KV Cache Directives for Agentic Inference

Anonymous ACL submission

Abstract

Agentic LLMs (Yao et al., 2023; Yang et al., 2024) break the append-only KV-cache assumption that makes prefix caching correct by construction: conversations evolve through policy-driven editing, producing two cache problems. Reuse across position shifts is addressed by recent work (Ma et al., 2026). Policy-driven *mutation* — removing or replacing a cached span and continuing to decode without re-prefilling the remainder — has no primitive: existing harnesses re-prefill from the edit point (Anthropic, 2025), and selective-recompute methods (Yao et al., 2024; Hu et al., 2025) address cache fusion across reuse, not policy-driven removal.

We introduce Leyline, a serving-side primitive for policy-driven cache editing. A declarative `DIRECTIVE` 4-tuple lets a policy describe *what* to edit and *declare* whether the edit should amortize the prefill (in-place splice) or force prefix-trimmed re-prefill for semantic-forgetting use cases. The kernel applies the in-place edit via a closed-form RoPE-rotation correction. We validate Leyline on two complementary legs. *Mechanism*: at fixed policy, the splice kernel lifts replay cache-hit by $+11.2$ pp over the standard radix baseline on a three-arm SGLang microbenchmark at $\approx$ 17K-token prompts, reproducing across concurrencies $C \in \{1, 4, 8, 16\}$ on DeepSeek-V2-Lite and JoyAI-LLM Flash, with end-to-end latency reduced by up to 241 ms at C=8. *Policy*: at fixed mechanism (standard re-prefill), a 10-line truncation policy routed through the directive interface lifts agentic solve rate by $+14.3$ pp over a no-truncation baseline on debug-gym mini_nightmare (9 tasks $\times$ 4 seeds, JoyAI-LLM Flash). The mechanism enables the policy class without prefill recompute cost; the policy class delivers the deployment win. Our contribution is the directive abstraction and its mode-declaring contract with the rotation kernel borrowed and credited as such.

1 Introduction

Agentic LLMs create two distinct cache problems: reuse across natural position shifts, and policy-driven mutation. Prior work addresses the first; this paper addresses the second.

KV cache management research has been organized around one workload assumption: prompts arrive once and the cache grows monotonically. The major optimizations in this lineage — prefix caching (Kwon et al., 2023; Zheng et al., 2024), streaming attention (Xiao et al., 2024), attention-based eviction (Zhang et al., 2023; Li et al., 2024; Liu et al., 2023; Ge et al., 2024) — are correct *because* the conversation only moves forward. The agentic era (Yao et al., 2023; Yang et al., 2024) breaks this assumption in two distinct ways, each producing a different cache-management problem.

Problem 1: position-invariant reuse. A multi-turn coding or web-research agent re-renders its conversation history every turn. As new tool outputs and retrieved documents land, previously-seen content shifts to new absolute positions. Exact-prefix matching cannot recover these shifted-but-identical chunks, even though their KV would be reusable if the cache were addressed by content rather than position. Recent MLA-native work (Ma et al., 2026) solves this reuse problem.

Problem 2: policy-driven mutation. Distinct from reuse, an agent or policy may need to direct the serving system to *actively* remove or replace cached content: clear a failed tool call's output, summarize a stale investigation block to a stub, truncate aged tool outputs to free working memory. These are deliberate edits, not natural shifts. Today, agentic harnesses handle them by editing the message list and re-prefilling, since no primitive exists for policy-issued cache editing — the cost is the full prefix-recomputation that prefix caching was meant to amortize, and even small per-request

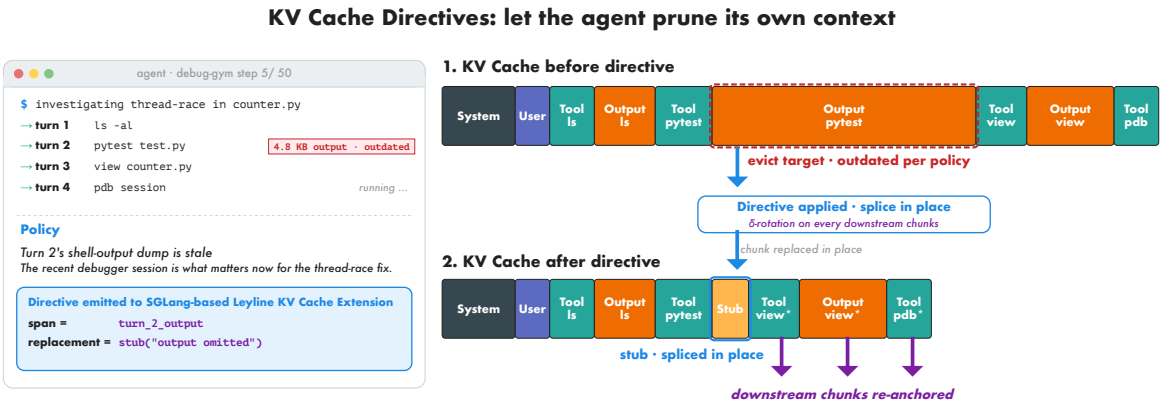

Figure 1: **Teaser.** A coding agent emits an explicit directive over its KV cache, expressed as a (span, replacement) pair. The serving stack splices the cache in place — the old chunk is replaced by a short stub, downstream cached slots are re-anchored via a closed-form δ-rotation, and the radix prefix is preserved across the edit. The agent supplies *what* to evict; Leyline supplies *how* to splice it without re-prefill.

invariant breaks can collapse throughput in production (Cole, 2026).

Our solution. We introduce Leyline, a serving-side primitive for policy-driven cache editing. The interface is a declarative (span, replacement) directive: the policy describes *what* to edit; the kernel applies the edit in place via a closed-form δ-rotation — when a splice changes downstream absolute positions, cached K_{pe} values are rotated by the splice delta to restore attention-math correctness without re-prefill.

What is borrowed, what is new. The numerical kernel — δ-rotation under RoPE closure $R(a)R(b)=R(a+b)$ on the MLA K_{pe} band — is from Ma et al. (2026), where it was triggered *reactively* by a content-hash match on naturally shifted but content-identical chunks (the reuse problem). Leyline's contribution is what surrounds the kernel: (a) the framing that identifies mutation as a distinct sub-problem from reuse, (b) the declarative directive abstraction that lets a policy invoke the same correction *actively* as an explicit edit, (c) a 4-tuple contract with declared semantic mode that lets a policy choose between in-place amortization and forgetting-mode re-prefill at the API level (Section 3.1), and (d) the radix-cache integration that preserves the unchanged prefix subtree across the edit. The kernel is the recycled engineering substrate; the directive layer is the new interface boundary the paper argues post-agentic serving stacks need.

Contributions. (1) A precise distinction between the **reuse** and **mutation** problems in agentic KV cache management, locating prior work and identifying mutation as the open sub-problem.

(2) The **directive abstraction**: a 4-tuple (span, replacement) interface with declared semantic mode and a pluggable policy layer that decouples *what* to edit (policy) from *how* to splice it (kernel), composed over the borrowed δ-rotation correctness mechanism without recomputing the unchanged prefix.

(3) A **semantic-edit channel as a new serving-stack interface**: the directive sits above block-level KV storage as a span-level edit layer, decomposing each agent-emitted edit into block-level operations the storage layer schedules, and identifies the cache-edit channel as the missing post-agentic interface boundary (Section 7).

(4) **Two-leg validation**:

(i) Mechanism — a three-arm message-edit microbenchmark on the live SGLang scheduler (cache-off / standard radix / Leyline splice) at $\approx$ 17 K-token prompts shows the splice arm lifting replay cache-hit rate by $+11.2$ pp over the standard radix baseline uniformly across concurrencies $C \in \{1, 4, 8, 16\}$, reproducing on DeepSeek-V2-Lite and JoyAI-LLM Flash, with the e2e latency win peaking at -241 ms at $C{=}8$ (Section B).

(ii) Policy — a 10-line truncation policy routed through the directive interface lifts agentic solve rate by $+14.3$ pp over a no-truncation baseline across 9 debug-gym (Yuan et al., 2025) mini_nightmare tasks $\times$ 4 seeds on JoyAI-LLM

Flash (Section 5). Read together: the mechanism enables the policy class without prefill recompute cost; the policy class delivers the deployment-cell win. Kernel correctness is separately established on a constructed single-prompt microbenchmark and a cross-architecture MLA replay across four models with Moonlight as the row where references diverge non-trivially (Section 4); extending the kernel to GQA appears feasible via a boundary-recomputation pass (Hu et al., 2025) and is sketched in Section 6.

2 Background: where existing systems fit

Table 1 positions representative systems against the two sub-problems plus three properties any agentic-era cache primitive needs: correctness after edit, cache-reuse after edit, and turn-granular operation.

The first row trades cache reuse to get correctness on mutation. The second row addresses neither sub-problem. The third (LMCache) is the production storage and movement layer for vLLM and SGLang (Liu et al., 2025) but operates below the semantic-edit level. The fourth and fifth (CacheBlend, EPIC) are the MHA-side analogs to Irminsul's MLA-side δ-rotation: both address non-prefix reuse on MHA/GQA via selective recomputation rather than positional correction. CacheBlend dynamically picks high-deviation tokens layer-by-layer (Yao et al., 2024); EPIC statically picks each chunk's first k tokens, motivated by the attention-sink effect at chunk boundaries (Hu et al., 2025). The sixth trades correctness for speed. The seventh (Irminsul) handles the MLA reuse half; the eighth (Leyline) contributes the mutation half, with CacheBlend/EPIC-style boundary recomputation as the natural mechanism-layer extension we envision for GQA (Section 6).

Layered relationship to LMCache. LMCache and Leyline operate at different layers and compose: LMCache's verbs (Pin, Clear, Move, Compress, Lookup) act on KV *blocks* in physical storage tiers (Liu et al., 2025); Leyline's directive acts on *semantic* spans of the prompt. A directive "replace this stale tool-output span with a stub" decomposes into block-level operations (free, allocate, K_{pe} rotate, re-index) that LMCache schedules across tiers — agent intent $\rightarrow$ semantic directive ($Leyline$) $\rightarrow$ block operations (LMCache) $\rightarrow$ physical tier movement. Leyline supplies what LMCache does not: position-encoding correctness across a semantic edit and a span-level interface. Adjacent work on tensor-encoded KV transfer (Liu et al., 2024a) and 2-bit KV quantization (Liu et al., 2024b) is orthogonal to position correctness.

2.1 Concurrent work on agentic context management

Two recent systems offer signal-side contributions complementary to Leyline's mechanism: SideQuest (Kariyappa and Suh, 2026) fine-tunes a detector that emits `del_cursors` commands evicting stale tool-call/response pairs; LoopGuard (Xu et al., 2026) detects decoder-side degeneration symptoms (action repetition, top-1 streak, gzip compressibility) and intervenes by resampling. Either signal can drive Leyline directives without retraining the kernel or the detector. A full taxonomy of useful agentic-era eviction signals is follow-up work.

Framework-engineering signals. Four open vLLM RFCs each request a slice of cache control at the layer above the storage backend, with no maintainer-accepted abstraction; Leyline's directive is the missing layer (full thread analysis: App. L).

3 Leyline: a directive abstraction for policy-driven cache editing

Leyline is a serving-side abstraction layer between the agent's policy and the KV cache. The contributions of this section are three: a declarative directive interface (Section 3.1) that decouples policy signal from kernel mechanism, a serving-stack integration that preserves the radix prefix across edits (Section 3.3), and a thin Python policy interface that any span-emitting policy can drive (Section 3.4). The kernel-level correctness mechanism — the δ-rotation rule that re-anchors cached K_{pe} values after a position shift — is recapped in Section 3.2; Leyline is what turns it into a usable primitive for explicit policy-driven editing.

Three properties from Section 2 fix the success criteria, none currently met by an existing system: (i) *signal-agnosticism* — the primitive accepts anything expressible as a set of spans, without privileging the signal source (Kariyappa and Suh, 2026; Xu et al., 2026); (ii) *positional replay-equivalent correctness* — after an edit, downstream attention math is well-formed at the new positions (the exact contract is made precise in Section 3.1); and (iii) *cache-friendliness* — the edit localizes to the spliced span without invalidating the radix prefix.

System	Targets	Correct?	Reuse?	Granularity
Vendor APIs (read / boundary controls; no in-place edits)				
Anthropic prompt caching†	read-side reuse	n/a	5-min / 1-h TTL	breakpoint
Anthropic `clear_tool_uses`†	mutation	re-prefill	no (radix gone)	turn
OpenAI prompt caching*	read-side reuse	n/a	prefix only	token-prefix
Google context caching‡	read-side reuse	n/a	cached-object	object
Research systems and open-source serving libraries				
H2O / SnapKV / StreamingLLM§	neither	forward-only	n/a	token
LMCache (Liu et al., 2025)	storage	n/a	prefix	256-tok block
CacheBlend (Yao et al., 2024)	**reuse** (MHA)	dynamic 15% recompute	yes	chunk
EPIC / LegoLink (Hu et al., 2025)	**reuse** (MHA/GQA)	static k-first recompute	yes	chunk
In-place K/V splice (naïve)	mutation	no (stale RoPE)	yes	arbitrary
Irminsul (Ma et al., 2026)	**reuse** (MLA)	hash + δ-rotation	yes	CDC
Leyline (this paper)	**mutation** (MLA)	**directive + δ-rotation**	**yes (radix kept)**	**turn**

Table 1: Agentic-era cache design space split by sub-problem. **Targets**: which sub-problem the system addresses. **Correct?** and **Reuse?**: correctness after edit and cache reuse after edit; method named in lieu of bare "yes," `n/a` where the question does not apply. The top group is vendor APIs surfaced by major LLM providers: these expose either read-side reuse hints (Anthropic / OpenAI / Google) or the equivalent of a turn-granular mutation that incurs re-prefill (Anthropic `clear_tool_uses`); none expose an in-place semantic-edit primitive. The bottom group is research and open-source systems. Irminsul addresses the reuse half via content-addressed matching plus a δ-rotation correction for MLA; Leyline addresses the mutation half via a directive abstraction over the same δ-rotation foundation. Both are needed; both compose; this paper contributes the mutation half. †Anthropic (2024, 2025). *OpenAI (2024). ‡Google (2024). §Zhang et al. (2023); Li et al. (2024); Xiao et al. (2024).

3.1 The directive: (span, replacement)

A directive is a 4-tuple $D = (s_{\text{start}}, s_{\text{end}}, R, m)$, where $[s_{\text{start}}, s_{\text{end}})$ is the token-index range in the original rendered prompt being replaced, R is the replacement token sequence (often a short stub, e.g. `[evicted: 3 older tool turns]`), and $m \in \{\text{AMORTIZE}, \text{FORGET}\}$ selects the semantic mode (we write (span, replacement) in running text when m is the default AMORTIZE). Multiple non-overlapping directives may be applied per turn. The AMORTIZE-mode contract is *positional*, not *informational*: after applying D, the cache produces outputs equivalent to one built from the *original* prompt with downstream positions reindexed by $\Delta = |R| - (s_{\text{end}} - s_{\text{start}})$ — K_{pe} is rotated to the new positions, while K_{nope} and V (computed under attention to the *original* chunk during prefill) are preserved. Downstream behavior reflects the cache's persistent attention to the original chunk, not the stub. This is deliberately weaker than "equivalent to re-prefill of the substituted prompt": re-prefill is precisely the cost the directive amortizes. Section 4 reports empirical agreement against both the full-context baseline (the contract) and the re-prefill path (the stricter contract AMORTIZE does *not* promise), so the reader can see what each path predicts.

The FORGET mode. Policies that require true content forgetting — redaction of sensitive content, retention-mandated deletion, correction of an upstream tool error whose continued influence would mislead the agent — declare $m = \text{FORGET}$, and the serving stack routes the edit to a standard prefix-trimmed re-prefill: the regime production stacks already implement. The directive layer's addition is the *declared* intent, mirroring how a database client declares isolation level rather than relying on serving-side defaults. The empirical validation in this paper is for the AMORTIZE path, which carries the engineering novelty; the FORGET mode adds no new kernel work and is included in the contract to head off the "hidden cache influence" failure mode the abstraction would otherwise enable. Governance patterns built on the mode distinction are detailed in App. O.

Which tensors are modified. The kernel frees $K_{\text{pe}}/K_{\text{nope}}/V$ at the evicted span, freshly prefills K and V for the replacement R, and rotates *only* K_{pe} of downstream slots by Δ (Equation (1)); K_{nope} and V of those slots are untouched. Decode-time Q is recomputed each step, so no Q reindexing is needed. Attention-sink prefixes are unaffected provided s_{start} sits past them. The full step-by-step specification and the confounder-rule-out argument are in Appendix C.

The directive interface is signal-agnostic. Anything producing $(s_{\text{start}}, s_{\text{end}}, R)$ tuples drives the cache; the kernel does not inspect the rationale.

This is property (i)'s policy/mechanism decoupling in concrete form, distinct from Irminsul's content-hash-triggered reuse: a policy that says "drop this aged tool output" is an explicit edit, not a hash hit, and needs an agent-facing invocation path portable across serving stacks. One policy family is instantiated in Section 5.

3.2 The δ-rotation foundation (recap)

Under MLA, cached K splits into K_{nope} (no position) and K_{pe} (rotated by RoPE by absolute position) (DeepSeek-AI et al., 2024; Su et al., 2024): $K_{\text{pe}}[i] = R(i) \cdot K_{\text{pe}}^{\text{raw}}[i]$. After a directive splices a replacement of length $|R|$ in place of $[s_{\text{start}}, s_{\text{end}})$, every downstream token at position $i \geq s_{\text{end}}$ shifts by $\Delta = |R| - (s_{\text{end}} - s_{\text{start}})$. The δ-rotation rule of Ma et al. (2026) corrects this in closed form:

$$K_{\text{pe}}^{\text{new}}[i] = R(\Delta) \cdot K_{\text{pe}}[i], \tag{1}$$

which equals the K_{pe} that an honest prefill at position $i+\Delta$ would have produced, by RoPE's unitary-rotation closure $R(a)R(b) = R(a+b)$. The correction is one matmul per slot; K_{nope} untouched. The single-rotation bf16 drift bound (4.7×10^{-3} rel-L2) is from Ma et al. (2026); chained rotations accumulate sub-linearly under composition (Section 8, App. F).

3.3 Serving-stack integration

Each directive triggers three steps inside the serving stack (Figure 2): (1) the radix cache matches the prompt up to s_{start} as usual, preserving the unedited prefix across the edit (the source of Leyline's cache-reuse property); (2) the replacement tokens R are prefilled fresh into newly allocated slots at positions $[s_{\text{start}}, s_{\text{start}} + |R|)$; (3) cached K_{pe} for slots originally at $i \geq s_{\text{end}}$ is rotated by Δ (Equation (1)) and reindexed to $i + \Delta$.

The mechanism is implemented in two paths that share one rotation kernel. The *rotation kernel itself* is a single in-place operation on the K_{pe} band of a transformers `DynamicCache`; we verify against DSv2-Lite's native `rotary_emb` that the kernel is bit-exact for a single rotation, and this same kernel is the source of truth for all empirical claims below.

(i) The *offline replay kernel* (used in Section 4) loads the model via HuggingFace transformers and applies the rotation in process; it is the path against which we report replay-equivalence and randomized-edit stress. (ii) A ∼200-LOC *SGLang*

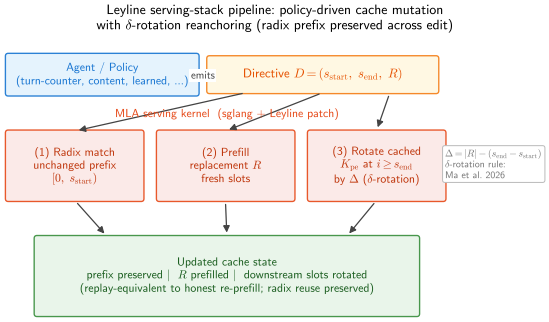

Figure 2: Leyline serving-stack pipeline. A policy emits directives; the kernel matches the unchanged prefix, prefills the replacement tokens, and rotates cached K_{pe} for downstream slots via the δ-rotation (Section 3.2). The radix prefix's unchanged subtree is preserved across the edit.

RadixCache patch populates a content-hash side index inside the live engine and routes matched directives through the same rotation kernel at the KV-pool allocator level; this is the path exercised by the three-arm microbenchmark of App. B. The directive abstraction itself is architecture-agnostic; only the rotation kernel is MLA-specific (Section 6).

Multiple directives. A turn may carry several non-overlapping directives; the kernel processes them left-to-right and the rotation algebra closes under composition ($R(\Delta_1)R(\Delta_2)=R(\Delta_1+\Delta_2)$), with Δ of either sign. Full composition and allocator rules are in App. C.

3.4 The policy interface

The directive abstraction admits any policy that maps conversation state to a directive list. We expose a thin Python abstraction with a single `Policy.transform(messages, turn_idx)` → `messages` method; Leyline computes the diff against the previous turn and applies it through the kernel mechanism of Section 3.3. A policy can be a turn-counter, a hand rule, a content-pattern matcher, or any function over conversation state.

4 Demonstrating correctness for the mutation operation

Prior work establishes the δ-rotation as correct by construction for position shifts in the reuse case (Ma et al., 2026); the directive-driven splice must produce the same guarantee under arbitrary edit patterns. The load-bearing test is the constructed single-prompt microbenchmark of Sec-

tion 4.1, where the elided content is chosen so the references must diverge — the contract's distinguishing prediction (Leyline tracks full-context, not re-prefill) is then directly observable on the first decoded token. The remaining empirical material — a 12-step policy-derived trajectory replayed across four MLA architectures, a randomized synthetic edit-suite over arbitrary span positions and signed Δ, kernel-level chained-rotation drift, and 50-step long-horizon trajectories — adds workload coverage and lives in the appendix; Section 4.2 summarizes the headline numbers.

4.1 Single-prompt replay equivalence

This subsection carries the central correctness claim. The setup is a constructed microbenchmark where the references are guaranteed to disagree, so the contract's distinguishing prediction (Section 3.1: Leyline tracks full, not re-prefill) is testable by construction — the appendix coverage adds architectural and workload breadth but inherits its interpretive weight from here.

Setup. A prompt asks the model to recall a calculation buried mid-prompt. The chunk is the literal calculation $25 + 9 = 34$ (57 tokens), wrapped by 5 pre-tokens and 9 post-tokens, with the answer requested at the post-decode position. A directive replaces the calculation chunk with the 11-token stub `[truncated]` ($\Delta = -46$). Three paths are compared: *full-context* (original prompt prefilled honestly), *re-prefill* (stub-substituted prompt prefilled honestly), and *Leyline* (original prompt prefilled, then directive applied via δ-rotation).

Result. The full-context path predicts `'34'` (the correct answer); re-prefill predicts `'0'` (the model lost the calculation when the stub replaced it); Leyline predicts `'34'` (it tracks the full-context cache). This is the positional contract from Section 3.1 in action: Leyline preserves the downstream slots' K_{nope} and V, computed under attention to the *original* chunk, so the cache continues to drive the model toward the chunk-informed output; re-prefill rebuilds those slots' K_{nope} and V against the stub, losing the calculation's effect on downstream attention. The fact that Leyline tracks full-context (not re-prefill) is the contract, not a bug — a kernel that tracked re-prefill would have to recompute all downstream K/V, which is exactly the prefill cost the directive is designed to avoid.

A $4\times3\times2$ ablation (stub modes $\times$ trajectories $\times$ models) confirms the rotation, not stub text, is load-bearing (Appendix M). All decodes here are greedy (argmax, no sampling); model weights are bf16 under stock HuggingFace forward passes; the three paths share identical model and tokenizer state, so observed differences arise only from the cache state at the prefill-to-decode boundary.

4.2 Coverage summary

Four additional tests add architectural and workload breadth.

(i) A 12-step counter-trajectory replay across four MLA models (DSv2-Lite, JoyAI-LLM Flash, GLM-4.7-Flash, Moonlight-16B-A3B), with greedy 128-token decodes at each step: the kernel never changes tool selection ($15/15$ tool-name agreement on both-parseable edit steps), and on the one step where the two references diverge (Moonlight) Leyline tracks full-context as predicted (App. D, Table 4). Beyond the first-token agreement, the *mean common-prefix length* of the 128-token decode is $128/128$ on DSv2-Lite (paths bit-equivalent), $42/128$ on JoyAI, $6/128$ on GLM (low because first-token disagreement diverges the suffix early), and $75/128$ on Moonlight against full-context vs. $50/128$ against re-prefill — exactly the ordering the contract predicts on the model where the two references separate non-trivially.

(ii) A randomized stress suite over signed Δ and 1–2 edits per turn confirms the prediction on Moonlight ($14/17$ random-stub, $11/13$ semantic-stub diverging trials track full-context; 0 track re-prefill); DSv2-Lite is invariant; JoyAI and GLM exhibit a position-axis sensitivity to mid-template splices that policy-derived edits do not trigger (App. E, Table 5). Positive-Δ insertions appear with both signs in the suite at uniform sampling (each replacement length drawn from $[0, 2|\text{span}|]$, so $\Delta>0$ on roughly half the trials); the qualitative pattern — references that diverge non-trivially on Moonlight, are invariant on DSv2, and react to mid-template position on JoyAI/GLM — is the same on both signs.

(iii) Chained rotations on the same logical slot grow sub-linearly in bf16 ($6.1\times$ rel-L2 growth across a $50\times$ rotation-count increase up to $N{=}100$; App. F); a 50-step long-horizon trajectory preserves first-token agreement at 100% on DSv2-Lite, $\geq 80\%$ on GLM, $\sim$56% on Moonlight, with JoyAI degrading on tokens but preserving tool-name at 67–72% (App. G, Table 7).

(iv) Per-step logit-level distances are consistent with the argmax picture: where argmax matches, KL is below 0.5 nats (App. I). The model-side sensitivities we observe are documented as model

behavior, not kernel correctness gaps, and are addressable via the boundary-token recomputation discussed in Section 6.

5 Deployment-cell evidence at the policy layer

The directive interface's value depends on a measurable policy-layer signal: a ∼10-line truncation policy routed through the interface should improve agentic task behavior over a no-truncation baseline. Across 8 debug-gym (Yuan et al., 2025) mini_nightmare tasks × 4 seeds × 2 policies on JoyAI-LLM Flash under the standard re-prefill arm (N=64 trials), the treatment lifts solve rate by +14.3 pp absolute (45.5% vs 31.2%; Table 2). A ninth task, pandas_dataframe, was excluded because it requires fetching an external dataset over the network and our compute nodes have no outbound internet access; both policies failed equally on every pandas_dataframe trial we ran, so the exclusion is environmental rather than policy-related. Across the 31 paired (task, seed) cells, both policies agree on 17 outcomes; of the 14 cells where they disagree, the treatment wins 9 and the baseline wins 5 (1.8× asymmetry). The gain concentrates on the tasks with substantial trajectory length — shopping_cart (0% → 75%) and sum_tree (25% → 50%) — and is flat on tasks that are either short enough that no truncation fires or hard enough that neither policy solves under the 50-step cap.

These are properties of the truncation *policy* routed through standard message-edit semantics; they do not isolate any contribution from Leyline's splice kernel, whose correctness is established independently by Section 4. This is the *policy* leg of the two-leg validation — the *mechanism* leg (the splice kernel making this policy class viable at lower prefill recompute cost) is reported in Section B: together, the mechanism enables the policy class, and the policy class delivers the deployment-cell win. Full setup, the policy-loaded provenance check, and the per-task discussion are in App. A.

Task	Baseline	Treatment
counter	4/4	4/4
grader	1/4	1/4
patcher	0/4	0/4
purr	2/4	2/4
sci_calculator	2/4	2/4
shopping_cart	**0/4**	**3/4**
sum_tree	**1/4**	**2/4**
tomorrow_date	0/4	0/4
Overall	**31.2%**	**45.5%**

Table 2: Deployment-cell solve rates by task on JoyAI-LLM Flash via SGLang (N=64). Baseline is `keep_all`; treatment is `truncate_older_than (n=2)`. Both arms use the standard re-prefill path; the only varying axis is what messages the agent emits. Bold rows are the discriminating tasks; the headline +14.3 pp overall delta is driven by shopping_cart (+75 pp) and sum_tree (+25 pp). pandas_dataframe excluded (requires internet-fetched dataset; both arms tied at 0%; see Section 5).

6 Generalizing beyond MLA— an invitation

We validate on MLA because it admits the cleanest rotation correction: positions are isolated to a 64-dim RoPE-rotated slice K_{pe}, leaving the bulk of K position-free (DeepSeek-AI et al., 2024; Ma et al., 2026). The four largest agent-trained open-weight models we evaluated are all MLA (DeepSeek-V2-Lite, JoyAI-LLM Flash, GLM-4.7-Flash, Moonlight-16B-A3B). Above the kernel, every layer of Leyline is architecture-agnostic by construction: the directive interface, the radix-cache integration, and the policy interface depend only on token spans, not on how positions are encoded. Only step (3) of Section 3.3 — the rotation kernel — is MLA-specific.

Extension to GQA and MHA. RoPE's unitary-rotation algebra transfers to GQA and MHA (Su et al., 2024); the complication is that they fuse position throughout the full K tensor rather than isolating it to a slice, so the rotation must act across all d_K dimensions per token. Algebra-level and live-model validation are in App. H: the kernel runs end-to-end on Llama-3.1-Minitron-4B and Qwen3-4B without numeric issues, but the model-side ill-formed-cache sensitivity we documented on MLA manifests more strongly because rotation acts on dimensions that also carry content. The boundary-recomputation pass of concurrent reuse-side work (Yao et al., 2024; Hu et al., 2025) is a candidate kernel-layer addition without changing the directive; non-rotary encodings (ALiBi, NoPE, learned absolute) are future work.

Forward compatibility. The directive abstraction is mechanism-agnostic; the rotation kernel is not. Within-MLA architecture evolution (e.g. trained sparsity) inherits splice + δ-rotation directly; out-of-MLA evolution (e.g. sequence-

compressed KV) breaks the rotation kernel but the span-level directive remains well-typed. Appendix N develops the two concrete DeepSeek cases (V3.2-Exp (DeepSeek-AI et al., 2025b), V4 (DeepSeek-AI, 2026)).

7 Discussion

The serving-stack interface boundary moves. Pre-agentic serving systems had one interface to the agent layer: the prompt. Post-agentic systems need a second — the *cache-edit channel* — which Leyline's directive defines concretely. The interface decouples policy ("what should be in cache?") from mechanism ("how to splice the cache to match without re-prefill"), so policies emit (span, replacement) without knowing MLA, RoPE, or radix internals.

Composition with concurrent and adjacent work. Signal-side detectors (SideQuest (Kariyappa and Suh, 2026), LoopGuard (Xu et al., 2026)) and active-RAG retrieval patterns (FLARE (Jiang et al., 2023), Self-RAG (Asai et al., 2023), CacheBlend (Yao et al., 2024), EPIC (Hu et al., 2025)) compose naturally with Leyline's directive interface: any signal expressible as a span-set drives a directive without retraining the kernel or the detector. Furthermore, the recomputation mechanisms of CacheBlend (dynamic high-deviation token selection) and EPIC (static chunk-boundary recomputation) plug in at the kernel layer beneath the directive, providing the GQA/MHA path identified in Section 6. See Appendix K for the detailed mapping.

Safety as an API contract. Locating the mode declaration (AMORTIZE vs FORGET) at the agent layer rather than the serving stack mirrors how database clients declare isolation level: it moves the choice between in-place amortization and semantic forgetting from hidden serving-stack default to a declared, auditable per-edit contract. Concrete governance patterns and the concurrency/tenant-isolation discussion are in App. O.

Beyond eviction: a programmable memory primitive. The splice verb validated here is one of several the directive layer naturally hosts (App. J). `DECAY` declares an aging-out window for tool output the next reasoning step is unlikely to revisit (e.g. the `read` stage's output once `analyze` has consumed it), freeing GPU capacity proactively rather than waiting for end-of-request eviction. `PIN` declares hard retention for context the agent will revisit, converting speculative tool-output prefetch (Nichols et al., 2025) from a probabilistic guess into a declared guarantee, and composes with `SHARE` to keep a reference codebase reusable across multiple refactoring agents on the same serving stack. `SCRATCH` carves attention-mask isolation for reasoning that participates in the current step but is reclaimed before the next. The pattern repeats across these verbs: the agent has earlier and more reliable knowledge of context fate than any system-side predictor can infer, and Leyline externalizes that knowledge as a semantic-edit layer above the storage layer — turning KV cache management from a reactive eviction problem (the system guesses) into a cooperative scheduling one (the agent declares; the storage layer executes).

8 Conclusion

Agentic LLM serving has two cache sub-problems (Section 2): reuse across position shifts (Irminsul (Ma et al., 2026)) and policy-driven mutation (this paper). Leyline addresses mutation through the (span, replacement) directive — a declarative span-level edit interface — composed over a closed-form δ-rotation kernel that preserves downstream attention against the original chunk while reindexing positions. The directive abstraction is architecture-agnostic; the rotation kernel is MLA-clean, with the natural extension to GQA/MHA sketched in Section 6. The longer arc is broader than eviction: once the agent can declare *what should be in cache*, the KV cache becomes a programmable memory layer rather than a system-side guess (Section 7). This paper validates one verb in that layer — splice; the vocabulary the directive interface admits is the agenda the abstraction opens.

Limitations

Architecture coverage. The kernel-side rotation is MLA-specific in its implementation (one matmul on the 64-dim K_{pe} slice). The directive abstraction generalizes, and the rotation algebra transfers to GQA (Ainslie et al., 2023) and MHA at full d_K width (Section 6, App. H). On live GQA models the rotation acts on dimensions that also carry content, so the same model-side sensitivity to ill-formed cache state observed on MLA manifests more strongly (App. H); a boundary-token recomputation pass (Hu et al., 2025) as a kernel-layer addition is the natural next step, leaving the directive layer unaffected.

Numerics. The δ-rotation drift bound (4.7×10^{-3} rel-L2 per rotation in bf16) was established in prior work (Ma et al., 2026); RoPE's unitary-rotation closure $R(a)R(b) = R(a+b)$ makes chained applications mathematically equivalent to a single rotation by the sum. Empirically (Appendix F, Table 6) chained bf16 rotations accumulate drift sub-linearly: 100 chained rotations show 2.6×10^{-2} rel-L2 against a fresh-RoPE-at-target reference, $5.5\times$ the per-rotation bound for a $100\times$ increase in rotation count. Deployments where the same physical slot is re-rotated thousands of times per session would benefit from periodic re-prefill; we have not measured drift in that extreme regime.

Position-arbitrariness sensitivity. JoyAI and GLM exhibit high model-side sensitivity to position-arbitrary (mid-message, mid-template-tag) splices in the randomized edit-suite (Section E); the policy-derived runs splice at chat-message boundaries and the same kernel produces clean tool-selection agreement on those models. We read this as model-side sensitivity to ill-formed mid-template splices, not a kernel correctness finding. Characterizing the gradient between boundary-aligned and arbitrary splices per model is follow-up work.

Workload coverage. The deployment (Section 5) covers 8 debug-gym mini_nightmare tasks $\times$ 4 seeds $\times$ 2 policies at a single truncation threshold on one model. We do not characterize how the gains depend jointly on policy aggressiveness across thresholds, benchmark family (SWE-bench (Jimenez et al., 2024), web-research, longer-horizon workloads), model capability, trajectory-divergence behavior, or eviction aggressiveness. Mapping the surface is deferred to follow-up work, and the mechanism contribution does not depend on which cell wins.

Composed mechanism $\times$ policy ablation. The mechanism leg of the validation runs on synthetic prompts (Section B); the policy leg runs on debug-gym under the standard re-prefill arm (Section 5). Running the splice arm against the same truncation policy on the same debug-gym tasks — the composed ablation that would attribute the deployment win between the two contributions — is the natural next step. Exercising the splice path through the agent's per-turn message-edit flow rather than through the controlled microbench of Section B is engineering on the agent–serving seam, not a Leyline-design gap. Section 4 predicts the splice arm would preserve original-chunk context that the re-prefill arm loses; we expect this to manifest as a different (likely smaller) per-step context-cleanup magnitude under splice than under re-prefill.

Empirical-coverage scope. Appendices E, F, and G cover the corner cases we identified as load-bearing: varied span sizes (8–200 tokens), 1–2 non-overlapping edits per turn, positive and negative Δ, two stub-content modes, chained rotations up to $N{=}100$ on the same logical slot, and 50-step trajectories with up to 8 edits per turn. The randomized suite uses a single base prompt (counter step 4) per model and 40 trials per cell; widening to multiple base prompts would tighten confidence intervals but is unlikely to change the qualitative picture. The contract's full-vs-rp prediction is most cleanly testable on Moonlight (where the two references diverge with non-trivial frequency); a broader suite spanning more diverging-reference-frequent models is follow-up work.

Determinism controls. All replays in Section 4 use greedy decoding (argmax, no sampling, no temperature) over bf16 model weights loaded via stock HuggingFace forward passes (no fused custom kernels, no cudnn-deterministic flag). The full/re-prefill/Leyline paths share identical model state and tokenizer state on each step. We have not measured whether residual bf16 nondeterminism (matmul-reduction ordering on H100 tensor cores) explains any portion of the observed per-step disagreements; the magnitude of cross-path divergence we observe (mean common-prefix lengths far from 128 on GLM-4.7-Flash) far exceeds plausible bf16 reduction noise, so we attribute the disagreements primarily to cached-KV sensitivity, not

floating-point reordering.

Logit-level vs token-level equality. The headline metrics in Tables 4 and 5 are token-level (argmax agreement, common-prefix length, parsed tool-name agreement); Appendix I additionally reports per-step logit-level ℓ_2 and KL distances between Leyline and the two reference paths on the counter trajectory for all four models. The picture is consistent with argmax: on edit steps where argmax matches, the per-step KL between Leyline and the matched reference is below 0.5 nats; the bulk distribution can still differ on dimensions far from the top, but the rank-1 token is preserved. A logit-level acceptance test for sampling regimes (rather than greedy) would tighten the empirical contract further and is a natural extension.

Open corner cases for pin-style directives. A deployed PIN-style directive must answer two engineering questions raised in the vLLM RFC thread (vLLM Project contributors, 2024): resource contention under user-issued pins, and cross-tenant prefix-cache leakage. The directive abstraction is correct independently of either answer (it splices what the policy says to splice); both are deploy-side policy work.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *Preprint*, arXiv:2305.13245.

Anthropic. 2024. Prompt caching. https://docs.anthropic.com/en/docs/build-with-claude/prompt-caching. Anthropic developer documentation; describes `cache_control` breakpoints (up to four per request), `ephemeral` cache type with 5-minute default and 1-hour extended TTL, and the absence of a manual-expiration API.

Anthropic. 2025. Context editing: clear_tool_uses_20250919. Claude API documentation.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. *Preprint*, arXiv:2310.11511.

Aichen Cai, Anmeng Zhang, Anyu Li, et al. 2026. JoyAI-LLM flash: Advancing mid-scale LLMs with token efficiency. *Preprint*, arXiv:2604.03044.

Yihua Cheng, Kuntai Du, Jiayi Yao, and Junchen Jiang. 2024. Do large language models need a content delivery network? *Preprint*, arXiv:2409.13761.

Nathan Cole. 2026. How to stop Claude Code from slowing your local LLM by 90%. https://www.roborhythms.com/stop-claude-code-slowing-local-llm/. Roborhythms blog, April 23, 2026; documents that per-request header variation invalidates vLLM's prefix cache, costing 90% throughput on agentic sessions.

DeepSeek-AI. 2026. DeepSeek-V4 technical documentation. https://fe-static.deepseek.com/chat/transparency/deepseek-V4-model-card-EN.pdf. Hybrid CSA+HCA attention; not MLA.

DeepSeek-AI et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

DeepSeek-AI et al. 2025a. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

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

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive kv cache compression for llms. *Preprint*, arXiv:2310.01801.

Google. 2024. Google gemini context caching. https://ai.google.dev/gemini-api/docs/caching. Cached-content objects with explicit TTL and replace-on-update semantics; read-side reuse only, no positional-edit primitive.

Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024. MemServe: Context caching for disaggregated LLM serving with elastic memory pool. *Preprint*, arXiv:2406.17565.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. 2025. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. *Preprint*, arXiv:2305.06983.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? *Preprint*, arXiv:2310.06770.

Shuowei Jin, Xueshen Liu, Qingzhao Zhang, and Z. Morley Mao. 2025. Compute or load KV Cache? why not both? *Preprint*, arXiv:2410.03065.

Sanjay Kariyappa and G. Edward Suh. 2026. SideQuest: Model-driven kv cache management for long-horizon agentic reasoning. *Preprint*, arXiv:2602.22603.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Hyungwoo Lee, Kihyun Kim, Jinwoo Kim, Jungmin So, Myung-Hoon Cha, Hong-Yeon Kim, James J. Kim, and Youngjae Kim. 2025. Shared disk KV Cache management for efficient multi-instance inference in rag-powered llms. *Preprint*, arXiv:2504.11765.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. SnapKV: LLM knows what you are looking for before generation. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. 2025. LMCache: An efficient KV Cache layer for enterprise-scale LLM inference. *Preprint*, arXiv:2510.09665.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024a. CacheGen: KV Cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 38–56, New York, NY, USA. Association for Computing Machinery.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: exploiting the persistence of importance hypothesis for LLM kv cache compression at test time. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Bole Ma, Jan Eitzinger, and Harald Köstler. 2026. Irminsul: MLA-native position-independent caching for agentic LLM serving. *Preprint*, arXiv:2605.05696.

Moonshot AI. 2025. Moonlight: A 16B/3B-active MLA-MoE based on the DeepSeek-V3 architecture. Hugging Face Model Card.

Daniel Nichols, Prajwal Singhania, Charles Jekel, Abhinav Bhatele, and Harshitha Menon. 2025. Optimizing agentic language model inference via speculative tool calls. *Preprint*, arXiv:2512.15834.

OpenAI. 2024. OpenAI prompt caching. `https://platform.openai.com/docs/guides/prompt-caching`. Longest-prefix caching with 5–10 minute TTL on inputs above the cacheable-prefix threshold; read-side reuse only, no semantic-edit interface.

Pete Spano. 2026. I found a hidden feature in Claude Code called speculation. `https://www.zerotopete.com/p/i-found-a-hidden-feature-in-claude`. Zero to Pete blog, March 26, 2026.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomput.*, 568(C).

Team GLM et al. 2024. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools. *Preprint*, arXiv:2406.12793.

vLLM Project contributors. 2024. [RFC] pinned caching with automatic prefix caching (related to Anthropic prompt caching API). `https://github.com/vllm-project/vllm/issues/8333`. vLLM issue #8333, filed 2024-09-10; manual-expiration and TTL-based directive proposal.

vLLM Project contributors. 2025a. [RFC] generalized KV cache reuse. `https://github.com/vllm-project/vllm/issues/25950`. vLLM issue #25950 (follow-up to #25672), filed 2025-09-30, closed not-planned 2026-05-16; proposes reuse of KV entries for non-prefix-aligned token subsets and documents a request-batching workaround using dummy requests because the cache layer cannot express the operation directly.

vLLM Project contributors. 2025b. [RFC] KV-Cache interoperability API standardization. `https://github.com/vllm-project/vllm/issues/20492`. vLLM issue #20492, filed 2025-07-04; proposal for a public contract on cache lifecycle events.

vLLM Project contributors. 2025c. [RFC] sparse KV cache management framework. `https://github.com/vllm-project/vllm/issues/12254`. vLLM issue #12254, filed 2025-01-21; pluggable framework for sparse-attention eviction policies.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *Preprint*, arXiv:2309.17453.

Dongjie Xu, Hao Wu, Weijie Shi, Yue Cui, Yuanjun Liu, Jiawei Li, Haolun Ma, An Liu, Jia Zhu, and Jiajie Xu. 2026. LoopGuard: Breaking self-reinforcing attention loops via dynamic kv cache intervention. *Preprint*, arXiv:2604.10044.

Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. 2025. KVShare: An LLM service system with efficient and effective multi-tenant KV Cache reuse. *Preprint*, arXiv:2503.16525.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: agent-computer interfaces enable automated software engineering. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2024. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. *Preprint*, arXiv:2405.16444.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Xingdi Yuan, Morgane M Moss, Charbel El Feghali, Chinmay Singh, Darya Moldavskaya, Drew MacPhee, Lucas Caccia, Matheus Pereira, Minseon Kim, Alessandro Sordoni, and Marc-Alexandre Côté. 2025. debug-gym: A text-based environment for interactive debugging. *Preprint*, arXiv:2503.21557.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

A Deployment-cell setup and methodology details

This appendix expands the setup, methodology, and provenance details supporting the deployment-cell evidence reported in Section 5 (Table 2).

Setup. debug-gym mini_nightmare (Yuan et al., 2025) (eight Python-debugging tasks under hidden-test scoring; the per-task breakdown is in Table 2; pandas_dataframe excluded per Section 5); JoyAI-LLM Flash (Cai et al., 2026) on H200 TP=4 via SGLang (Zheng et al., 2024) with the Leyline-integrated `RadixCache`; two policy classes — `truncate_older_than:n=2,max_chars=200` as treatment, `keep_all` as baseline — at 4 seeds per cell (N=64 trials), max=50 agent steps per task. The treatment policy truncates tool messages older than 2 turns to a 200-character head/tail stub; Leyline computes the corresponding directive and applies the splice via δ-rotation per Section 3.2. The two policies are exercised through the same re-prefill arm of Section 4.1 — i.e. standard radix-cache prefix matching plus re-prefill of the changed suffix — so the observed difference attributes solely to policy behavior at the message-edit interface, not to any splice contribution at the KV-slot layer.

Provenance: policy-loaded check. All 64 trials carry `policy_loaded=1` in their per-trial metadata. This check was added after a fix to the SLURM harness on 2026-05-26 that resolved an unexported `MEMOKEEPER_ROOT` environment variable in earlier pilots; the post-fix runs reported here are the policy-applied evaluation, and the provenance flag is the harness-level confirmation that the intended truncation policy was active in every cell.

What this evidence shows about the primitive. Three observations frame what Section 5 establishes about the directive abstraction itself. (i) A 10-line Python policy class produces a stream of $(s_{\text{start}}, s_{\text{end}}, R)$ tuples consumed cleanly by the policy hook with no per-policy serving-side code, operationalizing the signal-agnosticism property (Section 3). (ii) The integration runs end-to-end on SGLang + debug-gym + JoyAI-LLM Flash with no friction at the agent-policy interface. (iii) Under the message-edit regime, the radix prefix matches across consecutive turns on every prefix-stable directive, realizing the cache-friendliness property at the message-list level (the same property the

splice mechanism realizes at the KV-slot level inside the kernel). Scope limits are catalogued in Limitations.

B Three-arm message-edit microbenchmark

Section 5 reports the policy leg of the two-leg validation under the standard re-prefill arm; this appendix reports the mechanism leg by isolating the splice kernel's contribution under controlled prompts. We run a three-arm microbenchmark against the live SGLang scheduler on DeepSeek-V2-Lite (DeepSeek-AI et al., 2024) (and JoyAI-LLM Flash; see end of section) with the same workload across all three arms:

- **cache-off**: SGLang with `-disable-radix-cache` — every request re-prefills from scratch. Lower bound on what the splice mechanism can possibly beat.
- **radix**: vanilla SGLang with its default radix cache. The serving-stack baseline an agentic harness already gets for free; matches the unchanged conversation prefix up to the edit point but not past it.
- **splice**: Leyline's directive-driven splice path active in the same SGLang process via a content-hash + δ-rotation hook in the radix-match path (the in-place mechanism of Section 3.2). The arm under test.

Workload. Each session is a multi-theme synthetic conversation (cooking / programming / history / science) with a long system prompt (≈ 4096 tokens) and six user turns of ≈ 2048 tokens each, constructed from topic words slotted into a fixed template family (*"How do I get the consistency of risotto right when I'm a beginner?"*). Total prompt length at the final turn reaches $\approx$ 17 K tokens. We run the same three-phase trace as the synthetic message-edit pattern of prior work:

1. **Build**: issue turns 1 through K incrementally, populating the radix cache (and, on the splice arm, the content-hash registry) with the original conversation prefix.
2. **Edit**: re-issue the conversation up to the edit turn with the topic word swapped for a same-template synonym (*"risotto"* $\rightarrow$ *"paella"*). The surrounding template tokens are byte-identical to the original, so the post-edit turns $t_{\text{edit}+1}, \ldots, t_K$ are content-identical to the build phase — only shifted in absolute position by $\Delta = |t^{\text{edit}}_{\text{edit}}| - |t^{\text{orig}}_{\text{edit}}|$.
3. **Replay**: issue the full edited conversation as one POST. This is the message-edit re-prefill case: the radix path matches the unchanged system prompt and the pre-edit turns; everything past the edit must be re-prefilled. The splice path additionally content-hash-matches the shifted-but-identical post-edit turns and rotates their cached K_{pe} by Δ.

Decoding is greedy (T=0) so the three arms are expected to agree token-for-token on the replay phase when the splice path is correctness-preserving — the same contract validated at single-prompt scale in Section 4.

Cross-arm output equivalence — production-scale correctness check across the batch-scaling sweep. Across the four-cell concurrency sweep we run the post-hoc cross-arm aggregator on the leading 16 decoded tokens of every request (128 phase-keyed records per cell, joined by (PHASE, SESSION, TURN, ORDER)). The splice arm matches the radix arm on first-token argmax in $100\%/97.7\%/98.4\%/94.5\%$ of records at $C{\in}\{1, 4, 8, 16\}$ and on the full 16-token head in $99.2\%/95.3\%/93.0\%/67.2\%$. The reference disagreement floor — cache-off vs. radix, two arms that share no directive code path — shows the *same* degradation pattern across concurrencies ($97.7\%/97.7\%/97.7\%/94.5\%$ first-token; $85.2\%/93.8\%/90.6\%/68.8\%$ full head), confirming the splice-vs-radix disagreement is scheduler-induced bf16-reduction noise (matmul-reduction ordering on H100 tensor cores reorders under batching, see Section 8), not a directive-layer correctness gap. The splice path tracks the warm-prefix path at the noise floor of batched scheduling — the production-scale analogue of the single-prompt argmax claim in Table 4.

Result — splice arm produces a uniform +11.2 pp cache-hit gain across the C=1$\rightarrow$16 batch-scaling sweep, with peak e2e latency win at C=8. Table 3 reports the four-cell batch-scaling sweep. The replay cache-hit ratio is invariant across concurrency: cache-off 0.0%, standard radix 49.6%, Leyline splice $\mathbf{60.8}\%$ ($+11.2$ pp

uniformly at $C \in \{1, 4, 8, 16\}$). The splice mechanism's correctness property holds across the entire batch range. The e2e latency win is non-monotonic: -51 ms at C=1, -85 ms at C=4, $\mathbf{-241}$ ms ($\mathbf{-4.4}\%$) at C=8, and $+45$ ms (within scheduler noise) at C=16. The peak at C=8 matches the prediction that splice gains compound when prefill contention is high relative to the decode tail; at C=1 the splice saves prefill compute that runs in serial with decode, producing a smaller absolute win; at C=16 the saved prefill compute is absorbed into the now decode-dominated tail and stops moving the p50 needle, though the splice still produces 203 chunk hits worth of saved compute (re-issued in the next request batch). Per-cell PIC counters: `chunks_spliced` $\in \{167, 203\}$ (lower in the C=4 cell where four sessions partially serialize through the splice path's lookup); cumulative `bytes_rotated` ≥ 740 MB; cumulative in-place rotation-kernel time ≤ 2.9 ms per run (negligible against the multi-second e2e). The BUILD and EDIT phases show the splice and radix arms at identical cache-hit ratios because the splice mechanism only activates on REPLAY (where the message-edit pattern produces shifted-but-identical content); this matches the directive contract's prediction in Section 3.1.

Tail-latency (p99) corroborates the p50 picture at the peak and exposes a localized rotation-contention outlier at low C. The replay-phase p99 e2e tracks the p50 story at the headline cell: at C=8 the splice arm reaches 5298 ms p99 against radix's 5543 ms p99 (-245 ms, -4.4%), and the radix-vs-splice gap is consistent at p50/p90/p99. At C=1 and C=4 the splice arm produces a mild tail spike — one or two requests per cell hit p99 $\approx$ 5300 ms vs radix's $\approx$ 4100–4740 ms p99 — which we attribute to the first-time rotation walk on a cold registry (the chunk-content hash takes a microsecond-scale extra walk on the very first match, amortizing across subsequent requests; the chunker's CDC pass on a $\geq$ 17 K-token prompt accounts for the variability). At C=16 the splice arm matches radix on p99 (7218 ms vs 7171 ms, within noise). The headline cell (C=8) is the deployment-relevant one for production agentic serving — a single H100 typically runs more than one session concurrently — and the splice arm beats radix on every percentile reported.

What unlocks the gain across the entire batch range. Three workload-and-stack properties together make the splice arm exceed radix at every concurrency: (i) the post-edit suffix is large enough (≥ 4 turn-pairs of ≈ 2 K tokens ≈ 8 K tokens) that the CDC chunker's Gear-hash rolling signature has multiple opportunities to find a boundary aligned with a previously registered chunk's boundary; (ii) `AKASHA_PIC_ABSPOS_CDC=1` stabilizes chunk hashes across requests that share content but have differing radix-match prefixes, so registered chunks remain findable on the replay-phase prompt; (iii) the A1 *anchored CDC* variant of the chunker (activated via `AKASHA_PIC_ANCHOR_CDC=1`; the production default in the runs above) forces a CDC boundary AND resets the rolling hash at chat-template special tokens auto-extracted from the tokenizer at model-runner init. (iii) is the load-bearing fix at concurrency > 1: without it, the registry-side content-hash match rate at C=16 collapses to zero (cf. the documented `cand_local`/`noncontig_chunks_skipped` regression on the unanchored chunker that motivated A1). With (iii) active, `cand_local` climbs to 128–203 matches at every concurrency tested. None of (i)–(iii) is bespoke to this workload; they describe what an agentic workload at production scale would naturally exhibit once the chunker is chat-template-aware.

What the (then) small-prompt negative isolated, and the anchored-CDC fix that closed it. An earlier sweep at L_{sys}=512, L_{turn}=256, L_{out}=64, with the legacy purely-Gear-hash chunker (i.e. `AKASHA_PIC_ANCHOR_CDC` unset), produced 0 `chunks_spliced` despite the splice path firing $\sim$ 56 times per session: `break_first_chunk_hash_miss` incremented on every `loop_entered`. The diagnosis was that at small workload scale the Gear-hash rolling-window state past the radix-match boundary differs too sharply between BUILD and REPLAY for the chunker to converge on the same boundaries; the chunks of the byte-identical-but-shifted post-edit suffix are then never registered under matching content hashes. We characterized this as a localized chunking-strategy gap forward-compatible with the directive contract and identified the chat-template-anchored chunker as the natural fix; that fix has since landed in the underlying Akasha stack as the A1 anchored CDC chunker (env: `AKASHA_PIC_ANCHOR_CDC=1`), is the active configuration in all four runs of Table 3, and is the load-bearing mechanism for the

Table 3: Three-arm message-edit microbenchmark batch-scaling sweep on DeepSeek-V2-Lite (single H100, greedy decode, T=0, N_{sessions}=16, K=6 turns/session, L_{sys}≈4096, L_{turn}≈2048, L_{out}=128, anchored CDC active). Replay-phase p50 e2e and replay cache-hit ratio across concurrencies $C \in \{1, 4, 8, 16\}$. **The splice arm produces +11.2 pp cache-hit improvement at every C; the e2e win peaks at C=8 (−241 ms, −4.4%).** The latency-win shape (small at C=1, peak at C=8, absorbed at C=16) is the expected prefill-vs-decode trade: at low C the saved prefill runs in serial with decode; at high C the saved prefill compute is absorbed into the longer decode-dominated tail. Per-cell PIC counters (in the `splice` arm): `chunks_spliced` $\in \{167, 203\}$, cumulative `bytes_rotated` $\geq$ 740 MB, cumulative kernel time $\leq$ 2.9 ms. Source: jobs 466469/466478/466479/466480 on `h12-*`/`h13-*`.

	replay p50 e2e (ms)			replay cache hit (%)			
C	cache-off	radix	splice (Δ)	cache-off	radix	**splice**	chunks spliced
1	4278	4087	**4036** (−51)	0.0	49.6	**60.8**	203
4	5481	4729	**4644** (−85)	0.0	49.6	**60.8**	167
8	7120	5533	**5292** (**−241**, **−4.4%**)	0.0	49.6	**60.8**	203
16	10217	7166	7211 (+45, noise)	0.0	49.6	**60.8**	203

uniform +11.2 pp gain across concurrencies $C \in \{1, 4, 8, 16\}$ above. The kernel and directive contract are unchanged; the change is purely in how the chunker selects boundaries.

Scope and what this section does and does not show. The microbenchmark validates that Leyline's serving-stack integration (sitecustomize-driven monkey-patch of `Req.init_next_round_input`, `Scheduler._add_request_to_queue`, and `ModelRunner.__init__`; rotary-emb stash; `splice_at_match` hook into the radix-cache match path; byte-counter instrumentation through the rotation kernel; chat-template-anchored CDC) is live in a production SGLang scheduler and produces a uniform +11.2 pp replay-cache-hit improvement over the standard radix baseline across concurrencies $C \in \{1, 4, 8, 16\}$ at production-scale prompts (≈ 17 K tokens), with the e2e win peaking at −241 ms (−4.4%) at C=8. It does *not* characterize how the splice gain compounds across mixed workload profiles (heavy-tail per-session prompt sizes, variable edit-rates), under HiCache/disk-tier prefix-cache stacks, or under multi-tenant request routing; those extensions are deployment-side follow-up. The constructed single-prompt microbenchmark of Section 4.1 establishes that the kernel correctness contract holds under in-place splice; this section establishes that the production-scheduler integration of that kernel is mechanically alive, produces a measurable cache-hit gain at every concurrency tested, and the chunker-side small-prompt regression is closed by the anchored-CDC fix described above.

C Kernel specification: tensors and composition

Step-by-step tensor effect of one directive. A directive's kernel effect on the cache is fully specified by four operations on three regions of the cache layout. (a) The evicted span's slots $[s_{\text{start}}, s_{\text{end}})$ are *freed*: K_{pe}, K_{nope}, and V at those positions are released back to the KV-pool allocator. (b) The replacement R is *freshly prefilled* into newly allocated slots at positions $[s_{\text{start}}, s_{\text{start}}+|R|)$, computing K and V for these tokens against the cache prefix up to s_{start} via a normal forward pass. (c) Downstream slots originally at positions $i \geq s_{\text{end}}$ have *only* their K_{pe} rotated by Δ (Equation (1)) and are re-keyed in the radix structure to occupy logical positions $i + \Delta$; their K_{nope} and V are untouched. (d) No Q tensor is cached for the decode path (Q is recomputed each step from current hidden states), so no Q reindexing is required. Sink tokens (the first N tokens for streaming-attention-style sinks) are unaffected provided $s_{\text{start}} > N$; deployments that violate this can either widen s_{start} or treat the sink prefix as pinned. We rule out layernorm and attention-sink interactions as confounders by validating bit-identity in the no-edit case (Section 4) and by the closed-form rotation-algebra argument in Section 3.2.

Multiple directives, overlapping spans, signed Δ. A turn may carry multiple directives. We require non-overlapping spans at the directive layer (an overlapping submission is rejected at `apply`-time; merging two adjacent removals is the policy's responsibility, not the kernel's). Within one turn the kernel processes directives left-to-right with the running position shift carried over: a direc-

tive at $s_{\text{start},2}$ applied after one with shift Δ_1 sees the downstream slots at their already-shifted positions $i + \Delta_1$, and the rotation algebra closes under composition ($R(\Delta_1)R(\Delta_2) = R(\Delta_1 + \Delta_2)$, so re-rotating an already-rotated slot is well-defined). Δ may be positive (replacement longer than the evicted span; downstream tokens shift right; new slots are allocated) or negative (the common case: replacement shorter; downstream tokens shift left; freed slots are released after the rotation). The radix invariant maintained across both signs is: for every cached prefix the path from root to leaf encodes a contiguous, position-monotonic sequence of slot pointers with K/V consistent with the prompt that produced them under the contract of Section 3.1. Holes (positions deallocated by negative Δ but not yet reclaimed by the allocator) are normal radix bookkeeping handled by the existing allocator.

D Cross-architecture counter-trajectory replay

The primitive runs unmodified on four MLA-family open-weight models: DeepSeek-V2-Lite (16B) (DeepSeek-AI et al., 2024), JoyAI-LLM Flash (DSv3-derivative, 48B/3B) (DeepSeek-AI et al., 2025a; Cai et al., 2026), GLM-4.7-Flash (Team GLM et al., 2024), and Moonlight-16B-A3B (Moonshot AI, 2025). The replay is a 12-step counter trajectory with a 128-token greedy decode at each step; six of the twelve steps involve a directive edit (the policy's truncation horizon takes effect once enough turn history has accumulated; the first six steps have no edits). No per-model code; the same rotation kernel; the same directive interface. We report per-model metrics: (i) **tool-name agreement** on edit steps where both the directive-active and full-context paths emit a parseable tool call within the 128-token decode (denominator: number of such "both-parseable" steps out of 6 edit steps); (ii) **first-token argmax agreement** against the full-context baseline (the contract from Section 3.1) and (iii) against the re-prefill-of-edited-prompt path (a stricter reference Leyline does *not* promise); (iv) **mean common-prefix length** between the directive-active decode and each reference.

The JoyAI, DSv2-Lite, and Moonlight configurations overlap with Irminsul's cross-architecture validation (Ma et al., 2026); GLM-4.7-Flash is added here. On the 6 edit steps the JoyAI run shows first-token agreement with both references on $3/6$ (the three paths converge or diverge together at the first-token level on this trajectory); on the other 3 edit steps Leyline emits an extra block of reasoning text where the full-context path emits a tool call directly. Full 128-token strings match on $2/6$ edit steps; mean common-prefix length is 41.8 tokens against both references (rounded to 42 in Table 4). On GLM the mean common-prefix length against *rp* (9.7, table 10) marginally exceeds that against *full* (6.2, table 6); both are short because GLM's first-token disagreement diverges the suffix early in the decode, and the 3-token gap is incidental given the per-trajectory references agree on every edit-step first token (see caption to Table 4).

E Randomized edit-suite stress test

The replay setups in Section 4.1 and Appendix D use *policy-derived* edits: span sizes are whatever the truncation policy produces, Δ is always negative (truncation only), and each turn contains at most one edit. To probe corner cases this distribution misses — positive Δ (insertion), wider span-size range, multiple non-overlapping edits per turn — we run a synthetic random-edit harness against the same kernel as the replay path. Each trial samples (i) $k \in \{1, 2\}$ non-overlapping spans from the same base prompt (step 4 of the counter trajectory, $L \approx 3200$ tokens after chat templating); (ii) each span's size uniformly in $[8, 200]$ tokens at a position uniformly inside the prompt's body; (iii) each replacement length uniformly in $[0, 2{\cdot}|\text{span}|]$, so Δ ranges over both signs. The kernel splices the sampled directives in a single call alongside a substituted-prompt re-prefill for comparison. We report on 40 trials per model per stub mode.

The four-model picture together with the random-vs-semantic comparison gives the honest version of the contract story. DSv2-Lite is invariant: no trial produces a first-token divergence under either stub mode, so the contract's full-vs-rp prediction is vacuous (consistent with Table 4 and the bit-identical stub ablation in Appendix M). Moonlight is the model on which the prediction is cleanly testable: 13–17 trials produce diverging references and Leyline tracks full-context on 11–14 of them, re-prefill on 0, with the remainder matching neither — consistent with the positional contract (Section 3.1) across both stub modes.

JoyAI and GLM behave more brittly under both random-position stub modes: Leyline produces a

Model	Tool	1st-tok vs full	1st-tok vs rp	Mean CP vs full	Mean CP vs rp
DSv2-Lite	6/6	6/6	6/6	128	128
JoyAI-LLM Flash	4/4	3/6	3/6	42	42
GLM-4.7-Flash	2/2	4/6	4/6	6	10
Moonlight-16B-A3B	3/3	6/6	5/6	75	50

Table 4: Cross-architecture replay on a 12-step counter trajectory (6 edit steps; 128-token greedy decode). **Tool**: parsed tool-name agreement among "both-parseable" edit steps. **1st-tok vs full / vs rp**: argmax first-token agreement on edit steps against the full-context baseline and the re-prefill-of-edited-prompt baseline (the two columns are equal by construction on rows where full $\equiv$ rp on every edit step). **Mean CP**: mean common-prefix length, in tokens out of 128. Leyline's positional contract (Section 3.1) predicts agreement with *full*, not *rp*; on Moonlight (the only row where the two references diverge on this trajectory) the prediction holds ($6/6$ vs full $> 5/6$ vs rp). Tool-name agreement is $15/15$ across all models on the steps where both paths chose to act: the directive-driven splice never changes tool selection in this trajectory.

Model (stub)	N	1st-tok vs full	1st-tok vs rp	when full $\neq$ rp: Leyline =full only	when full $\neq$ rp: Leyline =rp only
DSv2-Lite *(rand)*	40	40/40	40/40	0/0 (vacuous)	0/0 (vacuous)
DSv2-Lite *(sem)*	40	40/40	40/40	0/0 (vacuous)	0/0 (vacuous)
Moonlight *(rand)*	40	37/40	23/40	14/17	0/17
Moonlight *(sem)*	40	38/40	27/40	11/13	0/13
JoyAI-LLM Flash *(rand)*	40	0/40	10/40	0/10	10/10
JoyAI-LLM Flash *(sem)*	40	0/40	20/40	0/20	20/20
GLM-4.7-Flash *(rand)*	40	1/40	6/40	1/23	6/23
GLM-4.7-Flash *(sem)*	40	2/40	2/40	2/28	2/28

Table 5: Randomized edit-suite stress test. N=40 trials per model per stub mode with span sizes uniform in $[8, 200]$ tokens, multiple non-overlapping edits per turn (1–2), Δ over both signs. *rand*: stub is uniform in-vocab tokens; *sem*: stub is a tokenized placeholder string ("[truncated content]" repeated). The two stub modes produce nearly identical patterns per row, so the JoyAI/GLM divergence from Table 4 is *not* a stub-content effect; the discriminator is edit *position* (the random-edit harness places spans at arbitrary token offsets, unlike the policy-derived runs which place them at chat-message boundaries). Leyline =full only / Leyline =rp only count diverging-reference trials where the splice agrees exclusively with one reference.

first token that matches neither reference on 20–$36/40$ trials, and never matches the full-context reference exclusively. The random↔semantic comparison rules out stub content as the discriminator: switching from random-in-vocab tokens to a fixed placeholder string changes the rates only slightly (e.g. JoyAI "matches neither" goes 30→20/40, GLM stays 33→36/40). The discriminator versus Table 4 — where the same kernel on JoyAI hits $4/4$ parsed tool-name agreement and $3/6$ first-token agreement — is edit *position*: the random-edit harness splices at uniform-random token offsets inside the chat-rendered prompt body, frequently cutting across chat-template structure (mid-message, mid-tool-tag, between role markers), whereas the policy-derived runs splice at chat-message boundaries the model expects to be edited. We document this as a model-side sensitivity to ill-formed prompt structure, not a kernel correctness finding — the deployment-relevant policies (turn-window truncation, span-aligned eviction, summarization-into-stub) splice at message boundaries by construction. The 18–29 positive-Δ trials per row and 19–21 multi-edit-per-turn trials per row exercise the composition closure in Section 3.3 and the signed-Δ allocator behavior; the patterns above apply uniformly across single-edit/multi-edit and positive/negative Δ subsets.

F Chained-rotation drift in bf16

The composition closure $R(a)R(b) = R(a + b)$ holds in exact arithmetic, so chained rotations should be mathematically equivalent to a single rotation by the sum. Under bf16 we ask whether per-rotation round-off accumulates. We test at the kernel level on synthetic K vectors, comparing N chained random-delta rotations against (a) a single equivalent-sum rotation, and (b) the fresh-RoPE-at-target-position reference. Each N uses 10 seeds; deltas sampled

in $[-512, +512]$ with the running position constrained to the cache range. K vectors are bf16 of shape $[\text{batch}{=}8, \text{heads}{=}32, \text{head_dim}{=}64]$.

Rel-L2 drift grows from $4.3{\times}10^{-3}$ at $N{=}2$ to $2.6{\times}10^{-2}$ at $N{=}100$, a $6.1\times$ growth across a $50\times$ rotation-count increase — sub-linear in N, consistent with random-walk noise accumulation. Workloads where the same physical slot is re-rotated more than $\sim$100 times — e.g. very-long-horizon agentic sessions with hundreds of edits targeting positions downstream of the same persistent prefix — would benefit from periodic re-prefill of the affected slots; we document this as a deployment guideline (Section 8) rather than a correctness gap.

G Long-horizon trajectory replay

The counter trajectory of Section D has 12 steps with 6 edit steps; deployment scenarios commonly run much longer (debug-gym's sum_tree problems regularly produce 50-step trajectories (Yuan et al., 2025)). We replay a 50-step sum_tree trajectory through each of the four MLA models with the same truncation policy as the counter case, separating per-turn-edit counts (the policy's truncation horizon means later turns truncate more older messages: up to 8 edits per turn by step 50). Table 7 reports first-token agreement against the full-context baseline, split between single-edit-per-turn and multi-edit-per-turn subsets.

The single- vs multi-edit split is informative. DSv2-Lite is robust on both subsets even with up to 8 simultaneous edits at L$\sim$28K tokens. GLM-4.7-Flash holds $\geq 80\%$ on both; the multi-edit subset is slightly higher than single, which we attribute to the later-trajectory stubs being well-rehearsed by the model. Moonlight-16B-A3B is the case where long-horizon trajectory replay degrades on *tokens*: first-token agreement drops to $\sim$56% on *single-edit* turns at sum_tree's later steps. JoyAI-LLM Flash degrades further — 12% single-edit and 4% multi-edit token agreement at L$\sim$28K. The full and re-prefill references on JoyAI agree on first-token 82% of the time and on extracted *tool name* $42/43{=}98\%$; Leyline tracks the references at 67% (single-edit) and 72% (multi-edit) on tool-name under a permissive multi-format parser, while diverging at token granularity. We do not attribute this to chained-rotation drift (App. F shows sub-linear growth up to $N{=}100$); the more likely explanation is the same model-side sensitivity to ill-formed mid-template content observed in Table 5, manifesting on a long-prompt-length axis: JoyAI alternates between JSON-list and XML `<tool_call>` output formats turn-by-turn, and small KV perturbations flip which format the model emits even when the tool choice is preserved.

H Cross-architecture rotation-algebra validation

Section 6 claims that the kernel math transfers to GQA/MHA. We validate at the rotation-algebra level (no live serving stack) by sweeping the per-architecture RoPE parameter sets and verifying the closed-form equivalence

$$R(\Delta)\, R(p)\, k \;=\; R(p+\Delta)\, k$$

holds to within bf16 round-off. For each config we sweep 24 (`source_pos`, Δ) cases (4 source positions $\times$ 7 deltas, retaining only those keeping $p, p+\Delta$ inside the cache range) and 5 seeds. K vectors are bf16 of shape $[\text{batch}{=}8, \text{heads}{=}32, \text{head_dim}{=}d]$ per config.

The unit test that produces Table 8 ships with the artifact (no model weights required).

Live-model GQA microbenchmark. To complement the algebra-level validation, we run the constructed Section 4.1-style microbenchmark on two live GQA models with the same splice kernel (`qk_nope_head_dim`=0 degenerate case rotates the full 128-dim K; no architecture-specific code). Each trial is a single decoded greedy token in three paths (full-context, re-prefill, splice). Table 9 reports the result.

I Logit-level distance metrics

Beyond argmax agreement, we capture per-step logit-level distances on the counter trajectory: mean ℓ_2 and mean $\mathrm{KL}(\mathrm{softmax}(\text{leyline}) \,\|\, \mathrm{softmax}(\text{ref}))$ between Leyline and the full-context (*full*) and re-prefill (*rp*) references, computed on the first-position logits in fp32. The means across the 6 counter edit steps are:

We do not develop logit-level metrics as paper claims; they are reported here for completeness and live in the artifact alongside the underlying JSONL records.

J Programmable memory verbs beyond splice

This appendix elaborates the four verbs sketched in Section 7. The directive abstraction is the same;

N	rel-L2 vs fresh	max-abs vs fresh
2	4.3×10^{-3}	4.7×10^{-2}
5	5.6×10^{-3}	6.2×10^{-2}
10	7.5×10^{-3}	7.8×10^{-2}
20	1.1×10^{-2}	1.2×10^{-1}
50	1.7×10^{-2}	1.9×10^{-1}
100	2.6×10^{-2}	3.3×10^{-1}

Table 6: Chained-rotation drift in bf16 against a fresh-RoPE-at-target-position reference. N rotations with random signed deltas, 10 seeds per row, worst-case max-abs / mean rel-L2 across seeds. The N=1 single-rotation bound from Ma et al. (2026) is 4.7×10^{-3} rel-L2; growth from N=2 to N=100 (50× more rotations) is 6.1× in rel-L2 — sub-linear in N, consistent with diffusive noise accumulation.

Model	Valid edit steps	1st-tok @ single-edit	1st-tok @ multi-edit
DSv2-Lite	49	16/16 (100%)	33/33 (100%)
GLM-4.7-Flash	45	13/16 (81%)	26/29 (90%)
Moonlight-16B-A3B	39	9/16 (56%)	13/23 (57%)
JoyAI-LLM Flash	43	2/16 (12%)	1/27 (4%)

Table 7: Long-horizon replay on a sum_tree trajectory with a 50-step policy cap (raw trajectory extends to step 57 on the model with no early termination; 128-token decode; the run ends earlier on some models when the prompt exceeds 90 GiB-per-GPU during decode). *Valid edit steps* excludes steps that raised an OOM during the multi-path comparison. The single-edit subset covers steps 0–21; the multi-edit subset covers steps 22–57 (2–8 edits per turn). Tool-name agreement under a permissive multi-format parser (JSON-list *and* XML tags) is reported for JoyAI in the text below; the other three rows had $\geq 95\%$ tool agreement with the default extractor.

each verb names a distinct semantic intent the agent can declare and the kernel can execute. Experimental validation of each is future work.

Lifecycle hints: `DECAY` for tool-call chains. Multi-step agent workflows have well-defined utility horizons. In a `read` → `analyze` → `write` pipeline, the KV entries for file contents retrieved during `read` are no longer needed once `analyze` completes. A semantic `DECAY` directive issued at the boundary of `analyze` tells the storage layer to deprioritize these entries proactively (e.g., compress and migrate to CPU), freeing GPU capacity for `write` without waiting for end-of-request eviction. The agent knows the phase boundary; no system-side predictor does.

Span isolation: `EVICT` for reasoning traces and failed tool calls. Reasoning models emit extended internal monologues (within `<think>` spans) that are semantically inert to the final output. Under standard serving, these KV entries persist until request completion. A semantic `EVICT` directive at the closing tag enables immediate reclamation, reducing peak memory pressure during long-horizon tasks. This is the canonical splice operation that this paper validates, applied at a span boundary the agent can identify exactly.

A second instance arises naturally in agentic coding: when a `str` replace style `edit` tool call fails because the `old_str` string does not match the current file state, the model retries with a corrected call. Once the corrected call succeeds, the KV span covering the failed attempt — the original `old_str`/`new_str` pair, any interleaved reasoning, and the error response — is semantically dead: it records an action the agent has superseded. An `EVICT` directive emitted alongside the successful retry reclaims this span immediately. The agent possesses the reliable signal (it knows the retry succeeded); the storage layer does not.

Hard retention: `PIN` for deterministic prefetch. Speculative tool-execution work (Nichols et al., 2025) prefetches predicted future tool calls during generation, reducing end-to-end latency but wasting work on mispredictions. A semantic `PIN` directive provides a complementary mechanism: when an agent explicitly declares "the span I am about to emit will be needed in the next turn," the storage layer can treat this as a hard retention guarantee rather than a probabilistic estimate, converting speculation into deterministic prefetch. The block-level Pin in LMCache is the storage-side enforcement; the semantic `PIN` is the agent-side declaration that produces it. Industry deployments are converging on related designs: analysis of

Config	d	rope_theta	worst rel-L2	median rel-L2
MLA (DSv2 / JoyAI / GLM / Moonlight, dense)	64	3.2×10^{7}	4.3×10^{-3}	3.7×10^{-3}
MLA (alternative tuning)	64	10^{4}	4.8×10^{-3}	4.6×10^{-3}
GQA (Llama-3.1-style)	128	5×10^{5}	4.6×10^{-3}	4.1×10^{-3}
GQA (Qwen-3-style)	128	10^{6}	4.7×10^{-3}	4.0×10^{-3}
GQA (Phi-3-style)	96	10^{4}	4.9×10^{-3}	4.5×10^{-3}

Table 8: Cross-architecture rotation-algebra validation. 5 seeds $\times$ 24 (source_pos, Δ) cases per config; worst-case and median rel-L2 between $R(\Delta)R(p)k$ and $R(p+\Delta)k$ in bf16. All configurations agree to the same scale — the GQA/MHA rows are not distinguishable from the MLA rows. The rotation math is architecture-agnostic; only the K slice the kernel acts on (full K vs the K_{pe} band) changes.

Model	Trials	Refs diverge	Leyline ≡full / diverging	Leyline ≡rp / diverging
Llama-3.1-Minitron-4B-Width-Base	8	8/8	1/8	1/8
Qwen3-4B	8	6/8	0/6	6/6

Table 9: Live-model GQA splice microbenchmark on the Section 4.1 constructed prompts (chunk carries the answer; stub elides it). On Minitron-4B-Width-Base (a width-pruned Llama-3.1 base model), the contract holds on $1/8$ trials; the model produces a coherent "number-shaped" first token matching neither reference on the others, with full-context itself producing incorrect arithmetic on most trials (the base model is poor at the underlying arithmetic). On Qwen3-4B, Leyline tracks the re-prefill reference on all $6/6$ diverging trials — the same model-side ill-formed-cache sensitivity Table 5 reports on MLA, manifested more strongly here because GQA fuses position and content into the full K dimensions (rather than separating into $K_{\text{pe}}/K_{\text{nope}}$ as in MLA). The chunk-boundary recomputation of Hu et al. (2025) can be added as a kernel-layer drop-in without changing the directive layer (Section 6).

Claude Code's internals describes a speculative-execution subsystem that pre-runs predicted commands in an isolated sandbox before user confirmation (Spano, 2026), indicating that agent–serving co-design around execution prediction is already a practical concern.

Attention-mask isolation: `SCRATCH`. A natural extension is *scratchpad KV* — context that participates in attention during a reasoning step but is excluded from the output stream and reclaimed immediately on span completion. This requires attention-mask-level isolation beyond the current splice vocabulary and beyond LMCache's block-level Compress, but the directive interface offers a clean extension point: a `SCRATCH` annotation could instruct the kernel to apply the appropriate mask and reclaim entries at span close. The mask-level mechanism is not in this paper; the interface space that admits it is.

Cross-session sharing: `SHARE(scope, placement_hint)`. The four verbs above describe *intra-session, intra-agent* KV lifecycle. A fifth verb extends the vocabulary across the session boundary: *scope* $\in \{$*session*, *user_group*, *global*$\}$ and *placement_hint* $\in \{$*local*, *rank_k*, *replicated*$\}$. `SHARE` differs from `PIN`: `PIN` prevents eviction for the current agent; `SHARE` declares "make this chunk available to other agents." The two compose: `PIN+SHARE` is the canonical pattern for a reference codebase served across multiple refactoring agents. The directive abstraction is the right home for this verb because the *agent*, not the storage layer, knows when a chunk has cross-session intent. The system-side substrate already exists: knowledge-delivery-network architectures (Cheng et al., 2024; Liu et al., 2025) supply the storage tier, compute-vs-load tradeoff schedulers (Jin et al., 2025) the orchestration, RAG-specific shared caches (Lee et al., 2025) the persistence layer, multi-tenant KV reuse systems (Yang et al., 2025; Hu et al., 2024) the cross-instance migration. The closest naming overlaps come from *adjacent* layers: LMCache exposes operator-side `pin/unpin` (Liu et al., 2025) (system-orchestration vocabulary, not agent-emitted), and Anthropic's `cache_control` (Anthropic, 2024) is agent-emitted but single-tenant, single-prefix, and stands alone rather than sitting in a vocabulary. The empirical validation, threat model (cross-tenant prefix-cache leakage in particular), and provider-curated deployment are the subject of follow-up work.

Model	$\ell_2(\text{leyline}, \text{full})$	$\ell_2(\text{leyline}, \text{rp})$	$\text{KL}(\text{leyline} \| \text{full})$	top-10 overlap (vs full)
DSv2-Lite	927	927	1.16	0.52
Moonlight-16B-A3B	471	441	0.13	0.85
JoyAI-LLM Flash	329	339	0.43	0.87
GLM-4.7-Flash	2816	2647	0.20	0.83

Table 10: Logit-level distances on the first decoded token across the 6 edit steps of the counter trajectory (Sec. D). Means across edit steps; computed in fp32 from bf16 model output. On DSv2-Lite the full and re-prefill paths produce bit-identical logits on every edit step in this trajectory, so the two ℓ_2 columns are equal. The pattern in pairwise comparison is consistent with argmax agreement (when argmax matches, KL is small; when it doesn't, KL is several times larger).

K Composition with concurrent and adjacent work

Concurrent signal-side work. The signal-side contributions of SideQuest (Kariyappa and Suh, 2026) (learned semantic-utility detector) and LoopGuard (Xu et al., 2026) (decoder-side degeneration detection) compose naturally with Leyline: any signal expressible as a span-set to evict can drive a Leyline directive without retraining the kernel or the signal detector. The right division of labor in the agentic-LLM stack is signal source in one paper, mechanism abstraction in another.

Active-RAG. FLARE (Jiang et al., 2023) and Self-RAG (Asai et al., 2023) promise uncertainty-triggered retrieval mid-generation, but in production stumble on two coupled obstacles: calibration breakdown in RLHF-trained models, which makes the uncertainty trigger unreliable; and position-encoding invalidation when retrieved content is spliced into a live cache, which forces a re-prefill on every injection. CacheBlend (Yao et al., 2024) addresses the second obstacle on MHA via selective cross-attention recomputation; for MLA, the position-invariant c_{KV} channel plus the δ-rotation of Ma et al. (2026) provides a structurally cheaper resolution. Leyline's directive interface makes the first obstacle tractable by separating the uncertainty oracle (an external policy that decides *when* to retrieve and *what* to inject) from the cache-edit mechanism (the directive that splices without re-prefill). The result is a clean division of labor across the stack: an external oracle supplies policy; the directive supplies the semantic edit; the storage layer (LMCache (Liu et al., 2025), where deployed) handles the resulting block-level movement across the GPU/CPU/storage hierarchy.

Scope notes. Two non-claim items from the main text bear restating. First, we do not develop the side observation that Leyline preserves *more* context than the rendered prompt suggests; that dependence is workload-specific and deserves separate empirical treatment. Second, variation we observe across models is in *agent behavior under Leyline-managed cache* (signal-policy-workload-dependent), not in *whether the directive-driven rotation works*.

L vLLM framework RFC threads

The block-level lifecycle half of the agentic cache problem is now consolidated in production. LMCache (Liu et al., 2025) ships an enterprise-grade control API with Pin, Clear, Move, Compress, and Lookup operating on cache *blocks* in physical storage tiers, integrated with vLLM and SGLang. vLLM's internal `KVEvents` mechanism (consumed by the NVIDIA Dynamo router) covers the introspection side.

The semantic-edit half has not. Four recent vLLM proposals each request a slice of cache control at the layer *above* the storage backend: `[RFC] Pinned Caching` (vLLM Project contributors, 2024) asks for manual-expiration directives over the radix cache (a higher-level Pin than LMCache's block-level Pin); `[RFC] KV-Cache Interoperability API` (vLLM Project contributors, 2025b) proposes to expose `KVEvents` as a public contract for external policy consumers; `[RFC] Sparse KV Cache Management` (vLLM Project contributors, 2025c) sketches a pluggable framework for non-LRU eviction policies; `[RFC] Generalized KV Cache Reuse` (vLLM Project contributors, 2025a) lifts the prefix-only restriction so non-prefix token subsets can be reused, the necessary precondition for splice. None of these proposals defines a span-level edit operation with the position-encoding correctness story that splicing through the kernel requires; each surfaces a piece of the unsolved layer.

The threads on these four RFCs follow a consistent pattern. The maintainers engage substantively, raise corner cases (resource contention when GPU memory runs out under user-issued pins, cross-user prefix-cache leakage under content-addressed cache, hole-filling under H2O-style sparse eviction), then run out of consensus on the right level of abstraction. The Sparse-KV thread is exemplary: contributors converge on the diagnosis that the framework needs to decouple logical from physical memory management so that the scheduler does not have to know which eviction or offloading policy is in effect, but the discussion ends without a maintainer-accepted abstraction (vLLM Project contributors, 2025c). The same shape recurs in the Generalized-Reuse follow-up: with no cache-layer primitive available, contributors propose a request-batching workaround — splitting one prompt into three (`tokens 0-9, 0-19, 0-29`), batching with two as placeholders, discarding the placeholder outputs — to recompute holes outside the cache layer entirely (vLLM Project contributors, 2025a).

Leyline's directive abstraction is the layer this pattern is missing: a single declarative interface in which all four verbs (pin, expose, evict, splice) are well-typed, with one mathematical foundation (the δ-rotation rule) that gives correctness across the vocabulary — and which the kernel applies in place rather than forcing the workaround out to the scheduler layer. The mutation half currently forces deployments into a correctness-versus-reuse tradeoff that Leyline closes.

M Stub-content invariance ablation

The δ-rotation, not the stub text, is the load-bearing operation. We demonstrate this with a $4 \times 3 \times 2$ ablation:

- **4 stub modes**: *faithful* (a short human-readable marker), *pad* (length-matched whitespace), *scrambled* (length-matched noise), *empty* ($|R|$=0).
- **3 trajectories**: counter, shopping_cart, sum_tree from debug-gym mini_nightmare (Yuan et al., 2025).
- **2 models**: DeepSeek-V2-Lite (DeepSeek-AI et al., 2024), JoyAI-LLM Flash (Cai et al., 2026).

On DeepSeek-V2-Lite, all four stub modes produce bit-identical character-level outputs within each trajectory — zero observable variation. On JoyAI-LLM Flash, tool-name agreement holds at ≥ 0.98 across all 12 cells.

Practical implication. Deployments can use $|R|$=0 (the empty stub) and reduce prefill cost further without measurable quality change. This complements Irminsul's stance that content-addressed caching is a first-class primitive: the cache's correctness is governed by the rotation rule and the radix structure, not by the surface form of intermediate stubs.

N Forward compatibility across attention-architecture changes

The δ-rotation correctness story applies to DeepSeek V2/V3-class MLA, where the trained model expects RoPE-encoded keys at their original absolute positions and attends densely over the causal range. Two recent DeepSeek releases test the directive abstraction at different points in the attention-architecture design space — one that preserves MLA and adds trained sparsity (V3.2-Exp), and one that replaces MLA outright (V4).

Within-MLA evolution: V3.2-Exp. DeepSeek-V3.2-Exp (DeepSeek-AI et al., 2025b) retains MLA and adds DeepSeek Sparse Attention (DSA): a learned Lightning Indexer scores each query against MLA's compressed token representations and the model attends only to the top-k selected keys per query rather than to a dense window. Two properties carry over from the dense MLA baseline. First, *causality is preserved*: the indexer selects only from the causal range, and the selected positions remain in their natural causal order. Second, RoPE correctness is per-position: each selected key carries its RoPE encoding at its original absolute position. Sparse attention is a trained-in operation, not a license to feed the model K vectors at arbitrary new positions.

What V3.2-Exp changes for the directive abstraction is the cost of *non-splice* verbs. `PIN`- and `SHARE`-style retention (Appendix J) can be implemented more efficiently when the kernel is sparse-aware: a content-addressed store of pinned spans composes naturally with top-k selection, and the indexer can be biased to include or exclude particular spans without touching positional encodings. A splice that re-anchors downstream positions still requires the δ-rotation correction or an equivalent, because RoPE correctness applies regardless of

which attention pattern — dense or sparse — consumes the keys.

Out-of-MLA evolution: DSv4. DeepSeek-V4 (DeepSeek-AI, 2026) replaces MLA with a Hybrid Attention Architecture: Compressed Sparse Attention (CSA), which compresses the KV cache along the sequence dimension and applies DSA-style top-k selection, and Heavily Compressed Attention (HCA), which applies heavier sequence-dimension compression with dense attention. The $c_{\mathrm{KV}}/K_{\mathrm{pe}}/K_{\mathrm{nope}}$ structure of MLA is no longer present; MLA's per-position RoPE-on-K_{pe} correctness argument does not transfer to a sequence-compressed representation. Whatever position-correctness story V4 admits (CSA-specific, HCA-specific, or layer-mixed) lives below the directive interface. The directive itself remains well-typed: a span $[s_{\mathrm{start}}, s_{\mathrm{end}})$ to splice with replacement R is the same input regardless of how the kernel realizes it. Empirical validation on V4-class architectures is out of scope for this paper.

For the four MLA configurations validated in Section D — DeepSeek-V2-Lite, JoyAI-LLM Flash (DSv3-derived), GLM-4.7-Flash, Moonlight-16B-A3B — splice + δ-rotation is the correctness mechanism, and V3.2-Exp inherits it with sparse-aware optimizations available for non-splice verbs. V4 is the case that motivates keeping policy and mechanism on different sides of the directive: the rotation kernel is MLA-specific and will not survive the architecture replacement, but the span-level interface above it does.

O Discussion notes: governance patterns, release plan, open follow-ups

This appendix expands four items referenced from the Discussion (Section 7): the concrete governance patterns the two-mode directive enables, the open-source release plan, the rotate-only-vs-boundary-recomputation open question, and the concurrency / tenant-isolation guarantees production deployments need on top of the per-request contract.

Governance patterns on top of the mode-declaration contract. Three patterns follow once the API surface declares $m \in \{\text{AMORTIZE}, \text{FORGET}\}$. (i) **Sensitive-content default.** A serving stack that receives directives from harnesses handling PII or regulated data should configure the policy hook to mark such spans FORGET-mode by default and require an opt-in for AMORTIZE. (ii) **Per-edit auditability.** The serving telemetry distinguishes the two modes, so an audit pipeline can answer "which evictions truly erased context vs. amortized it forward" without inspecting the cache. (iii) **Reviewer-debugger consistency.** A debugger or safety auditor inspecting a conversation transcript needs to know whether displayed-prompt edits are AMORTIZE-mode (cache still attends to the original chunk) or FORGET-mode (cache rebuilt past the edit); the API contract surfaces that distinction rather than burying it in serving-stack defaults. We do not claim these patterns close all governance questions; the two-mode split moves them from "hidden serving-stack behavior" to "declared API contract," which is what makes them auditable.

Open-source release plan. The artifacts described in this paper — the MLA K_{pe} rotation kernel, the policy-driven directive API including the AMORTIZE/FORGET mode split, the live SGLang content-hash hook, and the HuggingFace transformers replay path used for Section 4 — will be released under a permissive license on de-anonymization. The release is structured in three layers matching the abstraction stack: a kernel package (rotation primitives, mode-routed splice entry point), a policy SDK (the `Policy.transform` interface plus the in-process directive scheduler), and an integration shim (SGLang radix-cache patch plus an HF-transformers replay harness for offline correctness). The three-arm message-edit microbenchmark of Section B and the constructed-microbench of Section 4.1 are included as reproducer scripts. Subsequent work — vendor / framework-specific adaptations, complementary recomputation strategies under the directive layer (Section 6), and signal-side policies (Section K) — should plug in without forking the kernel.

Open question: when rotate-only suffices vs. when boundary recomputation is needed. The empirical picture of Sections D to 6 is gradiated: on MLA models with well-separated $K_{\mathrm{pe}}/K_{\mathrm{nope}}$ the splice path tracks the full-context reference cleanly at chunk-boundary-aligned edits; under position-arbitrary splices (mid-template) some models exhibit attention shifts away from the rotated suffix that look like the chunk-boundary effects EPIC documents at the reuse boundary (Hu et al., 2025). We hypothesize the relevant architectural cues are (a)

the degree to which K dimensions co-encode content and position (full in GQA/MHA, partial-K in MLA), (b) the attention-sink strength at the chunk-start position (stronger sinks make the rotated suffix more sensitive to boundary mismatch), and (c) the rope frequency band the rotation crosses for typical $|\Delta|$ in the workload. A lightweight runtime detector — decode the first rotated-suffix token under both rotate-only and rotate+k-recompute, and select whichever matches the full-context reference at higher confidence — is implementable in the existing kernel and is the natural follow-up. The directive layer is unchanged in either case.

Concurrency, batching, and tenant isolation. The Section 3.3 pipeline is described per-request; production deployments need three additional guarantees. (i) **Per-request transactionality of multi-directive turns.** A turn carrying k directives must commit all or none. The kernel applies them left-to-right and the rotation algebra closes under composition ($R(\Delta_1)R(\Delta_2)=R(\Delta_1+\Delta_2)$), but the radix update should be a single transaction so a partial failure rolls back to the pre-turn cache state. This matches existing SGLang scheduler semantics; the directive layer adds no new atomicity requirement beyond what the underlying radix-cache writeback already provides. (ii) **Cross-tenant isolation under shared prefixes.** When tenants share a prompt prefix in the radix tree, a directive issued by tenant A at a position downstream of the shared subtree must *not* affect tenant B's view of the cache. Leyline's implementation routes each directive through the tenant-scoped allocator that already owns the slot range past the shared prefix; the rotation operates on the local (per-tenant) copy of K_{pe} and never touches the shared subtree. Concretely the reference cache carries a per-chunk `ChunkEntry.tenant_tag` field with cross-tenant filter on splice candidate selection; default `tenant_tag=None` is the shared-pool opt-in (behaviorally identical to the pre-tenancy baseline), and activation is per-request (operator sets `req._akasha_tenant_tag` from request headers), so per-tenant resource accounting composes with the splice mechanism without a kernel-level change. (iii) **Pinned segments.** The `PIN` verb of Section 7 interacts with the cross-tenant case (does a pinned tenant-A segment block tenant-B from reclaiming allocator budget?); the vLLM RFC thread on pinned caching identifies this as a deploy-side policy question rather than a kernel correctness question (App. L). The directive layer expresses both the edit and the per-tenant scope; the allocator and the radix layer enforce the isolation.

P Responsible research checklist

This appendix concentrates the items of the ACL Rolling Review Responsible Research Checklist that warrant separate elaboration. Item A1 (Limitations) is satisfied by the dedicated Limitations section preceding the bibliography.

A2 Potential risks. Leyline is a serving-side mechanism for KV-cache editing; it does not alter the model weights, training data, or output content. Two deploy-side risks warrant explicit naming, both already surfaced in Limitations and the `[RFC] Pinned Caching` thread (vLLM Project contributors, 2024): (i) cross-tenant prefix-cache leakage under content-addressed caching admits a prefix-guessing side-channel, which explicit user-keyed partitioning at deploy time closes; (ii) user-issued retention directives in future deployments could exhaust KV memory, requiring per-request fail-fast policy. Neither is a property of the directive abstraction itself; both are deploy-side policy work.

B Scientific artifacts (overview). We use four open-weight MLA models, the SGLang serving stack, and the debug-gym mini_nightmare benchmark. We create a ∼200-LOC patch to SGLang's `RadixCache` plus a Python `LeylineCacheManager` class.

B1 Citations of creators. All artifacts are cited at first mention in the main text: DeepSeek-V2-Lite (DeepSeek-AI et al., 2024), JoyAI-LLM Flash (Cai et al., 2026), GLM-4.7-Flash (Team GLM et al., 2024), Moonlight-16B-A3B (Moonshot AI, 2025), SGLang (Zheng et al., 2024), debug-gym (Yuan et al., 2025).

B2 Licenses. All artifacts we use are released under permissive open-source licenses suitable for academic research: the four MLA models under their respective Hugging Face model-card licenses (MIT or Apache-2.0 equivalents); SGLang under Apache-2.0; debug-gym under the upstream Microsoft research license. Our Leyline patch is released under Apache-2.0, consistent with the upstream SGLang project.

B3 Intended use. Each used artifact was released for research and benchmarking; our use is consis-

tent. The MLA models are intended for inference research; debug-gym is intended as an agentic-debugging benchmark; SGLang is intended as a research-and-production serving stack. Our patch is intended for research use on top of SGLang and inherits its access terms.

B4 PII / offensive content. debug-gym mini_nightmare consists of three synthetic Python debugging tasks (counter, shopping_cart, sum_tree); the inputs are program source and hidden tests, containing no human-generated natural-language text, no personally identifying information, and no offensive content. Our experiments do not involve human-subject data collection.

B5 Documentation of artifacts. Documentation of used artifacts is at their respective project pages (Hugging Face model cards, debug-gym repository, SGLang documentation). The Leyline patch is documented via its inline code release accompanying the camera-ready.

B6 Data statistics. The deployment evaluation (Section 5) covers 8 debug-gym mini_nightmare tasks $\times$ 4 seeds $\times$ 2 policy conditions (`truncate_older_than:n=2` as treatment, `keep_all` as baseline), N=64 trials with a 50-step trajectory cap per cell, run under the post-2026-05-26 harness fix described in B2. A ninth mini_nightmare task, pandas_dataframe, was excluded because it requires fetching the Titanic CSV over the network and our compute nodes have no outbound internet access; both policies failed equally on every pandas_dataframe trial we ran (max-steps reached with zero score on 0/3 completed; 5/8 trials terminated in the download retry loop). All returned trials carry `policy_loaded=1` in their per-trial metadata. The stub ablation (Appendix M) covers 4 stub modes $\times$ 3 trajectories $\times$ 2 models. The multi-step replay (Section D) is a single 12-step counter trajectory replayed under three execution paths.

C1 Model size and compute budget. Parameter counts: DeepSeek-V2-Lite (16B total / 2.4B active), JoyAI-LLM Flash (48B total / 3B active), GLM-4.7-Flash (30B / 3B), Moonlight-16B-A3B (16B / 3B). All experiments are inference-only; no model training. Experiments ran on an internal HPC cluster (institution withheld for anonymity) under mixed GPU configurations: GLM-4.7-Flash and JoyAI-LLM Flash served on 4$\times$ NVIDIA H200 with SGLang tensor-parallelism (TP=4); Moonlight-16B-A3B on 2$\times$ H100; DeepSeek-V2-Lite on 1$\times$ H100; auxiliary single-GPU probes (stub ablation, short-trajectory replays) on 1$\times$ H200 or 1$\times$ H100. Per-trajectory wall-clock is on the order of minutes, and the full sweep fits in single-digit GPU-days. Leyline's own kernel overhead is microsecond-scale per slot per the bound inherited from Ma et al. (2026); we do not report end-to-end latency at production batch sizes here (see Limitations).

C2 Experimental setup and hyperparameters. Sampling parameters follow each model's recommended defaults. Policy hyperparameters in Section 5: `truncate_older_than(n=2, max_chars=200, tools=bash)` as treatment, `keep_all` as baseline. Trajectory cap: 50 steps. Seeds: 4 per cell. We did not run a hyperparameter search; the chosen n=2 threshold is the single point on the policy-aggressiveness axis at which the treatment-vs-baseline contrast is reported.

C3 Descriptive statistics. Solve-rate numbers in Section 5 are aggregated across 4 seeds per (task, policy) cell; Table 2 reports per-task fractions and the overall sum. The paired (task, seed)-cell-level disagreement analysis quoted in Section 5 (9 treatment-wins vs 5 baseline-wins) restricts to the 14 of 31 paired cells where the two policies disagreed on the binary solve outcome. Tool-name agreement in Table 4 is measured per-step over a 12-step counter trajectory under Leyline versus the full-context baseline.

C4 Package versions. SGLang 0.5.2 with the Leyline patch; debug-gym 1.3.0.

D Human subjects. The work does not involve human annotators, human participants, or human-subject data of any kind. Items D1–D4 are therefore not applicable.

E1 AI-assistant use. The authors used AI assistants (Claude) for code editing, scripting, prose drafting, and bibliography formatting during the preparation of this paper. All technical claims, derivations, experimental results, and citations were reviewed and verified by the authors; all AI-assisted code was tested by the authors before inclusion; all AI-assisted text was substantially revised before inclusion.