KV CACHE EVICTION AS SYSTEM ONE MEMORY CONTROL FOR SYSTEM TWO REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-chain reasoning expands the key–value (KV) cache, forcing memory-allocation decisions before the future roles of intermediate reasoning states are clear. We frame KV cache eviction as System One memory control for System Two reasoning: satisfying a fixed memory budget while avoiding premature semantic commitments. We propose a framework that uses relations within the evolving reasoning trajectory to selectively guide retention, rather than assigning a fine-grained importance score to every cached entry. It identifies active dependencies and explicitly superseded reasoning, while withholding semantic preferences when their roles remain unresolved. These judgments serve as revisable constraints on budget-driven eviction rather than direct deletion commands, with a simple stochastic policy resolving the remaining allocation decisions. Semantic audit runs asynchronously with generation, keeping the eviction path lightweight. Experiments on mathematical reasoning benchmarks across multiple reasoning models show higher mean accuracy than score-based and prompt-protected random eviction at matched cache budgets. An efficiency evaluation further shows reduced cache memory relative to full attention, with little additional eviction-path overhead over random eviction. These results suggest that selective, revisable semantic guidance can improve memory allocation for extended reasoning without requiring exhaustive importance estimation.

1 INTRODUCTION

A reasoning system must manage memory while it is still discovering what its computation requires. Long chains of thought introduce intermediate results, tentative assumptions, and revisions that support subsequent reasoning (Wei et al., 2022). Under a fixed key-value (KV) cache budget, eviction determines which representations remain directly available to attention. We view this as *System One memory control for System Two reasoning*, using the distinction operationally: direct retention decisions regulate the state available to the model's extended reasoning (Kahneman, 2011; Zhang et al., 2026a). This perspective includes existing eviction policies, but raises an additional design question: *how should a controller act when memory pressure requires a decision before the reasoning trace supports a clear semantic judgment?*

Existing methods offer increasingly effective ways to prioritize cached states, including attention-based retention (Zhang et al., 2023; Li et al., 2024), redundancy-aware scoring (Cai et al., 2025), and coarse-to-fine selection (Behnam et al., 2025). Recent methods also move beyond immediate attention: LookaheadKV anticipates response attention for prompt eviction without draft generation (Ahn et al., 2026), LazyEviction observes recurring importance before eviction (Zhang et al., 2026b), and ThinKV adapts compression to thought types and transitions (Ramachandran et al., 2026). Meanwhile, Random Attention shows that prompt-protected random eviction can be competitive in its evaluated reasoning settings (Wang et al., 2026). This motivates a complementary direction: use semantic information to identify where a strong default should be constrained, rather than require another dense ranking of the entire cache.

In an evolving reasoning trace, a span's contribution can depend on both the current trajectory and the other states still available. A derivation may remain an active dependency after a strategy change; a replacement may make an earlier derivation redundant. Crucially, a suspected error and a settled correction provide different grounds for retiring an assumption. Later reasoning can therefore change

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

Table 1: **Motivation. (a)** Attention is poorly aligned with causal reasoning utility: low-attention selection is worse than random for identifying causally harmful spans (AUROC 0.17), while harmful spans are disproportionately concentrated among high-attention states (0.83), despite weak overall rank correlation ($\rho \approx 0.09$). **(b)** Direct deletion based on semantic judgments also degrades accuracy, substantially more than random deletion. These results motivate separating semantic guidance from physical eviction.

Ranking rule	**Harm AUROC ↑**	**Rank corr. ρ**
Random ranking	0.50	0.00
Low attention	0.17	−0.09
Oldest span first	0.57	−0.06
Causal detector (MLP)	0.86	+0.08
High attention	0.83	+0.09

(a) Causal harm vs. cache-selection signals.

Cache policy	**Acc. ↑**	**Δ vs. Full**
Full attention	74.5	–
Random deletion	70.2	−4.3
Attention-guided deletion	68.1	−6.4
Direct semantic deletion	59.6	−14.9
Global-only semantic del.	66.0	−8.5

(b) Direct KV deletion policies.

a span's role, not merely reveal a fixed importance value. Context-conditioned scores can represent such relationships, but an ordering alone does not distinguish uncertain relevance from evidence warranting semantic intervention. Nor does semantic supersession establish causal dispensability: recognizing that a statement has been replaced does not prove that removing its KV representation will preserve future computation. Consistent with these concerns, Table 1 shows weak overall attention–harm rank correlation ($\rho \approx 0.09$) and lower accuracy under direct semantic deletion than under random deletion (59.6% vs. 70.2%).

We therefore separate *mandatory cache allocation* from *revisable semantic commitment*. The cache must satisfy its budget whenever eviction is triggered, but the semantic component need not assign every entry a usefulness verdict at that moment. A default policy can resolve physical allocation while semantic judgments remain open to revision as the trajectory develops. This is not deferred eviction: unconstrained entries remain eligible for removal, and subsequent guidance cannot recover discarded states. The purpose is instead to avoid forcing unresolved relationships into semantic retention preferences. We seek a semantic controller that decides not only what to retain, but also when the available reasoning warrants constraining its default policy.

We introduce **SPARE** (Selective Protection and Abstention for Reasoning-time Eviction), which implements this separation through trajectory-conditioned semantic audits over *episodes*, contiguous spans of generated reasoning. Using episodes, previous constraints, the current goal, and recent context, the controller selectively labels active dependencies PROTECT and explicitly superseded episodes OBSOLETE, leaving the remainder UNCONSTRAINED. Tentative corrections do not suffice for obsolescence; later audits may revise constraints when the trajectory supplies new evidence. Auxiliary semantic management has also been explored by SideQuest (Kariyappa & Suh, 2026); SPARE updates selective retention constraints, rather than issue direct deletion commands. UNCONSTRAINED is thus an explicit decision not to impose a semantic preference, not a claim of low importance or guaranteed future protection.

These episode-level constraints guide token-level eviction. Within a capped protection budget, SPARE retains protected entries, fills remaining capacity by uniform sampling from unconstrained entries, and retains obsolete entries only when needed to meet the prescribed cache size. Audits run asynchronously: eviction uses the latest constraints, and unaudited episodes follow the default policy. When all episodes are unconstrained, SPARE reduces to uniform stochastic eviction under the same prompt and recency protections. Its central distinction is therefore not a more expressive scoring function, but a protocol for deciding *when and where* evolving semantic evidence should affect memory allocation. Our contributions are threefold:

- **A memory-control formulation.** We separate the obligation to meet a KV budget from the decision to impose semantic retention constraints during ongoing reasoning.
- **A trajectory-conditioned control protocol.** SPARE combines revisable PROTECT, OBSOLETE, and UNCONSTRAINED states with asynchronous audits and constraint-guided stochastic eviction.
- **An empirical evaluation.** On the evaluated mathematical reasoning benchmarks, SPARE achieves higher mean accuracy than the compared score-based and stochastic baselines at matched KV budgets.

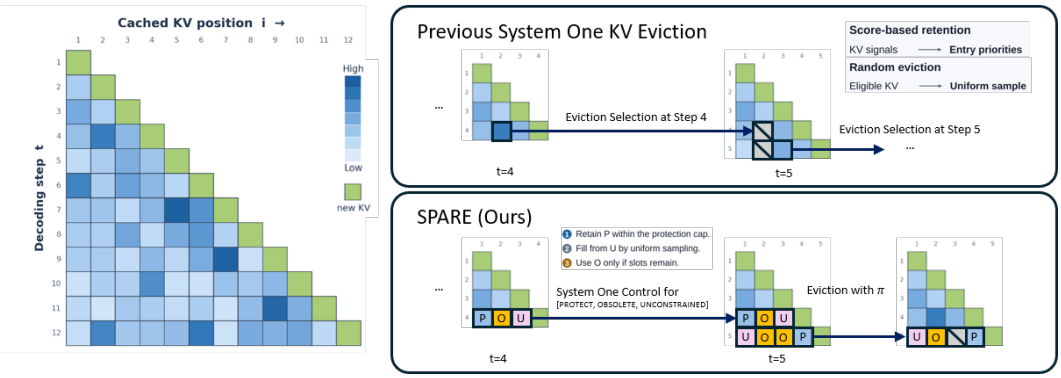

Figure 1: **Overview of SPARE.** SPARE frames KV eviction as System One memory control for System Two reasoning. As the reasoning trajectory evolves, a semantic controller revises sparse constraints over reasoning episodes—PROTECT for active dependencies, OBSOLETE for settled supersessions, and UNCONSTRAINED when their roles remain unresolved.

2 SPARE: SELECTIVE PROTECTION AND ABSTENTION FOR REASONING-TIME EVICTION

We consider decode-time KV cache management for an autoregressive System Two reasoning model (Kahneman, 2011; Zhang et al., 2026a; Almeida, 2026). At decoding step t, let $G_t^{(h)}$ denote the generated token positions whose KV states remain resident in head h. Prompt KV states are pinned throughout decoding. We additionally reserve a recency buffer $B_t^{(h)} \subseteq G_t^{(h)}$ containing the r most recently generated resident positions. The positions eligible for eviction are

$$C_t^{(h)} = G_t^{(h)} \setminus B_t^{(h)}. \tag{1}$$

Given a budget of K eligible positions per head, the cache manager must select a survivor set $S_t^{(h)} \subseteq C_t^{(h)}$ satisfying

$$|S_t^{(h)}| = \min\{K, |C_t^{(h)}|\}. \tag{2}$$

Thus, at most $K + r$ generated positions are retained per head, in addition to the pinned prompt states. Surviving KV entries preserve their original token positions and positional-encoding phases; evicted entries are not reconstructed.

System One Memory Control for System Two Reasoning. System-Two-like reasoning can revise assumptions, reuse intermediate results, and abandon earlier branches. Consequently, the role of past reasoning may change during decoding. SPARE uses a semantic controller to identify a small number of active dependencies and explicit supersessions, while abstaining where the relation is unclear. A separate cache policy converts these judgments into token-level eviction decisions. The controller is instantiated with System One models such as Jev (Almeida, 2026), CLM Kwok et al. (2026) or general purpose language-models. SPARE's eviction rule depends on the resulting constraints, not on the controller's internal architecture.

2.1 ONLINE REASONING EPISODES

SPARE makes semantic judgments over spans of reasoning, whereas KV entries are stored at token positions. We divide the generated trajectory online into non-overlapping episodes $\mathcal{E}_t = \{E_1, \ldots, E_{n_t}\}$. Episode E_j covers a contiguous interval of token positions $I(E_j) = [s_j, e_j)$. An episode closes at the first paragraph boundary after $m_{\min}$ tokens or at $m_{\max}$ tokens, whichever occurs first. An episode still being generated remains unconstrained.

For each episode, SPARE records a stable identifier, its token interval, and its generated text independently of the retained KV entries. A judgment about an episode applies only to KV entries within its interval that are still resident. Previously evicted entries cannot be restored by a later protection decision.

2.2 Relational Judgments and Abstention

At decision step τ, the semantic controller receives the original problem, an ordered representation of closed episodes and previous judgments, and the latest w raw generated tokens. Older episode text may be abbreviated when the input exceeds the controller's budget. The recent raw tokens help preserve corrections that an abbreviated representation might omit.

For each candidate episode, the System One controller—e.g., Jev (Almeida, 2026), CLM (Kwok et al., 2026), or general-purpose language model agent—assesses two relations:

1. Does the current reasoning still depend on information in this episode?
2. Has a later, settled reasoning step explicitly superseded its role?

An established active dependency yields PROTECT. An explicit, settled correction, replacement, or abandoned branch yields OBSOLETE. When neither relation is sufficiently supported, the episode remains UNCONSTRAINED. Mere age, low attention, similarity to a later passage, or a tentative alternative does not establish obsolescence. Conflicting or invalid judgments are also left as UNCONSTRAINED.

These labels have distinct operational meanings. PROTECT prevents eviction of currently resident entries belonging to the episode. OBSOLETE makes the episode a preferred eviction source when memory must be removed; it does not assert that deletion is harmless. UNCONSTRAINED is abstention, not a claim that the episode is unimportant. Since the reasoning trajectory evolves, the controller may revise a previous label at a later decision step.

2.3 Protection Budget and Eviction Policy

To prevent protection from occupying the entire cache, let $K_{\text{prot}} < K$ denote the maximum number of token positions assigned to protected episodes:

$$\sum_{E \in \mathcal{E}_{\tau,\text{protect}}} |I(E)| \leq K_{\text{prot}}. \tag{3}$$

If proposed protected episodes exceed this limit, they are considered in the controller's protection-priority order. An entire episode is admitted when it fits; otherwise, it remains unconstrained. The cap ensures that currently resident protected positions fit within the eligible survivor budget.

At eviction step t, SPARE uses the most recently completed judgments. Let

$$D_t^{(h)} = \max\{0, |C_t^{(h)}| - K\} \tag{4}$$

be the number of positions that must be removed from head h. SPARE applies a base eviction policy π to positions labeled OBSOLETE first, until either $D_t^{(h)}$ positions have been selected or no obsolete positions remain. If further evictions are required, it applies π to UNCONSTRAINED positions for the remainder. Resident PROTECT positions are excluded from eviction while their protection remains active. By the protection cap, $|C_t^{(h)} \cap I(P_\tau)| \leq K$, and therefore the required removals can always be satisfied using OBSOLETE and UNCONSTRAINED positions alone.

The policy π determines which positions to select *within* each group. It may use uniform sampling, attention-based scores, or another token-level criterion. Thus, SPARE determines the order in which semantic groups become eligible for eviction, while π performs the token-level selection. If the controller abstains on all episodes, every eligible position is unconstrained and SPARE reduces to the base policy π.

2.4 Online Execution

SPARE requests its first semantic judgment at the first closed episode boundary after the generated cache enters the budgeted regime. Subsequent requests occur after T generated tokens. A newly closed episode containing a revision cue may trigger an earlier request after at least $T_{\min}$ tokens; the cue itself does not assign a label.

3 EXPERIMENT

3.1 EXPERIMENTAL SETUP

Models and tasks. We evaluate SPARE on Qwen3-8B, Qwen3-4B, and Phi-4-reasoning (14B) across long-form reasoning benchmarks: MATH-500 (500 problems), AIME 2024 (30), AIME 2025/26 (60), and HMMT 2025 (60). For overlapping models and tasks, we follow the evaluation setting of Random Attention (Wang et al., 2026) where applicable, including the released sampling configurations (temperature 0.6 for Qwen and 0.8 for Phi-4-reasoning, with top-$p = 0.95$), a 32K reasoning budget, and matched task-level KV budgets. Specifically, we use $K = 1024$ for MATH-500 and $K = 4096$ for AIME and HMMT, corresponding to roughly $4\times$ KV compression.

Baselines and cache protocol. We compare against Full Attention (no eviction), Random Attention (RA) Wang et al. (2026), SnapKV Li et al. (2024), R-KV Cai et al. (2025), and VaSE Chang et al. (2026) under matched KV budgets and the same inference engine. To ensure a fair comparison of eviction policies, we also follow Random Attention's prompt-protection protocol (Wang et al., 2026): the full prompt KV cache is pinned and never eligible for eviction. All compressed methods receive this same protection, use a shared recency buffer of $r = 64$, and operate under the same physical eviction protocol. Detailed baseline configurations and cache-management parameters are given in Appendix A.

SPARE. SPARE segments the generated reasoning online into 128–256-token episodes and periodically audits their semantic roles using Jev (`typesafe/jev-1.13`). The controller operates only on the textual reasoning state, without access to the base model's hidden states, attention scores, or KV representations. At each audit, Jev assigns one of PROTECT, OBSOLETE, or UNCONSTRAINED to each episode in a single request. SPARE decouples semantic auditing from physical eviction, allowing audits to run asynchronously with token generation; we quantify the resulting overhead in Sec. 3.5. Audit scheduling, input construction, and constraint thresholds are detailed in Appendix A.

Metrics and statistics. Our primary metric is accuracy, computed from the final answer using symbolic verification. All method comparisons are paired over shared problem–seed runs. We report paired bootstrap confidence intervals and Wilcoxon signed-rank tests, and additionally measure reasoning length, forced-termination rate, KV footprint, and auditing overhead.

3.2 MAIN RESULTS

Table 2 reports pass@1 accuracy under matched KV budgets. Across models and reasoning benchmarks, SPARE consistently improves over cache policies that allocate the same physical memory without relational semantic guidance. On Qwen3-8B, for example, average accuracy increases from 75.9% with RA to 81.6% with SPARE under the same cache budget. Because SPARE and RA share the same stochastic cache manager and differ primarily in the semantic constraints imposed before eviction, this gap isolates the benefit of relational auditing rather than additional cache capacity.

SPARE also compares favorably with attention-, redundancy-, and value-based compression methods such as SnapKV, R-KV, and VaSE. The gains are observed across both competition-style reasoning tasks and MATH-500, and across Qwen3-8B, Qwen3-4B, and Phi-4-reasoning. This suggests that the benefit is not tied to a particular model scale or benchmark, but to preserving reasoning states according to their role in the evolving trajectory.

Importantly, SPARE does not attempt to semantically rank the entire cache. Most memory remains UNCONSTRAINED; the auditor intervenes only when there is sufficiently strong evidence that a reasoning episode remains depended upon or has been superseded. The results indicate that these sparse constraints are sufficient to improve fixed-budget cache allocation without requiring dense semantic scoring.

3.3 DEFERRED COMMITMENT UNDER RELATIONAL UNCERTAINTY

A central design choice of our controller is that it does not force every reasoning episode into a keep-or-remove decision. When neither continued dependence nor settled supersession is sufficiently

Table 2: Accuracy (%) under KV-cache eviction at each task's $\sim 4\times$ compression (per-head budget $K=1024$ on MATH-500, $K=4096$ on AIME and HMMT).

		MATH-500	AIME		HMMT 2025
			2024	2025/26	
Qwen3-8B	Full attention	97.1	70.8	65.8	49.6
	SnapKV	87.0	70.0	55.0	43.8
	R-KV	84.9	70.0	51.7	36.2
	VaSE	85.1	70.8	57.1	42.1
	Random Attention	91.0	70.0	54.2	44.6
	SPARE (ours)	**92.5**	**71.7**	**61.3**	**46.7**
Qwen3-4B	Full attention	96.8	71.7	63.7	51.2
	SnapKV	86.5	67.5	58.8	42.9
	R-KV	86.0	68.3	56.2	37.5
	VaSE	85.3	70.8	59.2	43.5
	Random Attention	90.4	71.7	60.9	47.9
	SPARE (ours)	**91.1**	**72.5**	**62.1**	**51.7**
Phi-4-reasoning	Full attention	95.3	71.7	71.7	48.3
	SnapKV	92.8	70.0	66.4	43.3
	R-KV	92.8	65.0	**67.5**	39.2
	VaSE	92.6	66.7	66.7	45.8
	Random Attention	93.2	**70.7**	67.1	43.3
	SPARE (ours)	**94.4**	70.0	**67.5**	**46.7**

clear, the episode remains UNCONSTRAINED. Because reasoning-state utility is set-conditional and state-dependent, an episode's role may change as the active reasoning state evolves. A controller that forces dense decisions at each audit therefore risks making irreversible commitments prematurely.

We first examine the subsequent usage of UNCONSTRAINED episodes to understand the risk of forcing a keep-or-remove decision (Table 3). If the controller were forced to evict based on age, the oldest quartile of these episodes is still re-referenced 49.6% of the time after at least 1K additional tokens. Attention-based policies face a similar issue: the lowest-scoring quartile of SnapKV—its primary eviction target—is reused at a rate of 40.8%. Even if we forced a decision using the agent's own uncertainty, the lowest-margin quartile retains a substantial 36.2% reuse rate.

These measurements demonstrate that low confidence at decision time is not equivalent to dispensability. The high reuse rates across all counterfactual eviction targets highlight how the subsequent reasoning trajectory frequently utilizes states on which the controller declined to commit.

Together, these results show that abstention is not merely a fallback for an uncertain controller; it preserves flexibility when relational utility remains unresolved. A System One controller compatible with System Two reasoning should not merely react quickly; it should make sparse commitments and defer judgment where the relational state of deliberation has not yet settled.

3.4 RELATIONAL UTILITY IN SYSTEM-2 DELIBERATION

Table 1 provides two motivating observations. First, conventional importance signals such as attention are poorly aligned with causal harm. Second, directly deleting states based on semantic judgments can be even more damaging than random deletion.

System-2 reasoning is not a monotonic accumulation of correct intermediate states. It revises earlier conclusions, revisits previously generated information, and can continue to rely on the trajectory through which an error was detected and corrected. A memory controller compatible with such deliberation should therefore not assume that the utility of a reasoning state is fixed.

We observe two complementary forms of dependence. First, utility is *set-conditional*: the value of one state can depend on which other states remain available,

$$U(E_i \mid C_t, S_t) \neq U(E_i \mid C_t \setminus I(E_j), S_t).$$

Table 3: **Observed re-reference after semantic abstention.** Cohorts represent the primary eviction targets (bottom 25%) as ranked by different policies at audit time. Despite being the lowest-scored candidates, their reuse rates remain substantial and comparable to the overall UNCONSTRAINED baseline. This demonstrates that (1) ambiguous reasoning states are frequently needed later, and (2) forcing immediate eviction based on dense scores risks discarding useful memory.

Audit-time cohort	Re-reference rate (%)			Delay med./p75
	≥1K	≥2K	≥4K	
UNCONSTRAINED (All)	40.3	39.6	38.7	1.5K / 3.9K
Counterfactual Eviction Targets (Bottom 25% of Unconstrained)				
Age (oldest)	49.6	49.7	49.4	1.3K / 3.4K
SnapKV (lowest score)	40.8	40.8	39.5	1.7K / 4.1K
Jev margin (lowest)	36.2	35.9	34.9	1.6K / 3.9K

Second, utility is *state-dependent*: the causal role of the same episode can change as deliberation moves to a new reasoning state,

$$U(E_i \mid C_t, S_t) \neq U(E_i \mid C_{t'}, S_{t'}).$$

Set-conditional utility. Our controlled OR-redundancy probe isolates the first effect. Two reasoning states encode substitutable copies of the same information: removing either copy alone has little effect, whereas removing both produces a large degradation of approximately -6.9 nat (Table 4a). Thus, the marginal value of either state depends on whether its substitute remains available. *What a state is worth depends on what else survives.* A fixed scalar importance assigned to each state independently cannot capture this interaction.

Correction relations and state dependence. Self-correction exposes a reasoning-specific form of relational utility. For natural correction trajectories, we identify a retracted source E_{ret} and the later episode E_{corr} that explicitly corrects it. At the first post-correction checkpoint, removing the retracted source changes next-window log-probability by -0.0070, compared with -0.0038 for the correcting episode and -0.0035 for a matched random UNCONSTRAINED control. Thus, an already superseded state can remain load-bearing for the ongoing computation: *semantically wrong does not mean physically useless.*

More importantly, this role changes as deliberation proceeds. The deletion cost of the retracted state falls from -0.0070 shortly after correction to -0.0041 after 1K tokens and -0.0016 after 4K tokens. The correcting episode shows the same trend ($-0.0038 \rightarrow -0.0023 \rightarrow -0.0011$), whereas matched random UNCONSTRAINED memory remains approximately flat ($-0.0035 \rightarrow -0.0032 \rightarrow -0.0033$). Protected memory remains substantially more disruptive to remove throughout ($-0.063 \rightarrow -0.055 \rightarrow -0.026$). *The same state's causal role changes as deliberation evolves.*

These results do not imply that semantically retired states should be deleted immediately. Rather, semantic retirement provides a *relative* eviction preference: a state may be textually superseded before it becomes computationally dispensable.

Taken together, the OR probe and correction interventions show that *System-2 reasoning does not present a stationary importance-ranking problem.* A compatible System-1 memory controller should therefore track relational state—whether a dependency remains active and whether a supersession has settled—and defer commitment when that relation is unresolved. This motivates our use of PROTECT, OBSOLETE, and UNCONSTRAINED rather than a dense per-state ranking.

3.5 MEMORY AND RUNTIME EFFICIENCY

Runtime and memory footprint. We measure efficiency on Qwen3-8B over 60 AIME 2025/26 problems using one isolated RTX PRO 6000 process per condition. All methods use the same decoding configuration, with $K = 4096$, $r = 64$, a 32K reasoning budget, and seed 0. We report decoding throughput over generated reasoning tokens, final and time-averaged KV footprint, eviction-path overhead accumulated per trace, and external audit cost.

Table 4: **Relational utility in reasoning memory.** **(a)** OR redundancy demonstrates set-conditionality: either copy is individually dispensable, but removing both is costly. **(b)** In correction trajectories, deletion effects change as deliberation progresses. Values are changes in mean next-window log-probability per token relative to the intact cache; more negative values indicate greater disruption.

(a) Set-conditional OR probe

Removed state(s)	$\Delta \log p$
E_1 only	0.0
E_2 only	0.0
E_1, E_2	-6.9

What a state is worth depends on what else survives.

(b) Correction relation across deliberation

Deleted	$\geq$64	$\geq$1K	$\geq$4K
Retracted	-0.0070	-0.0041	-0.0016
Correction	-0.0038	-0.0023	-0.0011
Random U	-0.0035	-0.0032	-0.0033
PROTECT	-0.063	-0.055	-0.026

The same state's causal role changes as deliberation evolves.

Table 5: **Memory and runtime efficiency.** All compressed methods operate under the same KV budget. SPARE reduces the final KV footprint by $3.8\times$ relative to Full Attention while increasing decoding throughput. Asynchronous auditing adds negligible online overhead beyond random allocation, at an external audit cost of approximately \$0.05 per trace.

Cache policy	**tok/s** ↑	**Eviction-path overhead** ↓ **(s/trace)**	**Final KV** ↓ **(GB)**	**Avg. KV** ↓ **(GB)**	**Audit cost** ↓ **($/trace)**
Full attention	53.1	–	2.30	1.09	–
Random Attention	59.6	2.5 (base)	0.60	0.49	–
SnapKV	43.0	5.1 (+2.6)	0.60	0.49	–
R-KV	46.1	11.1 (+8.7)	0.60	0.49	–
VaSE	43.4	6.6 (+4.2)	0.60	0.49	–
SPARE-Jev (sync)	59.6	28.6 (+26.2)	0.60	0.49	\$0.056
SPARE-Jev (async)	58.2	**2.7 (+0.2)**	0.60	0.49	\$0.052

As shown in Table 5, compressing the KV cache reduces the final footprint from 2.30 GB to 0.60 GB ($3.8\times$), while the time-averaged footprint decreases from 1.09 GB to 0.49 GB. The smaller cache also improves decoding speed, from 53.1 tokens/s under Full Attention to 58.2–59.6 tokens/s for the SPARE variants. Synchronous auditing incurs substantial blocking overhead (28.6 s/trace), whereas asynchronous auditing overlaps audit inference with decoding and adds only 2.7 s/trace, just 0.2 s beyond random allocation. The external audit cost is approximately \$0.05 per trace.

Semantic guidance under memory pressure. Existing KV-eviction methods can be viewed as System-1-like memory controllers: they make fast online decisions from local signals such as attention, recency, or token-level importance. Such heuristics are effective when memory is relatively abundant, but become increasingly brittle as the budget tightens, because the future utility of a state during deliberative reasoning may not be reflected by its instantaneous local score.

Our design keeps this fast eviction path, but augments it with semantic guidance about which states remain useful to the ongoing reasoning process. This separates *how fast to evict* from *what should be preserved*: eviction remains lightweight and online, while semantic judgments provide a higher-level signal aligned with the longer-horizon dependencies of System-2-style deliberation.

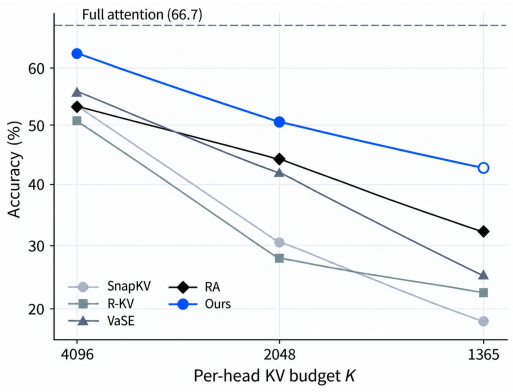

Figure 2: **Accuracy under shrinking KV budgets.** AIME 2025/26 on Qwen3-8B (seeds 0–1); full attention achieves 66.7%.

The shrinking-budget experiment makes this distinction most visible. As K decreases, conventional eviction policies suffer substantially larger accuracy drops, whereas Ours degrades more gracefully (Figure 2). This suggests that semantically guided eviction is better suited to supporting deliberative reasoning under bounded working memory, rather than merely optimizing local cache salience.

4 Related Work

4.1 General & Semantic KV Cache Compression

KV-cache compression exploits attention history (Liu et al., 2023; Zhang et al., 2023; Xiao et al., 2024), prompt observation (Li et al., 2024; Tang et al., 2024), and architectural non-uniformity (Cai et al., 2024; Fu et al., 2025; Tang et al., 2025; Behnam et al., 2025). To capture non-stationary relevance, methods periodically update importance (Xu et al., 2025; Ahn et al., 2026) or operate on semantic blocks (Liu et al., 2025; Chen et al., 2025). Unlike these settings, our cache growth is dominated by a self-generated reasoning trajectory. SPARE utilizes semantic units but uniquely permits semantic abstention (UNCONSTRAINED) rather than forcing a retention preference for every block.

4.2 Cache Management for Long Reasoning

Extended deliberation requires specialized eviction via redundancy-aware scoring (Cai et al., 2025), value-magnitude protection (Chang et al., 2026), delayed eviction (Zhang et al., 2026b), thought-adaptive compression (Ramachandran et al., 2026), or query-key structural estimators (Mao et al., 2026). Alternatively, Random Attention (Wang et al., 2026) shows prompt-protected uniform eviction is highly competitive. SPARE builds on this stochastic baseline, guiding it only when the evolving trajectory provides sufficient evidence to impose a semantic constraint, avoiding dense rankings of the entire cache.

4.3 Model-Driven and System-One Decision Control

Learned models can serve as memory controllers, such as SideQuest managing retrieved contexts (Kariyappa & Suh, 2026). However, managing a self-generated reasoning trace introduces a unique difficulty: an incorrect or superseded intermediate state can remain computationally useful as part of the trajectory that detected and corrected it. Our controller addresses this using the emerging paradigm of *System One Models*—specialized for fast, structured decisions over unstructured states (Almeida, 2026; Kwok et al., 2026). We use this distinction operationally: the base model performs extended System-Two-like deliberation, while the auxiliary controller makes lightweight structured judgments. Similar to JEV-as-a-Judge (Li et al., 2026), which leverages confidence to escalate uncertain cases, SPARE represents uncertainty explicitly by withholding a preference and leaving unresolved memory to a simple stochastic policy.

5 Conclusion

We studied KV cache eviction as System-One memory control for System-Two reasoning. Our results show that reasoning-memory utility is relational and evolves with deliberation, making premature semantic commitments unreliable. SPARE therefore uses selective, revisable constraints—protecting clear dependencies, retiring settled supersessions, and abstaining otherwise—while leaving physical eviction to a lightweight stochastic policy. Across long-reasoning benchmarks, this design improves fixed-budget memory allocation while preserving an efficient online eviction path. More broadly, effective memory control for deliberative reasoning may depend less on ranking every state precisely than on knowing when to constrain memory and when to leave the decision open. As reasoning models tackle increasingly complex tasks, managing their working memory dynamically will become just as critical as the reasoning capabilities themselves. Future directions include co-optimizing the System-One controller directly with the base model and extending this relational memory framework to broader multi-step agentic workflows.

AI USE STATEMENT

In this work, we used generative AI tools for editing and refining the manuscript text. We have not used generative AI tools for writing the core algorithmic logic, proving mathematical theorems, or manually verifying the benchmark answers, and data generation tasks are not applicable to this work. Additionally, we used the System I model Jev (typesafe/jev-1.13) as an auxiliary agent to assign semantic constraints (PROTECT, OBSOLETE, UNCONSTRAINED) during the online auditing phase of our proposed method. We have reviewed all AI-assisted work; the code for the auditing loop was verified and tested for correctness by the authors, and the qualitative behavior of the Jev model's constraints was manually inspected for consistency. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our research focuses on improving the efficiency of key-value (KV) cache memory allocation during the inference of large language models for reasoning. To the best of our knowledge, this work does not introduce new ethical risks such as the collection of sensitive human subject data, the release of personally identifiable information, or the deployment of inherently harmful applications. The benchmark datasets used in our evaluation (MATH-500, AIME, HMMT) are standard, publicly available resources for mathematical problem-solving. By reducing the memory footprint required for long chains of thought, our method has the potential to democratize access to advanced reasoning models by lowering the hardware requirements for inference. We declare no potential conflicts of interest or specific compliance issues related to this submission.

REPRODUCIBILITY STATEMENT

We are committed to ensuring the reproducibility of our proposed method. To facilitate this, we provide detailed descriptions of the cache management parameters, audit scheduling rules, and constraint thresholds in Appendix. The exact formulations for our semantic constraints and the stochastic eviction policy are clearly defined in Section 2 of the main text. For the experimental evaluation, we list the exact model versions (Qwen3-8B, Qwen3-4B, Phi-4-reasoning), the external LLM model (Jev), and the specific reasoning benchmarks used (Section 3.1). Furthermore, we plan to make the source code repository containing the implementation of the our cache controller, the asynchronous auditing loop, and the evaluation scripts publicly available upon acceptance of this paper.

REFERENCES

Jinwoo Ahn, Ingyu Seong, Akhil Kedia, Junhan Kim, Hyemi Jang, Kangwook Lee, and Yongkweon Jeon. Lookaheadkv: Fast and accurate kv cache eviction by glimpsing into the future without generation. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 71406–71430, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/746222c1871d3fa7a94bdacabc34e26c-Paper-Conference.pdf.

Diogo Almeida. Introducing system one models & jev. TypeSafe AI Blog, September 2026. URL https://typesafe.ai/blog/introducing-system-one-models-and-jev.

Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. RocketKV: Accelerating long-context LLM inference via two-stage KV cache compression. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 3358–3392. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/behnam25a.html.

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

Zefan Cai, Wen Xiao, Hanshi Sun, cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Animashree Anandkumar, Abedelkadir Asi, and Junjie Hu. R-kv: Redundancy-aware kv cache compression for reasoning models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 60980–61005. Curran Associates, Inc., 2025. doi: 10.52202/085713-2038. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/57e0358aed314784cad14a26e1eba642-Paper-Conference.pdf.

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

Jinhan Chen, Jianchun Liu, Hongli Xu, Xianjun Gao, and Shilong Wang. Sablock: Semantic-aware kv cache eviction with adaptive compression block size. *arXiv preprint arXiv:2510.22556*, 2025.

Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. Not all heads matter: A head-level kv cache compression method with integrated retrieval and reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 99269–99290, 2025.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, 2011. ISBN 978-0374275631.

Sanjay Kariyappa and G. Edward Suh. Sidequest: Model-driven KV cache management for long-horizon agentic workloads. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=wNjm4WVkOW.

Jacky Kwok, Hangoo Kang, Tarun Suresh, Jon Saad-Falcon, Marco Pavone, Christopher Ré, and Azalia Mirhoseini. Contrastive language models: A system one model for fast and generalizable decision-making, 2026. URL https://contrastive-lm.notion.site. Notion Blog.

Yubo Li, Yidi Miao, Ramayya Krishnan, and Rema Padman. Jev-as-a-judge: Accept when confident, escalate when unsure. *arXiv preprint arXiv:2609.26550*, 2026.

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 A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970. Curran Associates, Inc., 2024. doi: 10.52202/079017-0722. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/28ab418242603e0f7323e54185d19bde-Paper-Conference.pdf.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Bo Li, Xuming Hu, and Xiaowen Chu. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *Advances in Neural Information Processing Systems*, 38, 2025.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

Weian Mao, Xi Lin, Wei Huang, Yuxin Xie, Tianfu Fu, Bohan Zhuang, Song Han, and Yukang Chen. Triattention: Efficient long reasoning with trigonometric kv compression. *arXiv preprint arXiv:2604.04921*, 2026.

Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. Thinkv: Thought-adaptive kv cache compression for efficient reasoning models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 110072–110106, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/b2b502c3629beadda06311386d2c6f73-Paper-Conference.pdf.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Danning Ke, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads. In *International Conference on Learning Representations*, volume 2025, pp. 16632–16646, 2025.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Heng Wang, Jielin Qiu, Wenting Zhao, Cheng Qian, Liangwei Yang, Jiawei Han, Heng Ji, Silvio Savarese, Shelby Heinecke, and Huan Wang. Random attention: Rethinking kv cache eviction for efficient reasoning. *arXiv preprint arXiv:2609.03430*, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. *URL https://arxiv. org/abs/2309.17453*, 1(5.1), 2024.

Fangyuan Xu, Tanya Goyal, and Eunsol Choi. Refreshkv: Updating small kv cache during long-form generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24878–24893, 2025.

Duzhen Zhang, Zhong-Zhi Li, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Xiuyi Chen, Yingying Zhang, Fei Yin, Jiahua Dong, Zhijiang Guo, Le Song, and Cheng-Lin Liu. From system 1 to system 2: A survey of reasoning large language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 48(3):3335–3354, 2026a. doi: 10.1109/TPAMI.2025.3637037.

Haoyue Zhang, Hualei Zhang, Xiaosong Ma, Jie Zhang, and Song Guo. Lazyeviction: Lagged kv eviction with attention pattern observation for efficient long reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36335–36352, 2026b.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information*

Processing Systems, volume 36, pp. 34661–34710. Curran Associates, Inc., 2023. doi: 10.52202/075280-1506. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

A Experimental Details

A.1 Models and Decoding

All three base models (Qwen3-8B, Qwen3-4B, Phi-4-reasoning) are loaded from their public checkpoints in bf16 with the scaled-dot-product attention kernel, and decoded one sequence at a time with an explicit per-step position index so that eviction can act on a physical, per-head cache whose retained positions are not contiguous. The prompt is a single user turn rendered with the model's chat template with the generation prompt appended and thinking enabled. Tokens are sampled at temperature 0.6 for Qwen models and 0.8 for Phi-4-reasoning, with nucleus $p{=}0.95$ (probabilities are sorted, the tail beyond cumulative mass 0.95 is zeroed, the remainder renormalised, and one token drawn). The random generator for the sampler is seeded once per (problem, seed) pair with

$$\text{seed}_{\text{gen}} = 900 + 1000 \cdot s + \big(\text{crc32}(\text{problem_id}) \bmod 1000\big),$$

where $s \in \{0, 1, 2, 3\}$ is the run index. The generator used by stochastic eviction (Random Attention, SPARE and the VaSE sampling step) is seeded with s alone. Both seeds are identical across methods, so for a given (problem, s) all methods produce the same tokens until their caches first differ, i.e. until the first eviction.

A.2 Task Prompts

For the mathematics tasks the user message is the problem statement followed by

```
Please reason step by step, and put your final answer within
\boxed{}.
```

A.3 Datasets

- **MATH-500**: the 500-problem test split of `HuggingFaceH4/MATH-500`. Identifiers `m000`–`m499`.
- **AIME 2024**: 30 problems from AIME 2024. Answers are integers.
- **AIME 2025/26**: AIME 2025 I and II and AIME 2026 I and II from MathArena, 15 problems each, 60 in total. Identifiers `a25_01`–`a25_30` and `a26_01`–`a26_30`. Answers are integers.
- **HMMT 2025**: the February 2025 and November 2025 HMMT sets from MathArena, 30 problems each, 60 in total. Identifiers `h25_*` and `hn25_*`.

All datasets are read from local Parquet snapshots so that compute nodes need no network access.

A.4 Budgets and Run Counts

The per-head budget K is 1024 (MATH-500) and 4096 (AIME 2024, AIME 2025/26, HMMT 2025), and the recency buffer is $r{=}64$ everywhere. Each (problem, method) pair is run with seeds 0–3 (4 seeds) on AIME 2024, AIME 2025/26, and HMMT 2025, and with seeds 0–1 (2 seeds) on MATH-500.

A.5 Thinking Cap and Forced Closure

Thinking is capped at $32{,}768$ generated tokens and the answer at $2{,}048$. If the thinking cap is reached without a `</think>` token, the string

```
\n</think>\n\nGiven the work above, I will state my best
answer directly.\n\n**Final Answer:** $\boxed{
```

is fed to the model, and at most 48 tokens are decoded greedily, stopping at the first token containing a closing brace. The trace is marked *forced* and graded like any other. If the model closes its thinking naturally but its answer contains no `\boxed{`, the shorter string `\n\n**Final Answer:** $\boxed{` is appended and the same 48-token greedy completion is run; such traces are not counted as forced. Cache eviction continues during the answer phase and during both completion procedures.

A.6 GRADING

The answer is the content of the last `\boxed{...}` in the answer text, extracted with brace matching. For the mathematics tasks a prediction is correct if it equals the reference as a string or if `math_verify` judges the two equivalent for any of the four combinations of wrapping either side in `$...$`. A trace with no extractable box is wrong.

A.7 EVICTION ENGINE

The cache is the standard dynamic KV cache of the inference library; each layer stores rectangular key and value tensors, and a separate integer tensor records, for every KV head, the absolute position held in each slot. Because every head pins the same prompt and the same recency buffer, all heads of a layer hold the same number of slots, which allows the per-head rule to be executed as a single vectorised operation per layer.

Trigger and target. Prompt positions $[0, p)$ and the $r{=}64$ most recent generated positions are never candidates. An eviction is triggered when a head holds at least $K{+}2r$ generated positions; it then keeps exactly K candidates (plus the r buffer positions and the prompt), so that evictions occur every r generated tokens once the budget is reached. Survivors keep their absolute positions and their rotary phases; key and value rows of evicted slots are physically removed by a gather, and nothing is ever recomputed.

Selection rules. For each head, with $u_{h,i} \sim \mathrm{Uniform}(0, 1)$ drawn independently per candidate at every eviction:

- **Random Attention**: keep the top-K candidates by $u_{h,i}$.
- **SPARE**: keep the top-K by $u_{h,i} + 10 \cdot \mathbb{1}[\ell(i){=}\textsc{Protect}] - 10 \cdot \mathbb{1}[\ell(i){=}\textsc{Obsolete}]$, which realises the three tiers with a uniform lottery inside each tier. With obsolete retention $\rho_O > 0$, each head independently re-labels a Bernoulli(ρ_O) subset of its OBSOLETE candidates as UNCONSTRAINED before scoring; $\rho_O{=}0$ in all main runs.
- **Oldest-first** (analysis baseline): score is the position itself, i.e. keep the newest K generated tokens.
- **Score-based baselines**: see below.

A.8 SCORE-BASED BASELINES

SnapKV, R-KV and VaSE are re-implemented from the reference code released with Random Attention so that they share the trigger, target, buffer, prompt pinning, seeds and closure with every other method.

Query buffer. A forward pre-hook on every attention module recomputes the post-rotary query of the current token and stores it in a circular buffer of the last $r{=}64$ queries per layer, so that scoring uses the same observation window as the recency buffer.

Attention score. For each KV head, the softmax attention of the 64 buffered queries of its GQA group over the candidate keys is averaged over the queries, smoothed along the position axis with an average-pooling kernel of width 5, and averaged over the group's query heads.

SnapKV keeps the top-K candidates by this score.

R-KV normalises the candidate keys, computes their pairwise cosine similarity with the diagonal zeroed, takes the row mean and softmax as a redundancy score red_i, and keeps the top-K by $\lambda\,\mathrm{attn}_i - (1-\lambda)\,\mathrm{red}_i$ with $\lambda{=}0.5$.

VaSE computes the value range $\max_d v_{i,d} - \min_d v_{i,d}$ of every candidate, treats the first four candidate positions as sinks with infinite range, pins the $n_{\text{large}} = K/4$ candidates of largest range, and fills the remaining $K - n_{\text{large}}$ slots by sampling without replacement from the attention score (as a probability distribution with the pinned positions zeroed) using the eviction generator.

A.9 SPARE CONFIGURATION

The configuration below was fixed before the main runs and used unchanged for all models, tasks and seeds.

episode length bounds $m_{\min}$ / $m_{\max}$	128 / 256 tokens, closed at a blank line
raw-text window w	1536 tokens
audit clock T	1024 generated tokens
cue-triggered audit spacing $T_{\min}$	256 generated tokens
first audit	first episode boundary after the generated cache reaches K
protection cap ρ_P	0.5 (whole episodes, P(PROTECT) descending order)
retirement confidence gate	0.85
obsolete retention ρ_O	0
audit execution	asynchronous (decoding continues during agent inference)
agent	Jev (`typesafe/jev-1.13`), decision-only endpoint, default parameters

The input construction, the trigger rules and the label-conversion rules are given verbatim in Appendix B. Because audits run asynchronously with token generation, the labels in force at every eviction are those of the most recently completed audit.

A.10 HARDWARE AND SOFTWARE

All runs, including decoding speed and efficiency overhead measurements, were executed on isolated NVIDIA RTX PRO 6000 (48 GB) GPUs, one trace per GPU. Software: PyTorch 2.10 with CUDA 12.8, `transformers` 5.3.0, `datasets` 4.8.3, `math-verify` for grading. Agent latency (Section 3.5) is measured per API call and does not depend on the GPU.

A.11 MEMORY ACCOUNTING

Qwen3-8B has 36 layers with 8 KV heads of dimension 128; in bf16 each cached token occupies $36 \times 8 \times 128 \times 2 \times 2 = 147{,}456$ bytes (144 KB). A full-attention AIME trace ends with ≈ 18.1k cached tokens (≈ 2.6 GB per sequence), against ≈ 4.35k tokens (≈ 0.63 GB) under any eviction method at $K{=}4096$; the corresponding figures are 20.0k versus 4.3k on HMMT, and 5.6k versus 1.2k on MATH-500.

B AGENT PROMPT AND AUDIT PROTOCOL

This appendix provides the exact instructions given to the auditing agent, the input construction, and the deterministic rules that convert its responses into the PROTECT, OBSOLETE, and UNCONSTRAINED labels used by the SPARE eviction policy. The controller's input construction and rules were held fixed across all models, tasks, budgets, and seeds evaluated in this work. Note that in the episode headers of the state and in the code, the label `NEUTRAL` is used equivalently to the UNCONSTRAINED state described in the main text.

B.1 JEV CONTROLLER: INPUT CONSTRUCTION, QUESTIONS, AND LABEL RULES

The System One model used in our experiments, Jev (`typesafe/jev-1.13`), is invoked via the OpenRouter System One endpoint (`/api/v1/systemone`) with a decision-only `{state, questions}` payload. It operates without system/user messages, generation schemas, or episode summaries, and observes purely text-based context.

State Construction The `state` string consists of the problem statement, a budget warning (`"hard budget of about {K} reasoning tokens; tokens not protected will be evicted at random and never come back"`), the full text blocks of all closed episodes formatted as `[E{id} | {n} tokens | previous label: ...]`, and concludes with the recent raw text block `"MOST RECENT RAW REASONING (last 1536 tokens)"`. If the input exceeds an input budget of approximately 24k tokens, an aggressive

truncation sequence is applied: labeled episodes and the eight newest episodes are preserved, while older UNCONSTRAINED episodes are shrunk to 96+32 head/tail tokens, or their headers only if necessary. If the provider rejects the request, the state is rebuilt with the budget sequentially reduced to 70%, 45%, 25%, and finally 3k tokens.

Questions and Criteria Up to 120 questions are sent per request (split and re-sent if exceeded). For each episode, the question instruction is:

```
"For episode E{id}: which label applies, judged against
the CURRENT state of the reasoning (the most recent raw
reasoning)?"
```

The three criteria provided as choice options are verbatim:

- **PROTECT:** `"The model will still have to READ this episode again to finish correctly: an established intermediate result or value the final answer depends on, the current plan/sub-goal, the most recent correction and its corrected value, or a problem constraint the model re-derived. Not restatements, passed sanity checks, or exploration that led nowhere."`
- **OBSOLETE:** `"The content of this episode was EXPLICITLY retracted or replaced by a later, SETTLED correction (a superseded assumption or value, an abandoned branch, or results derived from them). Not if the model is still going back and forth or the correction is tentative; never the correction itself or the problem statement."`
- **UNCONSTRAINED:** `"Neither: ordinary reasoning that need not be re-read and was not retracted; leave it to random eviction."`

Label Rules The endpoint returns a predicted choice, probabilities, and confidence *per episode*. The labels ℓ_τ are assigned as follows:

1. **PROTECT:** Episodes where the chosen label is PROTECT are sorted in descending order of P(PROTECT) (with episode ID as an ascending tiebreaker). They are admitted via whole-episode first-fit under the capacity cap $\rho_P K = K/2$.
2. **OBSOLETE:** From the remaining episodes, those where the choice is OBSOLETE and the *per-episode* confidence is ≥ 0.85 are marked OBSOLETE.
3. **UNCONSTRAINED:** All other episodes are designated UNCONSTRAINED.

These labels are broadcast to all token positions within the episode interval. Labels are not retroactive. Under the asynchronous auditing variant, these labels are installed immediately upon audit completion, naturally overlapping with ongoing token generation without stalling the decoder.

B.2 AUDIT TRIGGER CONDITIONS

Audits are strictly issued at episode boundaries. An episode closes when its length reaches $m_{\max} = 256$ tokens, or when it has at least $m_{\min} = 128$ tokens and the decoded text of its last two tokens contains a double newline (`\n\n`). At a boundary, an audit is triggered if the generated cache holds at least K tokens and one of the following conditions is met:

- It is the first episode boundary after the generated cache reaches K.
- At least $T = 1024$ tokens have been generated since the previous audit.
- The recently closed episode contains a revision cue, and at least $T_{\min} = 256$ tokens have been generated since the previous audit. Revision cues are matched case-insensitively and include: `wait`, `actually`, `however`, `mistake`, `wrong`, `incorrect`, `reconsider`, `instead`, `let me check`, `i was wrong`, `assumption fails`.

No audits are issued during the final answer phase (e.g., after `</think>`).