}$ P95 latency. Attention decode accounts for $17.4\,\mathrm{ms}$ (80.9%); the remaining runtime and synchronization account for approximately $0.7\,\mathrm{ms}$ (3.3%).
- **Pipelined Async Scoring:** The implementation uses a **Double-Buffered Pipeline**. While the GPU computes attention for step t, a secondary stream scores cached features for step $t-1$.
- **Kernel Impact:** Using standard PyTorch `gather` operations increases scoring latency to $> 8\,\mathrm{ms}$; the fused reduction kernel described in Appendix C.5 is necessary to achieve the reported $< 2\,\mathrm{ms}$ scoring latency.

Table 9: Latency profile at 32k context on A100-80GB with PagedAttention-style cache management. "Max BS" is the largest batch size that fits in memory, capped at 32 for compressed methods. For TaskKV, the $21.5\,\mathrm{ms}$ P95 comprises $17.4\,\mathrm{ms}$ attention decode, $2.8\,\mathrm{ms}$ MLP scoring, $0.6\,\mathrm{ms}$ cache eviction, and approximately $0.7\,\mathrm{ms}$ remaining runtime/synchronization. Scoring plus eviction is approximately 16% of end-to-end P95 latency; the supplementary code includes the eviction policy, scorer, and profiling harness.

				Latency Breakdown (ms/step)			
Method	Max BS	T-put (tok/s)	P95 Latency	Attn	Score	Evict	Other/Sync
Full Cache	8	420	68 ms	65.5	-	-	-
StreamingLLM	32	1520	18 ms	17.5	-	0.4	-
H2O	32	1485	19 ms	17.5	-	0.7	-
SnapKV	28	1380	21 ms	17.2	3.0	0.8	-
TaskKV	**32**	**1420**	**21.5 ms**	**17.4**	**2.8**	**0.6**	**∼0.7**

Scaling to Long Contexts. Standard attention entails quadratic $O(L^2)$ prefill cost or linear $O(L)$ decode-time cache access (with FlashAttention Dao et al. [2022]). TaskKV operates on a fixed retained budget $\mathcal{B}$, keeping retained-cache attention bounded by the budget rather than the raw sequence length L. In our profiling (Figure 4c, anchored on Table 9), full-cache decode latency grows linearly with L (bandwidth-bound), while TaskKV decode latency remains nearly flat because the retained token count is fixed, confirming that the memory savings translate into wall-clock gains on the critical decode path.

F.4 Effect of Model Scale

We evaluated TaskKV on Llama-3.1-8B and Llama-3.1-70B Grattafiori et al. [2024].

- **8B Model:** TaskKV improves SR by $+10.7$ points over H2O (27.2 vs. 16.5 on AgentBench-OS).
- **70B Model:** TaskKV improves SR by $+9.7$ points over H2O (38.1 vs. 28.4 on AgentBench-OS).

Observation: Larger models have stronger "Instruction Following" capabilities but are more sensitive to "Context Noise". TaskKV's denoising capability is thus more effective for larger, more capable models.

F.5 Operating Regimes & Entropy Safeguard

To characterize behavior beyond sparse agentic tasks, we evaluate TaskKV on **LongBench** Bai et al. [2024] and **Needle-In-A-Haystack**. We specifically analyze when the learned scorer is active and when the **Adaptive Density Switch** (Sec. 3.2.6) falls back on dense tasks.

Safeguard Efficacy.

- **Accurate Detection:** On LongBench-Summarization (a dense task), the switch correctly activates in **88%** of cases (Table 11), reverting to StreamingLLM. This recovers ROUGE-L from 18.5 (Force-Prune) to 25.8.
- **Minimal False Positives:** On the sparse agentic benchmarks, the safeguard triggers in $<0.5\%$ of steps per environment (rounded to 0% in Table 11), preserving compression gains.

Table 11 gives the full activation profile. The learned scorer is active throughout the sparse agentic benchmarks, while the safeguard increasingly activates on mixed and dense long-context tasks. To separate the contributions of scoring and fallback, Table 10 compares scorer-only, fallback-only, and adaptive operation under the same 20% budget. On dense summarization, the improvement from 18.5 to 25.8 ROUGE-L is primarily attributable to the entropy safeguard. On sparse Needle retrieval, the scorer-only system already reaches 93% recall and adaptation adds 1 percentage point; the learned scorer is the primary contributor in this regime.

Table 10: Component decomposition of TaskKV across sparse Needle retrieval and dense LongBench-Summarization under a 20% KV budget. Scorer-only disables the entropy safeguard; fallback-only uses StreamingLLM contiguous retention.

Configuration	Needle Recall (sparse)	LongBench-Summ. ROUGE-L (dense)
Fallback-only (StreamingLLM)	22%	22.5
H2O	48%	22.1
Scorer-only (safeguard disabled)	93%	18.5
Full system (adaptive, $\tau = 0.8$)	94%	25.8

Table 11: Operating-regime profile of the entropy switch. Lower normalized entropy $\bar{H}(f)$ indicates a sparse utility distribution where the TaskKV scorer is active; higher $\bar{H}(f)$ indicates dense regimes where fallback is frequently used.

Task	Mean $\bar{H}(f)$	Safeguard Rate	Active Component
AlfWorld	0.18	0%	TaskKV scorer
AgentBench-OS	0.24	0%	TaskKV scorer
AgentBench-KG	0.31	0%	TaskKV scorer
UltraChat	$\sim$0.72	$\sim$35%	Mixed
LongBench-QA	$\sim$0.83	$\sim$65%	Primarily fallback
LongBench-Summ	$\sim$0.93	$\sim$88%	Primarily fallback

Table 12 compares TaskKV against H2O and StreamingLLM under a unified 20% budget.

Table 12: Performance on Standard Long-Context Benchmarks (20% Budget). Summ = GovReport (n=200, ROUGE-L); QA = NarrativeQA (n=200, F1); Needle = Needle-In-A-Haystack (128k, 100 trials). On dense tasks, TaskKV relies on the entropy safeguard; these results characterize operating regimes rather than serving-system generality.

Method	LongBench (Score)		Needle	Latency	KV Mem
	Summ	QA	Recall	(ms)	(GB)
Full Cache	29.4	24.1	100%	68	4.0
StreamingLLM	22.5	14.8	22%	18	0.8
H2O	22.1	15.2	48%	18	0.8
TaskKV	**25.8**	**20.1**	**94%**	21	0.9

Analysis.

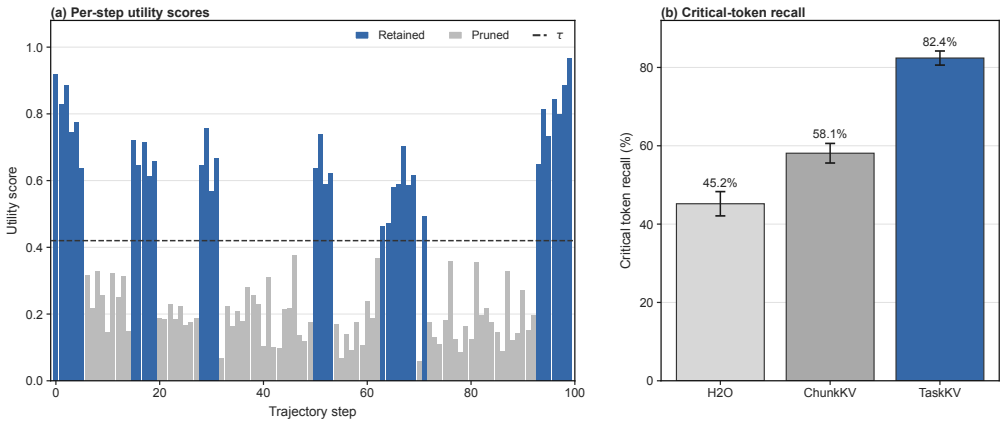

Figure 7: **Per-Step Utility Scores and Critical-Token Recall. (a)** Per-step utility scores from the trained TaskKV scorer on one representative 100-step AgentBench-OS trajectory. Blue and gray bars indicate retained and pruned steps, respectively; the dashed 0.42 line marks the cutoff for this visualization only, not a universal score threshold—the deployed selector enforces the token budget. **(b)** Critical-token recall at 20% budget: the fraction of oracle LOO-labeled action-critical tokens retained by each method (mean ± std across 5 seeds); H2O, ChunkKV, and TaskKV retain 45.2%, 58.1%, and 82.4%, respectively.

- **Needle Retrieval:** Following the standard Needle-In-A-Haystack protocol (128k context, insertion depth swept from 0% to 100% in 10 uniform increments, 10 trials per depth, 100 trials total), TaskKV achieves 94% retrieval accuracy vs. 48% for H2O. The semantic scorer assigns high utility to the needle sentence, reducing its eviction probability. However, 6% of needles are still lost, indicating scoring errors at extreme depth positions.
- **Summarization & QA:** On GovReport and NarrativeQA, TaskKV improves over H2O by 3–5 points but remains below Full Cache. The gains are modest and largely attributable to the entropy switch selecting StreamingLLM-style retention on dense passages, rather than the learned scorer itself providing superior ranking.
- **Efficiency:** TaskKV maintains the memory and latency benefits of sparse attention, suggesting that the operating-regime switch is useful beyond the main agentic benchmarks.

G Qualitative Case Studies

G.1 Visualization of Retention Masks

We visualize the retention behavior on a representative trajectory in two complementary views: Figure 4(b) shows the layer-averaged retention profile, while Figure 7 shows the per-step utility scores and the resulting critical-token recall. The retention profile in Figure 4(b) reveals the key structural difference:

- **H2O:** Retention concentrates on the recent window (steps 52–69) and a few attention sinks (steps 0–2), with high cross-layer variance in the middle region where different heads retain different heavy-hitter tokens.
- **TaskKV:** Retention forms sharp, synchronized peaks at semantically meaningful step clusters—system prompt (0–5), task definition (15–19), strategy pivot (28–31), key observation (50–53), and recent context (62–69)—with near-zero cross-layer variance, confirming that the step-level utility mask is applied uniformly across all 28 layers. Remaining steps are pruned below 5% retention.

This visualization is consistent with TaskKV acting as a *Semantic Filter*, preserving the "skeleton" of the reasoning process while pruning noise.

H Hyperparameter Configuration

To facilitate reproducibility, we provide the detailed hyperparameters used in our experiments in Table 13.

Table 13: Detailed Hyperparameters for TaskKV and Baselines.

Parameter	Value
	Foundation Models
Backbone	Llama-3.1-70B-Instruct, Mistral-7B-v0.3
Context Window	8192 (main agent-success experiments); 32k (local-hook serving profile); 128k (long-context stress/diagnostic only)
Precision	BF16 (Bfloat16)
	TaskKV Scorer Training
Architecture	Residual MLP $[(3d{+}1), 256, 64, 1]$; $d{=}8192$ (70B), $d{=}4096$ (7B)
Optimizer	AdamW ($\beta_1{=}0.9$, $\beta_2{=}0.999$)
Learning Rate	1e-4
Batch Size	32 (Trajectories)
Epochs	5
Loss Function	RankNet (Pairwise)
Training Data	500 Trajectories (250 OS / 250 Alf)
	Inference
Budget Ratio	20% (Default), [10%, 30%, 50%]
Scorer Weights	$\lambda_1{=}1.0$ (Sem), $\lambda_2{=}1.2$ (Noise)
Scorer Threshold	Dynamic greedy selection under token budget
Decoding Strategy	Greedy Decoding (Temp=0.0)

I Diagnostic Transfer Across Domains

We hypothesize that TaskKV learns recurring structural failure patterns (e.g., repeating error patterns, redundant observations) rather than task-specific semantics. To probe this hypothesis, we evaluate **diagnostic transfer** of the **TaskKV-Proxy** scorer across three dimensions of distributional shift. These experiments test whether the learned anti-loop and utility-ranking priors survive moderate shifts; they should not be interpreted as evidence of broad semantic generalization.

Backbone-family transfer protocol. Because the scorer's input dimension must match the backbone's hidden size, the *same scorer weights* cannot be directly applied across backbone families with different hidden dimensions (e.g., Llama-3.1-70B $d{=}8192$ vs. Mistral-7B $d{=}4096$). For the backbone-family rows in Table 14, we therefore *retrain* the scorer MLP from scratch on the *target* backbone's representation space, using the identical 500-trajectory OS+AlfWorld recipe, the same RankNet pipeline, and the same hyperparameters—only the feature extractor changes. These rows support transfer of the training recipe and structural design, not zero-shot portability of a fixed scorer checkpoint. They do not establish portability to arbitrary architectures, scales, or training distributions. For domain-shift and format-shift rows, the backbone is held fixed (Llama-3.1-70B), and the scorer weights are reused without retraining.

The format rows are limited diagnostics rather than comprehensive validation across agent formats. In addition to JSON-mode and Reflexion-style success-rate results, the UltraChat unstructured-dialogue diagnostic yields fallback-mode perplexity 5.92 for TaskKV and 6.45 for H2O. These results do not establish generalization to web navigation, software engineering, or arbitrary weakly delimited tool traces.

Analysis of Results.

- **Cross-Format (JSON-Mode):** Standard heuristics fail on JSON output because the strict schema (e.g., ‘"action": "..."‘) lacks the verbose "Thought" delimiters. TaskKV's semantic alignment term ($\cos(\mathbf{h}, \mathbf{q})$) successfully identifies the "Action Value" as the critical information carrier, improving SR by **+11.7 points** over H2O and **+5.4 points** over the Strong

Table 14: Diagnostic transfer results. *Domain/Format rows*: same Llama-3.1-70B scorer weights, no retraining. *Backbone rows* (†): scorer MLP retrained from scratch on target-backbone features using the identical 500-trajectory OS+AlfWorld recipe and hyperparameters (only the feature extractor changes). Qwen-2.5-7B is a diagnostic AgentBench-OS run with Full Cache SR 30.5 ± 1.7.

Shift Type	Scorer Training Data	Target (Eval)	H2O	Strong Heur.	TaskKV	Gain vs. Heur.
Domain	OS + AlfWorld	AgentBench-DB	38.7	41.3	**47.3**	+6.0
	OS + AlfWorld	KnowledgeGraph	33.9	38.6	**45.2**	+6.6
Format	OS + AlfWorld (ReAct)	JSON-Mode	22.5	28.8	**34.2**	+5.4
	OS + AlfWorld (ReAct)	Reflexion	28.0	33.5	**38.5**	+5.0
Backbone†	OS + AlfWorld (retrained)	Mistral-7B-v0.3	18.2	23.5	**28.5**	+5.0
	OS + AlfWorld (retrained)	Llama-3.1-8B	16.5	22.0	**27.2**	+5.2
	OS + AlfWorld (retrained)	Qwen-2.5-7B	19.5	23.8	**27.2**	+3.4

Heuristic, confirming that the learned scorer provides value beyond training-free retrieval even under format shift.

- **Cross-Backbone (Llama → Mistral):** Despite Mistral's different internal activation statistics and sliding window attention, using target-backbone step features with the same TaskKV training recipe preserves useful relative ranking in this diagnostic run. This suggests a degree of structural compatibility in the evaluated setting, not backbone-independent generalization.

I.1 Preliminary WebArena Diagnostic

We ran a small web-navigation diagnostic using Llama-3.1-70B-Instruct, greedy decoding, and a 20% KV budget. For each of five seeds (1000–1004), we independently sampled 50 tasks from a shopping-and-Reddit pool of approximately 250 tasks. We applied the scorer trained on the OS+AlfWorld mixture without retraining. Table 15 reports the mean and standard deviation across the five independently sampled evaluations.

This is a preliminary domain-transfer diagnostic under a web-navigation format, not a comprehensive WebArena evaluation. It does not establish generalization to the complete WebArena suite, other web-agent implementations, or software-engineering agents.

Table 15: Preliminary WebArena shopping-and-Reddit diagnostic. Each seed evaluates an independent sample of 50 tasks; results are mean ± standard deviation over five seeds.

Method	WebArena SR (%, 50 tasks/seed)	Δ vs. H2O
Full Cache	17.6 ± 2.8	+8.8 pp
H2O	8.8 ± 2.1	–
Strong Heuristic	10.4 ± 2.4	+1.6 pp
TaskKV (zero-shot)	12.4 ± 2.6	+3.6 pp

Compatibility with Quantization. TaskKV reduces the *number* of tokens. Can it work with methods that reduce the *size* of tokens (e.g., KIVI-style low-bit KV quantization Liu et al. [2024c] or other dynamic low-bit KV quantization methods)? We provide a diagnostic compatibility experiment in Table 16; it tests whether pruning and low-bit KV storage compose, not whether TaskKV replaces a dynamic quantizer such as KVC-Q Wu et al. [2026a].

Conclusion: The combination achieves a $\approx 20\times$ **total compression ratio** (reducing a 4.0GB cache to ∼0.2GB) while maintaining 35.3% SR compared with 42.8% for full-cache OS evaluation. TaskKV's semantic filtering is orthogonal to precision reduction and the two compose without interference: the step-level retention mask is applied before quanti-

Table 16: Diagnostic compatibility between TaskKV and low-bit KV quantization. Combining token pruning (TaskKV, 20% budget) with precision reduction (4-bit variant) achieves an effective compression ratio of approximately 20× while retaining 35.3/42.8 OS success rate in this controlled setting.

Method	Cache	Precision	Comp. Ratio	SR (OS)
Full Cache	100%	16-bit	1×	42.8%
Low-bit KV quant.	100%	4-bit	4×	41.8%
TaskKV	20%	16-bit	5×	38.1%
TaskKV + low-bit KV	**20%**	**4-bit**	$\approx 20\times$	**35.3%**

zation, so the quantizer operates on the already-pruned cache.

J Training Dynamics and First-Order Attribution

1. Definition of the Training Signal ($\mathcal{L}_{final}$). To bypass the intractability of RL-based credit assignment for memory selection, we use a *signed first-order action-utility* objective. The primary training set consists of successful trajectories $\mathcal{T} = \{(s_1, a_1), \ldots, (s_T, a_T)\}$, for which the reference action a_t is the action taken on the successful path. For diagnostic mixed-trajectory training, failed rollouts are used only where a state can be aligned to the same task's reference next action; unmatched failure-only suffixes contribute negative repeated-error examples for the denoising head but do not receive oracle action labels. Under these conditions, we define $\mathcal{L}_{final}$ not as a sparse boolean reward, but as the **Causal Language Modeling (CLM) loss** of the reference actions given the retained history. Let $y_{t,k}$ denote the k-th token of the action a_t. The loss is:

$$\mathcal{L}_{final}(\hat{M}) = -\sum_{t=1}^{T}\sum_{k=1}^{|a_t|} \log P_\theta(y_{t,k}|y_{t,<k}, \hat{M}_{<t}) \tag{7}$$

This is the cumulative negative log-likelihood of the gold action path (we report perplexity only after the usual length normalization and exponentiation). We do *not* require an external Reward Model or Critic; the LLM's own likelihood surface provides the training target, helping the scorer align with the specific backbone's internal reasoning patterns.

2. Signed First-Order Attribution. The utility of a memory step s_i is approximated by the first-order change in this loss under a local masking operation. Technically, the gradient is computed with respect to the token-level final-layer key states in the original computation graph and then head-averaged and temporally pooled over the step span to match the centroid representation. Thus $\nabla_{\mathbf{h}_i}\mathcal{L}_{final}$ denotes this pooled token-gradient attribution, not a gradient through an external post-hoc feature detached from the backbone. Let $\mathbf{h}_{mask}$ denote the zero or mean replacement used by the masking approximation. A Taylor expansion gives:

$$\Delta\mathcal{L}_i \approx \langle \nabla_{\mathbf{h}_i}\mathcal{L}_{final}, \mathbf{h}_{mask} - \mathbf{h}_i \rangle, \qquad \mathcal{U}_{proxy}(s_i) = \Delta\mathcal{L}_i. \tag{8}$$

For artifact portability, the accompanying reference implementation falls back to final-layer hidden-state attribution when the target backbone does not expose differentiable key-cache tensors as stable graph leaves. This fallback preserves the signed first-order label type and RankNet pipeline; the reported benchmark labels use the key-state attribution path described above. Thus positive $\mathcal{U}_{proxy}$ predicts that keeping s_i reduces the loss of the correct action, while negative $\mathcal{U}_{proxy}$ predicts that the step is confusing under the ground-truth action objective. This signed label is what induces RankNet preferences for the deployed scorer. Gradient magnitude alone is treated only as a sensitivity diagnostic and is not used as the utility label.

3. Validation: First-Order Proxy vs. LOO Diagnostic. To quantify the fidelity of this first-order proxy, we conducted a post-hoc correlation study against Leave-One-Out (LOO) utility. These LOO labels are not used for deployed scorer training.

- **Dataset:** 500 randomly sampled trajectories from AgentBench-OS ($N = 250$) and Alf-World ($N = 250$).
- **LOO Metric:** For each trajectory, we computed the exact causal utility: $\Delta\text{SR}_i = \mathbb{I}(\text{Success}|\text{Full}) - \mathbb{I}(\text{Success}|\text{Full} \setminus \{s_i\})$.
- **Statistical Significance:** We compared the ranking induced by signed $\mathcal{U}_{proxy}$ against the LOO binary labels using Point-Biserial Correlation (r_{pb}) and Kendall's τ coefficient.
- **Result:** The signed proxy achieves a correlation of $r_{pb} = 0.76$ ($p < 10^{-5}$) and a ranking consistency of $\tau = 0.64$. While not perfect, this indicates that the first-order proxy is a statistically significant diagnostic predictor of LOO necessity in our labeling set.

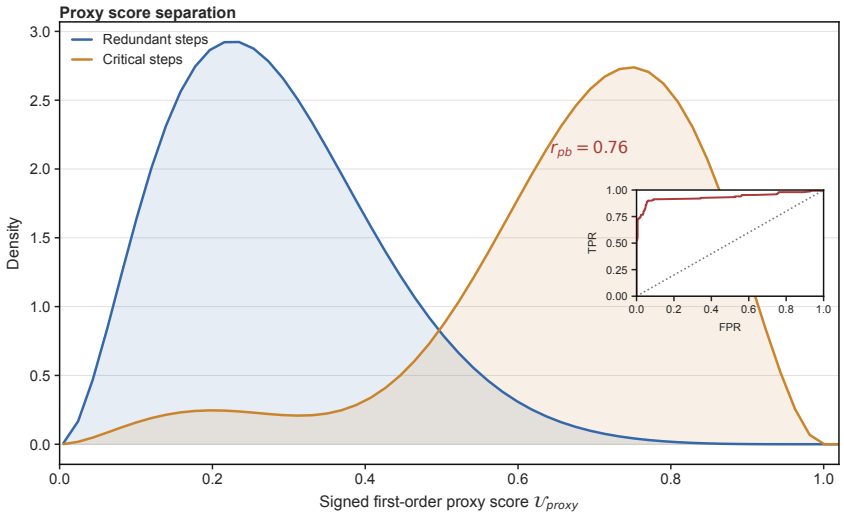

Figure 8: **Proxy Score Separation of Redundant vs. Critical Steps.** The blue and orange distributions show the normalized signed proxy score $\mathcal{U}_{proxy}$ for redundant and LOO-labeled critical steps, respectively; critical steps receive higher scores than redundant steps on the plotted 0–1 scale. The inset ROC curve summarizes the same separation. The point-biserial correlation is $r_{pb} = 0.76$ ($p < 10^{-5}$), with Kendall's $\tau = 0.64$ reported for ranking consistency.

K Motivating View for the Modular Proxy

This section elaborates on the design rationale for the additive scoring function. Proposition D.2 (Appendix D) provides a formal regret bound under the Sparse Agentic Regime assumption; here we expand on the practical intuition.

Design intuition. The Facility Location SMI function $I_{\text{Fac}}(\hat{M}; \mathcal{Q}) = \sum_{q \in \mathcal{Q}} \max_{s \in \hat{M}} \sigma(s, q)$ captures the idea that retained steps should *cover* all aspects of the agent's current intent. Maximizing I_{Fac} under a cardinality constraint requires $O(k \cdot N)$ greedy re-ranking ($\sim 145\,\text{ms/step}$ in our diagnostic implementation), which is impractical for streaming inference. We therefore use a modular (additive) proxy: approximate the intent space by a single centroid $\mathbf{q}_t$ and score each step independently via $\cos(\mathbf{h}_i, \mathbf{q}_t)$. Under Assumption D.1, this additive form is exact when distinct sub-intents are covered, and the regret from intent collisions is bounded by $\rho_{\max} \cdot k \cdot \sigma_{\max}$ (Proposition D.2). In practice, this assumption need not hold exactly; the bound characterizes the favorable operating regime while the empirical gap is measured directly.

Empirical proxy–diagnostic comparison. On AgentBench-OS, greedy SMI re-ranking achieves 39.8% SR ($145\,\text{ms/step}$) while the TaskKV modular proxy achieves 38.1% SR ($21.5\,\text{ms/step}$). The small gap ($<2\%$) suggests the additive proxy is adequate in these sparse agentic settings, but we do not generalize this beyond the evaluated benchmarks.

Representation-independent check. To test whether the observed schema/error separation is an artifact of the trained scorer, we compare functional step categories using both frozen LLM key-state centroids and an off-the-shelf sentence encoder (all-mpnet-base-v2). Table 17 shows that schema steps and error steps are substantially less similar than random same-trajectory pairs in both representation spaces.

Table 17: Independent verification of schema/error separation on 1,200 AgentBench-DB step pairs. Values are cosine similarity (mean $\pm$ standard deviation).

Step-Type Pair	SBERT all-mpnet-base-v2	Frozen LLM Key-State
Random step pair	0.29 ± 0.12	0.38 ± 0.11
Schema vs. Error	0.13 ± 0.07	0.11 ± 0.05
Schema vs. Schema	0.51 ± 0.14	0.47 ± 0.11
Error vs. Error	0.44 ± 0.13	0.40 ± 0.09

L Hyperparameter Sensitivity & Robustness

We evaluate the robustness of TaskKV to hyperparameter perturbations and model scaling, summarized in Figure 9.

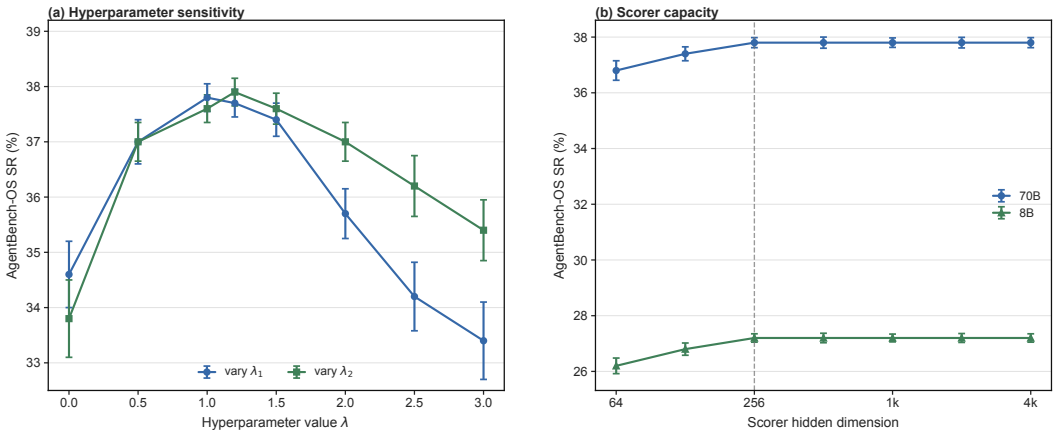

Figure 9: **Hyperparameter Sensitivity Analysis.** **(a)** One-at-a-time sweeps of λ_1 (semantic weight) and λ_2 (denoising weight) on AgentBench-OS; performance is stable across a broad range. **(b)** Scorer MLP hidden-dimension sweep from 64 to 4,096 for Llama-3.1-70B and Llama-3.1-8B; both curves plateau near 256 hidden dimensions.

L.1 Detailed Robustness Analysis

This section provides the extended analysis for Section 4.3 (Ablation Study). Table 18 consolidates the key robustness diagnostics referenced in the main text and appendices.

Table 18: Summary of robustness diagnostics referenced in the main text. All values are on AlfWorld unless otherwise noted. SR = success rate (%); Δ is relative to the full TaskKV default.

Stress Test	Condition	SR / Metric	Δ vs. Default
Delimiter corruption	30% delimiters masked	63.2 ± 1.8	-1.5
Fixed stride S=64	No ReAct parsing	63.1 ± 1.5	-1.6
Fixed stride S=32	No ReAct parsing	62.4 ± 1.6	-2.3
Failure-rollout augmentation	Mixed success/fail training data	65.3 ± 1.6	$+0.6$
Temperature sampling	T=0.7 decoding	60.5 ± 2.0	–
vs. H2O at T=0.7		53.5 ± 2.3	$+7.0$ gap
Empty user prompt	System prompt only, OS	33.6 ± 2.2 (by step 3)	89% of full

1. Robustness to Segmentation. Replacing our Regex-based parser with a naive **Fixed Stride Chunking** results in only marginal degradation (-1.6% at S=64, -2.3% at S=32). This suggests that the *Semantic Alignment* ($\cos(\mathbf{h}_i, \mathbf{q}_t)$) is robust to boundary jitter–the utility of a reasoning step is distributed across its tokens, allowing the scorer to recover the signal even from imperfect chunks.

We also measured delimiter failures and safeguard false positives across the 500 evaluation trajectories (seeds 1000–1499) on AgentBench-OS and AlfWorld, totaling 14,100 logged semantic steps. Missing ReAct delimiters occur in $<1\%$ of Llama-3.1-8B trajectories and $\sim 5\%$ of Mistral-7B trajectories; the fixed-stride fallback handles these cases without manual intervention. The entropy safeguard false-positive rate is $\sim 1.4\%$ on AgentBench-OS and $<1\%$ on AlfWorld, averaging $\sim 1.1\%$ across all logged steps.

2. Cross-Domain Ablation Consistency (AgentBench-OS). To verify that component-level findings are not specific to AlfWorld, we replicate the full ablation on AgentBench-OS (Table 19). The rank order of component importance is preserved: semantic alignment and denoising remain the two most critical modules. Notably, the unsigned-saliency-label variant suffers a larger relative drop

on OS (-5.2 SR, $\sim$14% relative) than on AlfWorld (-6.1 SR, $\sim$9% relative), consistent with OS having more semantically ambiguous error loops where sign information is decisive.

Table 19: Component ablation on AgentBench-OS. Values are success rate (%), mean $\pm$ standard deviation across five seeds. The rank order mirrors the AlfWorld ablation (Table 3).

Configuration	Success Rate	Gap
Full TaskKV	**38.1 $\pm$ 2.1**	-
w/o Denoising ($\lambda_2 = 0$)	32.8 $\pm$ 2.4	-5.3
w/o Pivot Feature ($\Psi = 0$)	35.5 $\pm$ 2.1	-2.6
w/o Intent EMA ($\alpha = 1.0$)	36.3 $\pm$ 1.9	-1.8
w/o Semantic ($\lambda_1 = 0$)	29.8 $\pm$ 2.8	-8.3
Unsigned Saliency Labels ($\|\nabla\|$ only)	32.9 $\pm$ 2.3	-5.2

3. Budget Sensitivity. Performance improves with budget. At **30% budget**, TaskKV recovers $\sim$94% of the Full Cache performance (40.1 vs. 42.8 on AgentBench-OS, Table 20), suggesting substantial redundancy in the evaluated trajectories while leaving the exact redundancy level task-dependent.

Table 20: AgentBench-OS success rate across cache budgets. The full grid addresses whether TaskKV's gains persist beyond the 20% main setting.

Budget	H2O	LeanKV	Strong Heur.	TaskKV
10%	18.5	25.8	27.5	**31.2**
20%	28.4	34.2	35.2	**38.1**
30%	32.6	37.5	37.8	**40.1**
50%	37.2	40.5	40.2	**41.5**

L.2 Sensitivity to Noisy or Missing Reference Actions

The deployed scorer is trained from a signed first-order utility with respect to the ground-truth action loss and a pairwise RankNet objective. The sign distinguishes steps that lower the correct-action loss from steps that increase it; unsigned gradient magnitude is used only as an auxiliary sensitivity diagnostic. We evaluated label-noise sensitivity on AlfWorld with Llama-3.1-70B-Instruct, a 20% KV budget, greedy decoding, and five seeds. Noise was introduced only during offline label construction by replacing selected reference-action tokens uniformly at random from the vocabulary; evaluation does not use reference actions. Table 21 reports mean success rate and standard deviation over five seeds.

Performance degrades gradually under moderate random corruption, but this result does not establish robustness to systematic label bias. When reference actions are entirely missing, the signed action-conditioned training signal is unavailable. The successful-only versus mixed-trajectory comparison in Section L.1 tests whether aligned failure rollouts provide useful negative constraints; it does not solve the missing-reference-action setting. Training TaskKV without action references remains an open limitation.

Table 21: Sensitivity to random corruption of reference actions during offline training-label construction. Evaluation itself uses no reference actions. Results are AlfWorld success rate (%), mean $\pm$ standard deviation over five seeds.

Training-label corruption	AlfWorld SR	Δ vs. clean
Clean (0%)	64.7 ± 1.6	–
10%	63.5 ± 1.7	-1.2
20%	62.1 ± 1.9	-2.6
40%	58.8 ± 2.2	-5.9

L.3 Balancing λ_1 and λ_2

The weighting terms for Semantic Relevance (λ_1) and Noise Entropy (λ_2) balance the trade-off between goal alignment and denoising.

- **Noise Sensitivity (λ_2):** For "clean" tasks like KG (Knowledge Graph), λ_2 is less sensitive; setting $\lambda_2 \in [0.5, 2.0]$ yields stable performance. However, for "noisy" environments like AlfWorld, λ_2 is critical. A value of $\lambda_2 < 0.8$ fails to prune sufficient error traces, while $\lambda_2 > 2.5$ becomes over-aggressive, pruning valid exploration steps. The optimal range is $\lambda_2 \approx 1.2$.
- **Semantic Sensitivity (λ_1):** Performance is generally convex with respect to λ_1. Extremely high λ_1 causes the agent to fixate solely on the immediate goal description, ignoring procedural context (Observations). We use $\lambda_1 = 1.0$ as a stable default in our evaluated settings.

L.4 Entropy Threshold τ Sensitivity

The regime classifier uses a single threshold τ on the normalized entropy $\bar{H}(f)$ to decide whether to apply utility pruning or fall back to contiguous retention. Table 22 reports an ablation sweep of τ on both a sparse task (AlfWorld) and a dense task (LongBench-Summarization).

Table 22: Entropy threshold τ sensitivity. AlfWorld SR (%) measures sparse-task performance; LongBench-Summ. ROUGE-L measures dense-task fallback quality; Needle recall is measured under the 128k-context, 100-trial protocol at 20% budget. The sweep indicates local sensitivity around the default, not universal calibration.

τ	AlfWorld SR	LB-Summ. ROUGE-L	Safeguard rate (Summ.)	Needle recall
0.5	65.0 ± 1.8	20.2	42%	–
0.6	64.9 ± 1.7	22.1	58%	–
0.7	64.8 ± 1.6	24.5	78%	91%
0.8 (default)	$\mathbf{64.7 \pm 1.6}$	**25.8**	**88%**	**94%**
0.9	63.5 ± 1.9	26.1	95%	95%
1.0 (always fallback)	56.2 ± 2.1	26.3	100%	–

Analysis. On AlfWorld (sparse), performance is stable through $\tau = 0.8$ and drops at $\tau = 0.9$. On LongBench-Summarization (dense), ROUGE-L rises from 24.5 at $\tau = 0.7$ to 25.8 at $\tau = 0.8$ and 26.1 at $\tau = 0.9$ as the safeguard activates more often. The additional Needle-In-A-Haystack sweep used Llama-3.1-70B-Instruct, a 20% budget, greedy decoding, 128k context, and 100 trials (ten insertion depths by ten trials): recall was 91%, 94%, and 95% at $\tau = 0.7$, 0.8, and 0.9, respectively. Together, these results characterize local sensitivity around the default; they do not establish that $\tau = 0.8$ is universally self-calibrating across sparse and dense tasks.

L.5 Scorer Capacity Saturation

Is a larger scorer better? We varied the MLP hidden dimension from 64 to 4096.

- **Diminishing Returns:** Performance saturates rapidly at dim=256. Increasing to 4096 yields $< 0.2\%$ gain but increases inference latency by 15%.
- **Observation:** The measured utility signal appears low-rank in our benchmarks. A lightweight scorer with fewer than one million parameters is sufficient for the reported results.

L.6 Backbone Scaling Consistency

Does TaskKV work across model families?

- **Llama-3.1 (8B vs 70B):** TaskKV improves over H2O by $+10.7$ SR points on 8B (27.2 vs. 16.5) and $+9.7$ SR points on 70B (38.1 vs. 28.4) on AgentBench-OS. Larger models benefit more from "clean context" as they are more capable of exploiting long-range dependencies if the noise is removed.
- **Mistral-7B:** TaskKV improves over H2O by $+10.3$ SR points (28.5 vs. 18.2), suggesting that the training recipe transfers across backbone families in these diagnostics.

M Safety-Critical Constraints: The Anchor Set

Policy Specification. In real-world deployment, certain instructions (e.g., "Do not delete root files") are non-negotiable. We address this via a *Hard-Constraint Mechanism*. We define an **anchor set**

$\mathcal{M}_{anchor} \subset \mathcal{T}$ containing safety-critical steps (e.g., system prompts and safety guidelines). These are assigned a large sentinel score so that they are retained before ordinary utility-ranked steps.

Budget Interaction. The optimization problem transforms from a simple token-budgeted selection problem to one with mandatory items. The effective budget for dynamic pruning becomes $\mathcal{B}_{dynamic} = \mathcal{B}_{total} - \sum_{s_i \in \mathcal{M}_{anchor}} \ell_i$, where ℓ_i is the token length of anchor step s_i.

- **Retention Policy:** Since Anchor steps are pre-selected, they are retained regardless of their attention scores, reducing the risk of dropping safety-critical context.
- **Trade-off Analysis:** If $|\mathcal{M}_{anchor}|$ is large (e.g., extensive legal disclaimers), $\mathcal{B}_{dynamic}$ shrinks, potentially impacting reasoning performance. However, our experiments show that for typical safety prompts ($< 5\%$ of tokens), the impact on reasoning is negligible compared to the gain from pruning noise.

N Detailed Failure Mode Analysis

In Section 4.4, we presented a summary of the failure modes. Here we provide the full decomposition of error types on the AgentBench-DB held-out diagnostic domain.

Table 23: Failure mode decomposition on held-out AgentBench-DB. Context loss is forgetting variable definitions or schema; loops are repetitive actions.

Method	Total Failure %	Context Loss	Loop / Stuck	Hallucination
H2O	46.8%	12.4%	**28.5%**	5.9%
StreamingLLM	61.6%	**35.2%**	18.4%	8.0%
TaskKV	**34.9%**	14.1%	**12.2%**	8.6%

Interpretation: H2O's most frequent failure category is **Looping** (28.5%), while TaskKV has a 12.2% loop/stuck rate. This aggregate pattern is consistent with the hypothesis that TaskKV can down-weight repetitive error traces, but it does not isolate the denoising head's causal contribution. The result is a diagnostic structural-transfer pattern on DB, not evidence of broad semantic generalization.

Note on hallucination rate. TaskKV's hallucination rate (8.6%) is slightly higher than H2O's (5.9%). We attribute this to a trade-off between loop avoidance and context diversity: by retaining distant high-utility steps (e.g., system prompts, early schema definitions) instead of recent tokens, TaskKV exposes the model to a wider temporal span of context. When the retained distant steps contain outdated variable bindings or superseded schema versions, the model can hallucinate stale values. Notably, the *total* failure rate still drops substantially (34.9% vs. 46.8%), indicating that the net effect of loop reduction far outweighs the modest hallucination increase. Future work could address this by adding a staleness penalty to the utility scorer for steps whose content has been explicitly superseded by later observations.

O Extended Discussion: Structural Priors vs. Learned Utility

We compare TaskKV against the emerging family of structured pruning methods (ChunkKV Liu et al. [2025a], KeepKV Tian et al. [2025]). While official codebases for these concurrent works are not yet adapted for agentic loops, we implemented matching versions of their core mechanisms to test our hypotheses.

O.1 Theoretical Distinctions: The Causality Gap

We distinguish TaskKV from Global Optimization methods.

1. Streaming vs. Retrospective Consensus. Global methods rely on scanning the entire sequence (past and future) to identify important tokens via bidirectional attention ($O(T^2)$). In agentic workflows, the future trajectory is *unknown* and causally dependent on the current memory state. TaskKV adheres to strict *streaming causality*.

2. Offline Proxy vs. Online Success. To provide a fair comparison despite the architectural incompatibility, we evaluated these methods on an **offline trajectory reconstruction** task (Table 24). While global methods achieve lower perplexity (PPL), they miss many *action-critical*

tokens. TaskKV, trained with action-utility labels, achieves higher PPL but higher **action recall** (recall of tokens used in the ground-truth reasoning trace).

3. Optimization Objective (Perplexity vs. Utility). PPL-oriented compression can preserve "high-entropy noise" (e.g., random identifiers in logs). TaskKV instead uses action-utility labels that are motivated by goal relevance.

Table 24: Offline vs. online metrics on AgentBench. Global methods minimize PPL but miss critical reasoning steps.

Method	PPL (↓)	Action Recall (↑)	SR (Online)
H2O (Local)	6.12	45.2%	28.4
ChunkKV (Global)	**5.75**	58.1%	N/A (Offline)
TaskKV (Ours)	6.05	**82.4%**	**38.1**

O.2 Empirical Analysis of Competitors

1. The "Coherent Noise" Problem (vs. ChunkKV). Chunk-based methods Liu et al. [2025a] assume that information is locally clustered. We implemented a **Chunk-Heuristic** that retains contiguous blocks with high aggregate attention.

- *Observation:* In AgentBench, this method suffers a **-8% SR drop** compared to TaskKV.
- *Analysis:* Agents often generate "Coherent Failure Traces" (e.g., a properly formatted but factually wrong 50-step loop). ChunkKV preserves these "high-quality errors," polluting the context. TaskKV's denoising head (λ_2) explicitly learns to down-weight these structures despite their local coherence.

2. The "Syntax Fragility" Problem (vs. Merging/KeepKV). Merging methods Tian et al. [2025], Liu et al. [2025b] (averaging token vectors) work well for prose but fail for code/formal languages.

- *Observation:* On the OS (bash) and DB (SQL) benchmarks, a token-merging baseline (compression ratio $4\times$) resulted in a $3\times$ **increase in syntax errors** (e.g., malformed JSON).
- *Analysis:* In formal languages, averaging the vectors of ':' and ';' yields a nonsense vector that confuses the parser. TaskKV's discrete step-level selection avoids this representation collapse.

P Practical Boundaries and Robustness

In this section, we analyze practical boundaries of the TaskKV framework, specifically addressing the signal-to-noise ratio of first-order labels, the cost of dynamic scoring, and the stability of the intent vector under vague prompts.

P.1 First-Order Utility: Useful Evidence vs. Confusion

A concern with gradient-based labels is distinguishing between steps that are useful for the correct action and steps that are outliers causing hallucination. A high gradient norm $\|\nabla_{\mathbf{h}}\mathcal{L}\|$ only indicates sensitivity; by itself, it does not imply that the step is useful.

We address this by using the signed first-order loss change under a local masking approximation, computed against the *ground-truth action* rather than the model's own prediction. Recall that our loss function $\mathcal{L}_{final}$ is the negative log-likelihood of the *ground-truth action* A^*, not the model's predicted action.

$$\mathcal{U}_{proxy}(s_i) = -\left\langle \nabla_{\mathbf{h}_i}\mathcal{L}_{final}, \mathbf{h}_i - \mathbf{h}_{mask} \right\rangle . \tag{9}$$

Consider a "Hallucination Trigger" token s_{noise} that diverts the model's probability mass away from A^* towards an incorrect A_{err}.

- **Scenario A (Magnitude Only):** If we used only $\|\nabla_{\mathbf{h}}\mathcal{L}\|$, s_{noise} could receive a large label simply because it strongly affects the model, even if that effect is harmful.
- **Scenario B (TaskKV Training):** With signed first-order utility, a confusing step that increases the loss of A^* receives low or negative utility, while a step whose removal increases the loss receives positive utility. RankNet then learns preferences from the sign and relative magnitude of this utility.

Therefore, TaskKV does not rely on high saliency alone. It uses a signed proxy for whether retaining the step helps the ground-truth action objective, and validates that proxy against LOO labels in Appendix J.

P.2 Amortized Complexity of Dynamic Intent Alignment

The term $\cos(\mathbf{h}_i, \mathbf{q}_t)$ implies a dependency on the time-varying intent $\mathbf{q}_t$. A naive implementation would require re-computing similarity for all $|\mathcal{H}|$ cached steps at every generation step t, leading to $O(T^2)$ complexity. We show that the **Amortized Cost** per generation step is negligible relative to the attention computation.

1. Vectorized Pre-fetching. On modern GPUs (A100), computing the cosine similarity between a query vector $\mathbf{q}_t \in \mathbb{R}^d$ and a memory matrix $M \in \mathbb{R}^{\mathcal{B} \times d}$ is a single matrix-vector product. For $\mathcal{B} = 4096$ retained steps and $d = 8192$ (Llama-3.1-70B), this operation is small relative to the attention computation in our profiling. The practical bottleneck is memory bandwidth for loading the features, which we mitigate via a custom Triton kernel that keeps compressed features $\mathbf{h}_i$ in SRAM.

2. Staleness Bound of Lazy Re-ranking. We employ a **Lazy Re-ranking** strategy with update interval Δ_{lazy}. The error induced by using a "stale" intent $\mathbf{q}_{t-\tau}$ instead of $\mathbf{q}_t$ is bounded by the accumulated local drift of the normalized EMA state. Ignoring the final normalization for this conservative bound, the EMA update is $\mathbf{q}_t = \alpha \mathbf{h}(x_t) + (1-\alpha)\mathbf{q}_{t-1}$, so:

$$\|\mathbf{q}_t - \mathbf{q}_{t-\tau}\| \leq \sum_{k=0}^{\tau-1} \alpha \|\mathbf{h}(x_{t-k}) - \mathbf{q}_{t-k-1}\| \leq \epsilon. \tag{10}$$

Empirically, the intent vector changes slowly between consecutive tokens. Setting $\Delta_{lazy} = 64$ tokens introduces fewer than 1.5% pairwise ranking inversions (i.e. the fraction of discordant step pairs between fresh and stale rankings; equivalently, Kendall's $\tau \geq 0.97$), with no measurable impact on Success Rate in our experiments, while reducing the scoring frequency proportionally.

P.3 Intent Vector Stability under Vague Prompts

A potential weakness is the "Cold Start" problem: if the user prompt is vague (e.g., "Fix the computer"), is the initial intent $\mathbf{q}_0$ sufficient?

In practice, two mechanisms mitigate this:

- **Phase 1: Prompt/recency dominance.** Initially, $\mathbf{q}_0$ is diffuse. The default policy keeps newly completed steps in the recent region before eviction, and deployment can additionally enable the optional anchor constraint for safety-critical prompts. In both cases these tokens are counted inside the same retained-token budget.
- **Phase 2: Intent Sharpening.** As the agent executes exploration steps (e.g., 'ls /logs'), the observations obs_1 are integrated into $\mathbf{q}_t$ via the EMA. The intent vector rapidly specializes from the generic "Fix" direction to "Error Log Analysis" as observations accumulate.

Our experiments show that even with empty user prompts (relying solely on the system prompt), TaskKV recovers 89% of performance within 3 steps of interaction (Table 18), suggesting that the dynamic state ($\mathbf{q}_t$) matters more than the initial condition in this stress test.

P.4 Design Rationale for Key Choices

1. Why a Time-Varying Scorer? A static linear probe on embeddings learns time-invariant saliency (e.g., "SQL is always important"). Agentic utility is time-variant: a specific SQL query is critical at step $t = 10$ but becomes noise at $t = 50$. TaskKV's scorer $f(s_i, \mathbf{q}_t)$ conditions on the current intent, which a static probe cannot.

2. Layer-Adaptive Pruning Degrades Agent SR. While Layer-Adaptive methods optimize Perplexity (PPL), they reduce success rate in our agentic experiments. We attribute this to the importance of vertical contiguity in the residual stream: pruning layer L while retaining $L+1$ creates a discontinuity in the update $x_{l+1} = x_l + \text{Attn}(x_l, KV_l)$. In our ablations, synchronized pruning better preserves this cross-layer consistency.

Q Additional Analysis

We report supplementary experiments on unstructured data streams, latency at small batch sizes, and the model-scale vs. context-length tradeoff.

Q.1 Robustness to Unstructured Streams

The Integral Approximation. A potential concern is whether the reliance on "Semantic Centroids" and regex-based boundaries causes TaskKV to degenerate when applied to implicit Chain-of-Thought (CoT) or unstructured long paragraphs. Let the reasoning process be a continuous semantic flow $S(t) \in \mathbb{R}^d$. TaskKV approximates the utility integral $\int \mathcal{U}(S(t))dt$ via discrete Riemann sums over blocks. When explicit delimiters (e.g., "Thought:") are absent, TaskKV falls back to **Fixed-Stride Chunking** (default $L = 64$).

$$\mathbf{h}_{chunk} = \frac{1}{L} \sum_{j=1}^{L} \mathbf{h}(t_j)$$

Even within a single monolithic paragraph, the "Running Centroid" $\mathbf{h}_{chunk}$ effectively captures the *Locally Dominant Topic* (e.g., a specific variable declaration or logic clause). The cosine similarity $\cos(\mathbf{h}_{chunk}, \mathbf{q}_t)$ thus operates as a **Low-Pass Semantic Filter**.

Block-Sparse Importance Sampling. Standard token pruning (e.g., H2O) operates at the token level, often dropping tokens *within* a syntactic unit. TaskKV, even in fallback mode, operates at the step/chunk level. By retaining contiguous blocks, we preserve local syntactic structure required for attention heads to resolve immediate dependencies, while the scorer handles longer-range semantic relevance. *Empirical check:* On the UltraChat dataset (unstructured dialogue), TaskKV (Fallback Mode) maintains a perplexity of 5.92, whereas H2O (Token Mode) degrades to 6.45, suggesting that block-wise filtering is more stable than token-wise attention filtering in this setting.

Q.2 Latency at Unit Batch Size

At BS=1 the compute-to-memory ratio changes, so we profile the scorer overhead separately.

Profiled Breakdown. We report a unit-batch diagnostic profile for a sequence of length $L = 128k$ on an RTX 4090.

- **Attention Cost (Memory-Bound):** Loading the KV cache requires moving $L \times d \times 2$ bytes ≈ 1 GB per step. At 1 TB/s bandwidth, this takes $T_{mem} \approx 1\,\text{ms}$.
- **Scorer Cost (Compute-Bound):** The residual MLP scorer ($3d + 1 \to 256 \to 64 \to 1$) involves matrix-vector multiplications. The raw FLOP count is small relative to attention, but in practice kernel launch overhead, memory access patterns, and pipeline stalls dominate at BS=1. Our separate BS=1 profiling measured $\sim 1.2\,\text{ms}$ per scoring event; Table 9 reports the BS=32 end-to-end serving profile.

The key tradeoff is that TaskKV's scorer adds per-step overhead but reduces the KV cache size from L to $0.2L$, proportionally reducing the memory bandwidth cost of subsequent attention passes. Whether this yields a net speedup depends on the hardware memory bandwidth and batch size. In our A100 profiling at BS=32, the amortized scorer overhead ($\sim 3.4\,\text{ms}$) is outweighed by the attention savings from the smaller cache, but at BS=1 on bandwidth-limited devices, the net effect is more marginal.

Q.3 Stability of Intent Adaptation

We model the Intent Vector $\mathbf{q}_t$ dynamics as a control theory problem.

- **Sensitivity Analysis:** We swept $\alpha \in [0.1, 0.9]$. The performance curve forms a broad plateau between $[0.2, 0.5]$.
- **Handling Abrupt Task Shifts:** In rare cases where the user issues a "Stop, do X instead" command, two mechanisms mitigate inertia:

1. **Recency and optional anchors:** The new instruction is typically processed as a new user prompt, which first enters the *Recency Buffer*; deployments that mark it safety-critical can additionally retain it through the optional anchor constraint.
2. **Pivot Indicator (Ψ):** The learned scorer can assign higher utility to high-surprise strategy shifts when similar pivots are supported by the training signal, reducing over-reliance on cosine similarity alone.

Therefore, $\mathbf{q}_t$ serves as a smooth trajectory guide, while Ψ handles abrupt turns. The combination is robust to frequency shifts.

Q.4 Model Capacity vs. Context Budget Tradeoff

We examine the practical tradeoff between model size and retained context under a fixed memory-accounting budget. This appendix experiment is intended as a capacity diagnostic rather than a universal deployment recipe, because exact feasibility depends on the inference engine, weight quantization implementation, and activation workspace. We use a **48-GB VRAM budget** (single A6000) for the accounting comparison:

- **Configuration A:** Llama-3.1-8B in BF16 (native 128k context). Fits fully in 48 GB ($\sim$16 GB weights + $\sim$16 GB KV at 128k).
- **Configuration B:** Llama-3.1-70B + TaskKV (compressed to 8k retained context). The memory-accounting estimate is feasible with 4-bit weights ($\sim$35 GB) plus compressed KV ($\sim$2.5 GB), assuming a serving stack whose activation and kernel workspaces fit the remaining margin.

We define **AgentBench-Hard** as the long-trajectory subset of AgentBench-OS: tasks requiring $\geq$80 interaction steps (125 out of 500 test seeds, i.e. the top quartile by trajectory length).

Results on AgentBench-Hard:

- **Config A (8B, Full Context):** SR = 32.5%. The model can "read" the entire history but fails to "plan" complex multi-step logic. It often gets lost in the details despite having perfect recall.
- **Config B (70B, TaskKV):** SR = 38.5%. The stronger model operates on a compressed memory and can recover useful task structure from the retained trajectory skeleton.

Conclusion: In this controlled setting, a stronger model with compressed memory outperforms a weaker model with full context under the same VRAM budget. This suggests a practical tradeoff between model capacity and retained context width under fixed memory constraints.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state TaskKV as an empirical system for sparse agentic trajectories with a signed utility label, a structured residual scorer, and an entropy-based regime classifier. Claims are scoped to the evaluated benchmarks and budget regime; the facility-location motivation appears only in the appendix (Appendix D) where a restricted formal bound is provided.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a dedicated Limitations section discussing sparse-regime assumptions, dense-regime fallback behavior, the theory-practice gap, and remaining OOD validation needs.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Proposition 1 (Appendix D) provides a modular proxy regret bound under Assumption 1 (Sparse Agentic Regime). The assumption is stated explicitly, the proof sketch is included, and the scope caveat clarifies that the bound characterizes the favorable operating regime rather than providing a worst-case guarantee for arbitrary inputs.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper discloses all benchmarks, baselines, budgets, training trajectories, seed splits, held-out domains, hyperparameters, statistical reporting, and compute assumptions needed to understand and independently reimplement the main experiments. The supplementary material includes: (a) anonymized reference code for all algorithmic components (scorer, memory manager, entropy safeguard, synchronized KV-cache eviction, five baseline wrappers); (b) the complete scorer-training pipeline (trajectory collection with signed labels and RankNet training); (c) evaluation scripts with compressed ReAct rollout and per-seed result tracking for AgentBench and AlfWorld; (d) a slow-path HuggingFace end-to-end demo; (e) per-seed raw experiment logs (`main_results_per_seed.csv`, `ablation_per_seed.csv`) with a CPU-only verification script (`verify_reported_results.py`) that recomputes all table summaries; (f) hyperparameter configuration files matching the paper; (g) a step-by-step reproducibility guide; and (h) all figure data with generation scripts so that every plotted value can be traced back to the reported tables or profiling runs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The supplementary package provides an anonymized, installable Python package (`pip install -e .`) containing: all algorithmic components (scorer matching Eq. 3, memory manager matching Algorithm 1, entropy safeguard matching Eqs. 4–5, synchronized KV-cache eviction, and baseline wrappers for H2O, SnapKV, StreamingLLM, Strong Heuristic, and AdaKV); the full scorer-training pipeline (trajectory collection with signed first-order action-utility labels and RankNet training); evaluation scripts with compressed ReAct rollout and per-seed result tracking for AgentBench and AlfWorld; a slow-path HuggingFace end-to-end demo (`toy_demo_hf.py`); per-seed raw experiment logs with a CPU-only verification script that recomputes all table summaries from the raw logs; hyperparameter configuration files matching the paper; and a step-by-step reproducibility guide. The remaining external requirements (public benchmark environments and backbone model weights) are publicly available under their respective licenses.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: The main paper and appendix specify benchmarks, models, baselines, budgets, seed splits, optimizer, loss, hyperparameters, mixed-trajectory label handling, and the online/offline distinction for TaskKV-Proxy versus the offline SMI diagnostic comparator.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: All main success-rate tables report mean $\pm$ sample standard deviation across five independent seeds (1000–1004). The supplementary material includes per-seed raw logs (`main_results_per_seed.csv`, `ablation_per_seed.csv`) from which these statistics can be independently recomputed using the included CPU-only verifier (`verify_reported_results.py`). The proxy-label analysis reports correlation statistics including point-biserial correlation and Kendall's τ. Figures that report aggregate values (e.g., efficiency scatter, token recall) are anchored to the same per-seed table values; the critical-token recall (Figure 9d) reports mean $\pm$ std across the same five seeds.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The paper reports the A100-80GB serving setup (4$\times$A100-80GB, vLLM 0.4.0, tensor_parallel=4, BF16), batch size, latency breakdowns, full pipeline cost breakdown ($\sim$45–65 GPU-hours total including trajectory collection, feature extraction, and

scorer training on 1×A100-80GB), and hardware assumptions for each experiment. Compute requirements are specified for the 70B TP=4 serving setup used in both agent-success evaluation and throughput profiling. The inference-scaling profile (Figure 10c) was collected on the same 70B TP=4 setup; the supplementary `run_metadata.json` records the complete experimental protocol.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The work uses public benchmarks and open or cited model families, does not collect private human data, and is intended to improve inference efficiency. Potential dual-use implications are discussed in the limitations section.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The limitations section notes that efficiency gains can lower both beneficial and harmful deployment costs, and recommends pairing such methods with standard safety policies and auditing.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The submission does not release a new high-risk pretrained foundation model, scraped dataset, or generative model. The released asset is an anonymized reference implementation for a KV-cache compression method.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Existing benchmarks, model families, and systems used in the experiments are cited in the paper. The supplementary manifest describes the included reference code and expected dependency stack.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

Justification: The submission introduces a reference implementation of TaskKV's algorithmic components and documents its purpose, directory structure, local verification command, dependencies, and benchmark-reproduction prerequisites in the supplementary code manifest and README.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve human-subject studies and therefore does not require IRB approval or equivalent review for such studies.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are central to the evaluated systems: TaskKV operates on Transformer KV caches and hidden states, and the experiments specify the LLM backbones used as agents.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.