PGCR: DIRECTIONAL CONTINUATION SUBSTITUTION FOR RECURRENT LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reusing eight transformer layers across different requests requires a valid autoregressive handoff, not merely a nearby hidden state. The handoff must jointly replace the post-block residual and every key/value write produced inside the bypassed block, and its validity depends on the ordered source–target pair. Provenance-Gated Continuation Routing (PGCR) learns this directional relation: it searches only completed requests at the same observable workflow position, retrieves with a contrastive subroutine key, reconstructs a full residual/KV continuation from a 512-dimensional prototype, and authorizes substitution with a target-conditioned utility governor. On CodeContests-Edit, SWE-bench-Lite, and ToolBench-Recurring with Llama-3.1-8B-Instruct, PGCR delivers a $1.23\times$ median end-to-end speedup while keeping signed quality changes within -0.7 points. Against a common-interface LLMCache-style similarity route, it lowers the accepted-request 1-to-0 regression incidence from 18.7% to 10.7% on CodeContests-Edit while increasing speed from $1.19\times$ to $1.23\times$; the all-request regression incidence falls from 6.4% to 2.8%. The lower accepted-request incidence appears in every template- and family-separated comparison, including 11.8% versus 19.6% on unseen Code templates. ToolBench-Recurring reaches $1.27\times$ versus $1.17\times$ for exact speculative decoding, and its high-recurrence stratum reaches $1.31\times$. Replication on Qwen2-7B and H100 preserves the method ordering, establishing complete continuation substitution as a practical compute-sharing primitive for provenance-rich recurrent services.

1 INTRODUCTION

A repository agent may repair hundreds of issues with the same model, and a tool service may execute the same schema stage with different arguments. These requests share workflow structure even when their token prefixes do not (Chen et al., 2021; Jimenez et al., 2024; Qin et al., 2024). Recomputing every transformer block ignores that recurrence. Yet blindly inserting a semantically similar activation is consequential: in our CodeContests-Edit stream, an LLMCache-style route accepts 34.2% of requests and changes 18.7% of those accepted requests from a correct full-forward result to failure.

Modern caches already move beyond exact-prefix reuse. Non-prefix and position-independent systems reuse static prompt chunks, and semantic or inter-task systems align KV state across related prompts (Yao et al., 2025; Hu et al., 2025; Luo et al., 2025; Ye et al., 2025; Zhao & Mastorakis, 2025; Liu et al., 2026). Their primary object is prompt-side KV state and their principal saving is prefill work. Reusing a *decode-time internal block* poses a different interface problem: the target needs the residual after the block and the current-token KV write at every skipped layer. It also poses a decision problem because source j can preserve target i even when the reverse substitution fails.

We formalize this decision as *directional continuation substitutability*. At a decoding step, a source continuation is usable when source and target occupy a common observable subroutine position, inserting the complete residual/KV package preserves the target's full-forward outcome, and the saved block time exceeds routing overhead. This relation makes cross-request compute sharing measurable through both outcome transitions and wall-clock utility. It also identifies why similarity alone is insufficient: distance proposes a source, whereas substitutability authorizes an ordered intervention.

PGCR estimates the relation at an eight-layer mid-model boundary. Provenance and workflow position partition a bank of completed clean-prefix steps. Within a partition, a 256-dimensional

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

contrastive key retrieves a source and a 512-dimensional prototype reconstructs the source's post-block residual and pre-RoPE KV writes. A governor then scores the target boundary summary together with the source package and measured skip time. Accepted keys are rotated to the target position before the model resumes above the block; rejected requests execute the native layers and can contribute a clean future entry.

The evaluation isolates the three conditions needed for this mechanism to matter: transferable continuation structure, lower regression incidence than similarity routing, and positive end-to-end utility. At one fixed operating point, PGCR reaches $1.23\times$, $1.19\times$, and $1.27\times$ on CodeContests-Edit, SWE-bench-Lite, and ToolBench-Recurring with signed quality changes of -0.7 to -0.6 points. On CodeContests-Edit, governance reduces accepted-request regressions from 18.7% to 10.7% and all-request regressions from 6.4% to 2.8% while improving speed over the similarity route. The accepted-request ordering survives four unseen-template or disjoint-family comparisons. Recurrence predicts the systems return: high-recurrence Code, SWE, and Tool strata reach $1.29\times$, $1.23\times$, and $1.31\times$.

Taken together, the method and experiments establish a single result: recurrent workflow structure can yield end-to-end block savings when reuse is treated as an ordered intervention over the complete autoregressive interface. Continuation substitutability separates source proposal from target-conditioned authorization; PGCR realizes that factorization through segmented support, contrastive retrieval, joint residual/KV reconstruction, rolling utility governance, and an exact native fallback. Matched frontiers, transfer controls, mechanism ablations, family shift, bank dynamics, and model/hardware replication test each link between recurrence, authorization, and measured savings.

2 Related Work and Technical Positioning

Exact and modular context caches reuse token-identical computation (Kwon et al., 2023; Zheng et al., 2024; Gim et al., 2024). CacheBlend selectively recomputes tokens when combining cached RAG chunks, EPIC links position-independent chunks, and CacheSlide corrects relatively shifted agent-prompt segments (Yao et al., 2025; Hu et al., 2025; Liu et al., 2026). These systems substantially broaden prompt KV reuse while retaining reusable content inside the target prompt. HCache and CAPTURE instead restore same-context state from hidden or hybrid checkpoints (Gao et al., 2025; Lee et al., 2025). All are complementary to a decode-time route that replaces a block using computation produced by another request.

Cross-request semantic reuse is the closest conceptual neighborhood. Sim-LLM reuses top-layer KV state between similar tasks, SemShareKV performs token-level semantic KV matching, and KVCOMM aligns shared context across agent prefixes (Luo et al., 2025; Zhao & Mastorakis, 2025; Ye et al., 2025). LLMCache fingerprints an input, retrieves a semantically similar layer activation, and thresholds the match (Bansal, 2025). Our common-interface LLMCache-style comparator preserves semantic fingerprinting, cosine lookup, thresholded admission, capacity, eviction, and fallback, while giving it the same residual/KV handoff used by PGCR. The controlled difference is the decision model: PGCR adds subroutine-trained keys, provenance–position support, and an ordered source–target governor.

Early exit and adaptive depth reduce layers within one request (Teerapittayanon et al., 2016; Xin et al., 2020; Schuster et al., 2022; Raposo et al., 2024; Elhoushi et al., 2024); speculative decoding reduces serial token steps through target verification (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023; Cai et al., 2024; Li et al., 2024). Retrieval-augmented generation fetches external evidence (Guu et al., 2020; Lewis et al., 2020), while task vectors and model merging share parameter-space changes (Ilharco et al., 2023; Yadav et al., 2023). PGCR occupies the block-execution axis: it transfers one concrete source computation only after estimating its directional effect on the current target.

Table 1 makes the relevant boundary explicit. Prompt-side methods reuse or repair KV state for tokens that appear in the target context; restoration methods recover target-owned state; and adaptive methods alter target-only execution. PGCR instead asks whether a package generated at one source decoding step can replace a target's next eight-layer computation. Segmented support and contrastive retrieval define the candidate set, the codec supplies the complete block interface, and the governor estimates the ordered substitution before execution.

Table 1: Inference reuse mechanisms organized by execution position, transferred object, skipped work, and validity control.

Method	*Transferred object*	*Skipped work*	*Validity control*
Prefix/non-prefix KV	Target prompt KV	Prefill attention	Identity/position repair
Semantic/inter-task KV	Cross-request prompt KV	Prefill or KV storage	Semantic/token alignment
State restoration	Target-owned hidden/KV state	Restore/recompute	Context identity/planner
LLMCache-style	Cross-input residual/KV package	Decoder block	Fingerprint threshold
Early exit/speculation	Target depth/draft tokens	Layers/serial token steps	Confidence/verifier
PGCR	Cross-request residual/KV continuation	Decoder block $b{+}1{:}b{+}s$	Provenance + ordered governor

3 Continuation Substitutability Objective

For target decoding step t and source-bank step u, let $\mathbf{h}_{\ell,t} \in \mathbb{R}^{d_{\text{model}}}$ be the current-token residual after layer ℓ, $\pi(x)$ observable provenance, and $\sigma(x,t)$ the workflow's observable subroutine position. Source request $j \neq i$ supplies a decoded package $\mathcal{C}_{j,u\to t} = (\hat{\mathbf{h}}^{(j,u)}_{b+s,t}, \widehat{\mathrm{KV}}^{(j,u)}_{b+1:b+s,t})$ aligned to target position t. Let $q_i^{\text{full}} \in \{0,1\}$ be target i's full-forward outcome and $q_i(\mathcal{C}_{j,u\to t})$ its outcome when that package is inserted. The ordered governor input

$$\mathbf{z}_{i,t\leftarrow j,u} = (\mathbf{r}_{i,t}, \mathbf{k}_{i,t}, \mathbf{k}_{j,u}, \mathbf{p}_{j,u}, \rho_{j,u}, t_{\text{skip}})$$

retains both endpoints and the source package; nearest-key distance, candidate age and hit count, and bank occupancy are derived features rather than a symmetric replacement for the pair. Writing A for cross-request provenance–position admissibility, directional substitutability and its pre-execution estimator are

$$\begin{aligned} A_{i,t,j,u} &= \mathbb{1}[j \neq i]\,\mathbb{1}[(\pi(x_i), \sigma(x_i,t)) = (\pi(x_j), \sigma(x_j,u))], \\ R_{i,t\leftarrow j,u} &= A_{i,t,j,u}\,\mathbb{1}[q_i(\mathcal{C}_{j,u\to t}) \geq q_i^{\text{full}}]\,\mathbb{1}[U_{i,t\leftarrow j,u} > 0], \\ \widehat{R}_{i,t\leftarrow j,u} &= A_{i,t,j,u}\,\mathbb{1}[\|\mathbf{k}_{i,t} - \mathbf{k}_{j,u}\|_2 < \tau_{\text{match}}]\,\mathbb{1}[g(\mathbf{z}_{i,t\leftarrow j,u}) > 0.5]. \end{aligned} \tag{1}$$

When $q_i^{\text{full}} = 0$, the correctness term records non-degradation. The primary request-level harm estimand is *accepted-route regression incidence*: accepted requests with $q_i^{\text{full}} = 1$ and routed outcome 0, divided by all accepted requests. Its denominator is the system's admission volume, so it measures the regression burden per admitted request rather than conditional risk among full-forward successes. We also report *all-request regression incidence*, dividing the same 1-to-0 count by every request in the stream. Aggregate quality change remains separate because it includes both 1-to-0 and 0-to-1 transitions. Section 5 tests whether governance lowers both incidence measures relative to the common-interface similarity route and whether denser compatible neighborhoods translate into utility.

The route has positive wall-clock utility when

$$U_{i,t\leftarrow j,u} = \underbrace{t_{\text{skip}}(s)}_{\text{saved compute}} - \underbrace{(t_{\text{key}} + t_{\text{lookup}} + t_{\text{decode}} + t_{\text{correct}} + t_{\text{maint}})}_{\text{routing overhead}} > 0. \tag{2}$$

Residual correction is disabled at the reported operating point, so $t_{\text{correct}} = 0$; the term retains optional correction in the generalized cost model. Table 11 shows 7.68–9.10 ms net savings per accepted step after keying, lookup, decode, and amortized maintenance, with 8.4–8.9 MiB resident bank storage.

4 Provenance-Gated Continuation Routing

PGCR estimates Eq. 1 in four stages. Observable provenance and subroutine position define admissible source support, the contrastive key proposes the nearest completed source request, the codec

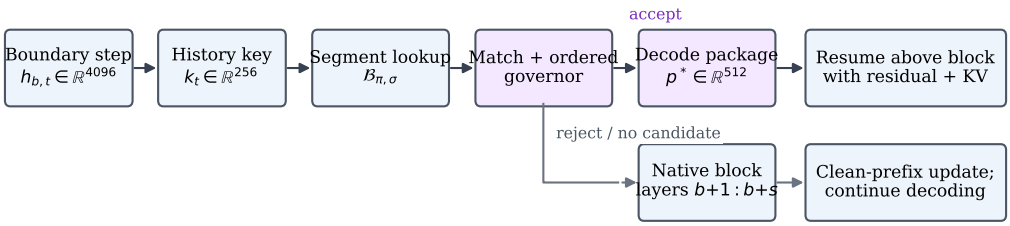

Figure 1: PGCR execution at one decoding step. Retrieval stays inside the observable provenance–subroutine segment and excludes the current request. Acceptance decodes the post-block residual and skipped-layer KV writes; rejection executes the native block, and a clean-prefix fallback may then update the bank.

exposes a complete block interface, and the governor predicts validity for that ordered source–target pair jointly with hardware utility. This factorization keeps candidate generation separate from route authorization.

The learned interfaces are fixed across suites: $f_{\text{key}} : \mathbb{R}^{4096} \to \mathbb{R}^{256}$ produces the normalized retrieval key; E compresses one teacher-forced residual/KV package to $\mathbb{R}^{512}$; D_h and D_{KV} jointly recover the post-block residual and s per-layer KV writes; and g maps the ordered target summary, source descriptors, and systems features to one route score. Only these modules are fitted offline. Online execution performs keying, nearest-source lookup, package decoding, governance, and clean-prefix bank maintenance without updating model weights.

4.1 Execution semantics

PGCR operates at each autoregressive decoding step after a conventional full prefill, which creates exact KV state for every prompt token. At generated token t, the target model executes layers $1{:}b$ and exposes $\mathbf{h}_{b,t}$. The context summary $\mathbf{r}_t = \mathbf{h}_{b,t} + t^{-1} \sum_{v=1}^{t} \text{LN}(\mathbf{h}_{b,v})$ remains 4096-dimensional while incorporating request history, and the key encoder maps it to $\mathbf{k}_t \in \mathbb{R}^{256}$. Retrieval is restricted to the bank segment indexed by $(\pi(x_i), \sigma(x_i, t))$. Candidates therefore share a schema-defined subroutine position but may occur at different absolute token indices. The continuation package stores pre-RoPE keys; after decoding, PGCR applies the model's standard rotation for target index t before appending each reconstructed key, while values require no positional transformation.

A token-granular bank entry is $(\mathbf{k}_{j,u}, \mathbf{p}_{j,u}, \pi_j, \sigma_{j,u}, \text{id}_j, \rho_{j,u})$, where ρ records candidate age and hit count. An entry becomes visible only after its native step has completed, and lookup excludes the current request identifier, enforcing $j \neq i$ even when requests share a batch. Its 512-dimensional prototype $\mathbf{p}_{j,u}$ decodes the residual $\hat{\mathbf{h}}_{b+s,t} \in \mathbb{R}^{4096}$ and one key/value write for the current token at every skipped layer. These writes have shape $2 \times s \times n_{\text{kv}} \times d_{\text{head}}$ and, after target-position rotation of the keys, are appended to the ordinary per-layer KV caches. Later tokens can therefore attend to accepted tokens at the expected layers. Variable request and generation lengths require no padding because each prototype represents one decoding step. Rejected queries execute the native layers on the current trajectory. Only fallbacks whose request history has used native execution at every earlier step can seed the bank, preventing routed histories from becoming sources.

4.2 Learning directional substitutability

The key encoder is trained with InfoNCE (van den Oord et al., 2018):

$$\mathcal{L}_{\text{key}} = -\log \frac{\exp(\text{sim}(\mathbf{k}_i, \mathbf{k}_j^+)/\tau)}{\exp(\text{sim}(\mathbf{k}_i, \mathbf{k}_j^+)/\tau) + \sum_{n=1}^{N_{\text{neg}}} \exp(\text{sim}(\mathbf{k}_i, \mathbf{k}_n^-)/\tau)}. \tag{3}$$

Positive pairs share a labeled subroutine and provenance segment. Thirty percent of training pairs are hard negatives with similar surface form but different argument structure, edit operator, or output

contract. This adapts contrastive and dense-retrieval supervision to continuation compatibility (Chen et al., 2020; Khosla et al., 2020; Karpukhin et al., 2020). The key therefore ranks plausible source continuations; it never authorizes a route by itself.

The encoder E maps the teacher-forced continuation package $\mathcal{C}_t$, including pre-RoPE keys, to $\mathbf{p}_t = E(\mathcal{C}_t)$, and the joint decoder $D = (D_h, D_{\mathrm{KV}})$ reconstructs its post-skip residual and KV writes. Training uses

$$\mathcal{L}_{\mathrm{codec}} = \|D_h(\mathbf{p}_t) - \mathbf{h}_{b+s,t}\|_2^2 + \beta \cdot \mathcal{L}_{\mathrm{next-token}}\big(D_h(\mathbf{p}_t), D_{\mathrm{KV}}(\mathbf{p}_t)\big), \tag{4}$$

with $\beta = 0.5$. The next-token term evaluates the reconstructed residual together with the skipped-layer KV writes, so gradients train both D_h and D_{KV} to preserve downstream autoregressive behavior.

The utility governor receives $\mathbf{z}_{i,t\leftarrow j,u}$: the target boundary summary, both target and source keys, the source prototype, candidate age and hit count, predicted skip time, and bank occupancy. Because target and source occupy distinct input slots, reversing the pair does not preserve the score. Counterfactual labels use the inference-time nearest candidate, insert its package for one step, and complete the target natively; the resulting task outcome attributes a final consequence to exactly the action authorized by one governor call. A label is positive exactly when that candidate satisfies $R_{i,t\leftarrow j,u} = 1$. Deployment applies this rule in receding-horizon form: after every native or routed step, PGCR recomputes the target summary on the resulting trajectory before authorizing another single-step substitution. Thus no score licenses an unobserved sequence of future routes, and summaries after earlier routed steps remain part of later decisions. The gate threshold is 0.5, and τ_{match} is selected once on validation under the accepted-route regression-incidence constraint.

4.3 Data boundaries and online bank

The hierarchy is explicit: a suite contains observable provenance segments; each segment contains task families; a family contains surface templates; and a template can instantiate one or more subroutine positions. Segments are coarser than families: code provenance uses programming language and coarse AST class, repository provenance uses language and issue category, and tool provenance uses the released tool name. Families refine these keys into edit-operator/fine-AST-context pairs, normalized repair motifs, and tool/schema pairs, respectively. At inference time, the workflow exposes the same subroutine position used to index the bank: edit-trace stage for code, test-localized repair stage for repositories, and schema execution stage for tools. Family and template identities remain split-construction variables, not route-time shortcuts.

All suites are partitioned before trace extraction. Offline training fits f_{key}, E, D, and the governor on training families and templates; validation selects thresholds; evaluation streams are never inserted into an offline bank. At serving time, the bank begins empty and grows only from clean-prefix native fallbacks. It holds at most $M = 4096$ token prototypes, uses LRU eviction, and evicts entries with fewer than three lifetime hits first. For Llama-3.1-8B-Instruct, $b = 16$ and $s = 8$, so PGCR routes around layers 17–24.

5 Experiments

5.1 Setup and protocol

We evaluate Llama-3.1-8B-Instruct (Grattafiori et al., 2024) on three forms of recurrent work: CodeContests-Edit (500 AST-level edit requests), SWE-bench-Lite (300 Python repository issues) (Jimenez et al., 2024), and ToolBench-Recurring (400 requests grouped by released tool name and normalized argument schema, retaining groups with at least three instances) (Qin et al., 2024). Code success is pass@1 under the task tests, SWE-bench success follows the Lite resolved-instance criterion, and tool success requires completion of the specified tool task. ToolBench-Recurring is the recurrence-filtered slice; RQ3 additionally evaluates its unfiltered parent pool. Code families are edit-operator/AST-context pairs, templates are normalized before identifiers and literals are instantiated, and subroutines are trace-aligned edit stages. Tool families are tool/schema pairs, templates are normalized requests, and subroutines are schema execution stages. Every split is frozen before traces or counterfactual labels are constructed.

All methods retain ordinary autoregressive KV caches. The exact-cache baseline additionally reuses identical cross-request prefixes atop PagedAttention (Kwon et al., 2023). The LLMCache-style

Algorithm 1 PGCR inference for one autoregressive step

Require: Token state at step t, request ID, clean-prefix flag c, boundary b, span s, provenance $\pi(x)$, subroutine position $\sigma(x,t)$, segmented bank $\mathcal{B}$, thresholds τ_{match} and 0.5

Execute layers $1{:}b$ and form boundary summary $\mathbf{r}_t$
Compute $\mathbf{k}_t = f_{\text{key}}(\mathbf{r}_t)$
if $\mathcal{B}_{\pi(x),\sigma(x,t)} \setminus \{\text{current request}\}$ is nonempty **then**
  $(\mathbf{k}^*, \mathbf{p}^*, \pi^*, \sigma^*, \text{id}^*, \rho^*) = \arg\min_{e \in \mathcal{B}_{\pi(x),\sigma(x,t)},\, \text{id}_e \neq \text{id}_i} \|\mathbf{k}_t - \mathbf{k}_e\|_2$
else
  Mark query as no-candidate
end if
if candidate exists **and** distance $< \tau_{\text{match}}$ **and** $g(\mathbf{z}_{i,t\leftarrow *}) > 0.5$ **and** $(\pi(x), \sigma(x,t)) = (\pi^*, \sigma^*)$ **then**
  Decode $(\hat{\mathbf{h}}_{b+s,t}, \widehat{\text{KV}}_{b+1:b+s,t}) = D(\mathbf{p}^*)$
  Rotate decoded keys to position t, append KV writes, and bypass layers $b{+}1{:}b{+}s$
  Set $c \leftarrow \text{false}$
else
  Execute native layers $b{+}1{:}b{+}s$ and append their KV writes
  if c **then**
    Encode residual and pre-RoPE keys/values; update $\mathcal{B}_{\pi(x),\sigma(x,t)}$
  end if
end if
Continue through layers $b{+}s{+}1{:}L$
return next-token distribution, updated KV caches, and c

similarity baseline uses a boundary-state cosine fingerprint and the same residual/KV package, bank capacity, and fallback interface as PGCR; its unthresholded form routes the nearest available source, and its thresholded form rejects sources below a validation-selected similarity. This gives semantic activation caching a valid autoregressive handoff while withholding only PGCR's contrastive key, provenance–subroutine admission, and ordered governor. Early exit allocates depth from token confidence. Speculative decoding uses a 1.1B draft model and exact target-model verification (Leviathan et al., 2023); under the deterministic decoding protocol it returns the same token sequence and task outcome as full forward, so proposal length changes latency but not quality.

The frontier sweep varies confidence for early exit, cosine threshold for the similarity cache, proposal length for speculative decoding, and match/gate thresholds for PGCR while fixing model, hardware, batch size, sequence limits, deterministic decoding, request streams, and timing boundary. Exact speculative points therefore lie on the zero-quality-change axis. The central PGCR point fixes $b = 16$, $s = 8$, gate threshold 0.5, and $M = 4096$ for every suite.

Quality is pass@1 for code and repository repair and task success rate for tools. Main-table quality reports mean $\pm$ standard deviation over three seeds, and signed quality change is in percentage points relative to full forward. Routing is token-granular, while harm and coverage are request-granular: an accepted instance contains at least one routed step; acceptance divides accepted instances by requests; accepted-route regression incidence divides 1-to-0 full-forward-to-routed transitions by accepted instances; and all-request incidence divides those transitions by the complete stream. Rates are means of seed-wise proportions; appendix counts provide denominator scale and can differ slightly under pooled aggregation.

Speedup uses paired request completion time with request-level bootstrap intervals. Every method runs on the same 8×A100-80GB node with CUDA 12.4, batch size 8, maximum sequence length 2048, identical tokenizer and decoding limits, and method-specific overhead included. Stream order is independently randomized per seed, timing starts before method-specific routing or draft computation, and GPU synchronization occurs at both boundaries. This boundary includes keying, lookup, package decode, verification, fallback, and bank maintenance.

Table 2: Source-selection controls on 500 CodeContests-Edit requests per seed. "Reg./acc." divides 1-to-0 outcome transitions by accepted requests; "Reg./all" divides them by all requests. Quality change is signed pass@1 points from full forward.

Method	*Hit*	*Reg./acc.*	*Reg./all*	*Shortcut*	Δ	*Speed*
LLMCache-style cosine	34.2	18.7	6.4	0.62	-1.8	$1.19\times$
+ Similarity threshold	22.1	12.4	2.8	0.51	-1.2	$1.12\times$
PGCR without hard negatives	28.6	13.1	3.8	0.43	-1.1	$1.22\times$
PGCR	26.3	10.7	2.8	0.28	-0.7	$1.23\times$

Table 3: Continuation transfer under template and family separation. Each split is fixed before fitting; N is requests per seed, and accepted/regression counts are rounded three-seed means. Rate divides regressions by accepted requests. Appendix C gives Wilson reference intervals and paired quality/speed intervals.

Suite	*Split*	*Method*	N	*Accepted*	*Regress.*	*Rate (%)*	Δ*/speed*
Code	Template held out	Similarity	150	46	9	19.6	$-1.7/1.16\times$
		PGCR	150	34	4	11.8	$-0.6/1.19\times$
	Family disjoint	Similarity	125	37	9	24.3	$-3.2/1.14\times$
		PGCR	125	20	2	10.0	$-0.8/1.11\times$
Tool	Template held out	Similarity	120	44	8	18.2	$-1.9/1.19\times$
		PGCR	120	33	3	9.1	$-0.7/1.23\times$
	Family disjoint	Similarity	100	31	7	22.6	$-2.8/1.13\times$
		PGCR	100	16	2	12.5	$-0.8/1.10\times$

5.2 RQ1: Does source–target compatibility survive template and family separation?

This test separates the learned relation from surface recurrence. The key and governor use 30% high-similarity hard negatives to prevent template identity from standing in for computation. An independent difficulty estimator based on full-model perplexity provides the *shortcut ratio*: the fraction of accepted instances classified as easy. It is used only for post-hoc audit and supplies no feature to PGCR.

PGCR is faster than the unthresholded LLMCache-style comparator while reducing regression incidence by 43% per accepted request and 56% over the complete stream (6.4% to 2.8%); its shortcut ratio simultaneously falls from 0.62 to 0.28. Similarity thresholding reaches the same 2.8% all-request incidence only by reducing speed to $1.12\times$, whereas PGCR reaches $1.23\times$. Table 3 then tests whether the accepted-request ordering survives unseen templates and families.

PGCR has lower accepted-request regression point estimates than the similarity route in all four separated comparisons. Under the harder family-disjoint split, it converts weaker support into fewer admissions: the Code incidence is 10.0% versus 24.3%. Across both suites, this ordering accompanies smaller quality changes; Appendix C reports the small-split Wilson intervals.

5.3 RQ2: Does PGCR improve the attainable trade-off?

PGCR's cross-request frontier reaches $1.23\times$ on CodeContests-Edit and $1.27\times$ on ToolBench-Recurring inside the -0.7-point band; the central exact-speculative references reach $1.21\times$ and $1.17\times$ with unchanged outputs. ToolBench provides the clearest speed separation: PGCR's interval is [1.24,1.30], above the speculative interval [1.15,1.19]. Code has a higher PGCR point estimate with overlapping intervals. On the less recurrent SWE-bench-Lite stream, exact speculation reaches $1.22\times$, while PGCR reaches $1.19\times$ at its central point. Together with RQ3, the frontier localizes PGCR's advantage to streams where compatible continuations occur often enough to amortize routing.

The fixed point corroborates the sweep. PGCR is the fastest row on CodeContests-Edit and ToolBench-Recurring while staying within 0.7 quality points, and its accepted-request regression incidences

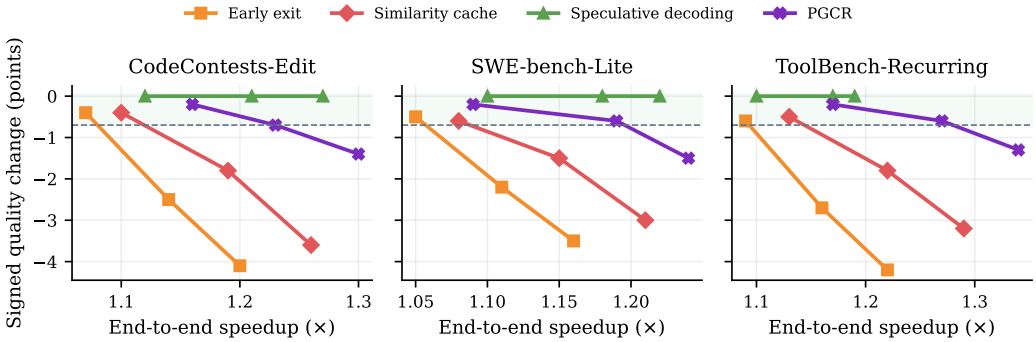

Figure 2: Speed–quality operating points under one evaluation and timing protocol. Curves sweep each method's control; exact speculative decoding remains at zero quality change because target verification preserves the deterministic output. The shaded band marks changes within -0.7 points. Appendix A gives all coordinates and central speed intervals.

Table 4: Main results under the shared protocol. Quality change is signed percentage points from full forward. "Reg./acc." is accepted-route regression incidence; it divides 1-to-0 outcome transitions by accepted instances.

Method	*Speedup*	*Quality*	*Δ pts.*	*Reg./acc. (%)*
CodeContests-Edit				
Full forward	1.00×	42.8±1.2	—	—
KV cache (prefix)	1.08×	42.8±1.2	0.0	—
Early exit	1.14×	40.3±1.4	−2.5	—
Similarity cache	1.19×	41.0±1.3	−1.8	18.7
Spec. decoding	1.21×	42.8±1.2	0.0	—
PGCR	1.23×	42.1±1.2	−0.7	10.7
SWE-bench-Lite				
Full forward	1.00×	28.3±1.5	—	—
KV cache (prefix)	1.06×	28.3±1.5	0.0	—
Early exit	1.11×	26.1±1.6	−2.2	—
Similarity cache	1.15×	26.8±1.5	−1.5	21.3
Spec. decoding	1.18×	28.3±1.5	0.0	—
PGCR	1.19×	27.7±1.4	−0.6	12.4
ToolBench-Recurring				
Full forward	1.00×	51.4±1.3	—	—
KV cache (prefix)	1.09×	51.4±1.3	0.0	—
Early exit	1.16×	48.7±1.5	−2.7	—
Similarity cache	1.22×	49.6±1.4	−1.8	16.8
Spec. decoding	1.17×	51.4±1.3	0.0	—
PGCR	1.27×	50.8±1.3	−0.6	9.3

are 43% and 45% lower than the LLMCache-style similarity cache. It improves both speed and quality over that cross-request comparator on all three suites. Exact speculative decoding preserves full-forward quality; PGCR exceeds its central speed by $0.02\times$ on Code and $0.10\times$ on Tool, while the two central points are adjacent on SWE-bench-Lite.

5.4 RQ3: Does recurrence density track utility?

High-recurrence Code, SWE, and Tool bins reach $1.29\times$, $1.23\times$, and $1.31\times$, while low-recurrence bins remain near parity through fallback. Acceptance rises across tertiles without rising regression

incidence, making family frequency plus boundary-key density an observable utility map. The unfiltered ToolBench pool follows the same response.

5.5 RQ4: Does governance abstain under family shift?

On unseen families (Table 8), PGCR lowers acceptance to 15.2% and holds regression incidence at 10.5%; similarity reuse accepts 27.2% and rises to 29.4%. PGCR converts weaker support into abstention and a -0.9-point quality change.

5.6 RQ5: Do the mechanism and ordering replicate?

On Qwen2-7B-Instruct (Yang et al., 2024), PGCR reaches $1.22\times$ and $1.25\times$; on H100 it remains faster than both comparators with regression incidence near 10% (Table 10). Absolute gains follow Eq. 2, while the ordering persists across model depth and hardware throughput.

5.7 Mechanism ablation

Table 13 isolates each term in Eq. 1: contrastive keys and hard negatives reject surface shortcuts, the downstream loss preserves the block interface, provenance restricts support, and the governor produces the largest regression reduction. Always reuse reaches $1.31\times$ but loses 2.1 points and raises incidence to 24.8%; ordered authorization creates the useful operating point.

6 When Continuation Routing Pays

Family/schema frequency and boundary-key density expose PGCR's regime before outcomes are known: higher support raises acceptance and speed, while family shift raises rejection. Each accepted step saves 7.68–9.10 ms after routing overhead with an 8.4–8.9 MiB bank. The evidence covers structured code, repository, and tool streams, two model families, A100 and H100 hardware, and fixed mid-model boundaries. Because the decision relies on observable workflow position and the local compute-to-routing ratio, sparse or differently provisioned traffic should retain native fallback and calibrate thresholds on representative validation traffic; the held-out, perturbation, sensitivity, and replication results show how this mechanism responds as support changes.

7 Conclusion

PGCR turns repeated workflow structure into a reusable transformer execution object: a complete residual/KV continuation governed for an ordered source–target pair. The resulting route bypasses eight layers while retaining an exact native fallback. Across three suites, PGCR reaches a $1.23\times$ median speedup with signed quality changes within -0.7 points; on CodeContests-Edit it lowers accepted-request regressions from 18.7% to 10.7% and all-request regressions from 6.4% to 2.8% relative to LLMCache-style routing. The ordering survives separated templates and families, grows stronger with recurrence, and replicates on a second model and GPU generation. Directional continuation substitutability therefore provides a concrete design target for recurrent LLM serving: identify compatible work, transfer the full autoregressive interface, and compute natively whenever the pair lacks support.

Reproducibility statement

Sections 4 and 5 specify execution semantics, tensor shapes, objectives, data boundaries, baselines, hardware, timing, metrics, and uncertainty. Appendices A–J provide sweep coordinates, split construction, raw denominators, bank dynamics, replication, perturbation, and sensitivity details.

AI Use Statement

Generative AI tools assisted manuscript preparation, including copyediting, presentation, formatting, and reference checking. This disclosure is limited to the documented preparation assistance and does not assert unrecorded uses.

REFERENCES

Harsh Vardhan Bansal. LLMCache: Layer-wise caching strategies for accelerated reuse in transformer inference. In *Proceedings of the 13th IEEE International Conference on Intelligent Systems and Embedded Design*, 2025. doi: 10.1109/ISED67359.2025.11405274. URL https://doi.org/10.1109/ISED67359.2025.11405274.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023. doi: 10.48550/arXiv.2302.01318. URL https://arxiv.org/abs/2302.01318.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. doi: 10.48550/arXiv.2107.03374. URL https://arxiv.org/abs/2107.03374.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1597–1607, 2020. URL https://proceedings.mlr.press/v119/chen20j.html.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

Shiwei Gao, Youmin Chen, and Jiwu Shu. Fast state restoration in LLM serving with HCache. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 128–143, 2025. doi: 10.1145/3689031.3696072.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 1–19, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. doi: 10.48550/arXiv.2407.21783. URL https://arxiv.org/abs/2407.21783.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3929–3938, 2020. URL https://proceedings.mlr.press/v119/guu20a.html.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Zhang Qin, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24391–24402, 2025. URL https://proceedings.mlr.press/v267/hu25j.html.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 6769–6781, 2020. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *Advances in Neural Information Processing Systems 33*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/d89a66c7c80a29b1bdbab0f2a1a94af8-Abstract.html.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Sanghyeon Lee, Hongbeen Kim, Soojin Hwang, Guseul Heo, Minwoo Noh, and Jaehyuk Huh. Throughput-oriented LLM inference via KV-activation hybrid caching with a single GPU. In *Proceedings of the 43rd IEEE International Conference on Computer Design*. IEEE, 2025. doi: 10.1109/ICCD65941.2025.00045. URL https://ieeexplore.ieee.org/document/11310852/.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948, 2024. URL https://proceedings.mlr.press/v235/li24bt.html.

Yang Liu, Yunfei Gu, Liqiang Zhang, Chentao Wu, Guangtao Xue, Jie Li, Minyi Guo, Junhao Hu, and Jie Meng. CacheSlide: Unlocking cross position-aware KV cache reuse for accelerating LLM serving. In *24th USENIX Conference on File and Storage Technologies*, pp. 83–99. USENIX Association, 2026. URL https://www.usenix.org/conference/fast26/presentation/liu-yang.

Ruikun Luo, Changwei Gu, Qiang He, Feifei Chen, Song Wu, Hai Jin, and Yun Yang. SimLLM: Optimizing LLM inference at the edge through inter-task KV reuse. In *Advances in Neural Information Processing Systems*, volume 38, pp. 99091–99115, 2025. doi: 10.52202/085713-2984. URL https://papers.nips.cc/paper_files/paper/2025/hash/80b677bb4d17f77220be88e1d1fdebde-Abstract-Conference.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024. doi: 10.48550/arXiv.2404.02258. URL https://arxiv.org/abs/2404.02258.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, volume 35, pp. 17456–17472, 2022. doi: 10.52202/068431-1269.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems 31*, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/c4127b9194fe8562c64dc0f5bf2c93bc-Abstract.html.

Surat Teerapittayanon, Bradley McDanel, and HT Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *Proceedings of the 23rd International Conference on Pattern Recognition*, pp. 2464–2469, 2016. doi: 10.1109/ICPR.2016.7900006.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. doi: 10.48550/arXiv.1807.03748. URL https://arxiv.org/abs/1807.03748.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251, 2020. doi: 10.18653/v1/2020.acl-main.204. URL https://aclanthology.org/2020.acl-main.204/.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-Merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36, 2023. doi: 10.52202/075280-0310. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1644c9af28ab7916874f6fd6228a9bcf-Abstract-Conference.html.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024. doi: 10.48550/arXiv.2407.10671. URL https://arxiv.org/abs/2407.10671.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109, 2025. doi: 10.1145/3689031.3696098. URL https://doi.org/10.1145/3689031.3696098.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. KVCOMM: Online cross-context KV-cache communication for efficient LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://openreview.net/forum?id=yGOytgjurF.

Xinye Zhao and Spyridon Mastorakis. SemShareKV: Efficient KVCache sharing for semantically similar prompts via token-level LSH matching. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 432–445, 2025. doi: 10.18653/v1/2025.findings-ijcnlp.25. URL https://aclanthology.org/2025.findings-ijcnlp.25/.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000.

A SHARED-PROTOCOL FRONTIER COORDINATES

Table 5 supplies the coordinates used by Figure 2. Each tuple is speedup/signed quality change under the shared protocol.

Table 5: Conservative, central, and aggressive sweep coordinates under the shared protocol. Entries are speedup/signed quality change; exact speculative decoding remains quality preserving.

Suite	*Method*	*Control swept*	*Conservative*	*Central*	*Aggressive*
Code	Early exit	Confidence	1.07/−0.4	1.14/−2.5	1.20/−4.1
	Similarity cache	Cosine threshold	1.10/−0.4	1.19/−1.8	1.26/−3.6
	Spec. decoding	Draft length	1.12/0.0	1.21/0.0	1.27/0.0
	PGCR	Match/gate	1.16/−0.2	1.23/−0.7	1.30/−1.4
SWE	Early exit	Confidence	1.05/−0.5	1.11/−2.2	1.16/−3.5
	Similarity cache	Cosine threshold	1.08/−0.6	1.15/−1.5	1.21/−3.0
	Spec. decoding	Draft length	1.10/0.0	1.18/0.0	1.22/0.0
	PGCR	Match/gate	1.09/−0.2	1.19/−0.6	1.24/−1.5
Tool	Early exit	Confidence	1.09/−0.6	1.16/−2.7	1.22/−4.2
	Similarity cache	Cosine threshold	1.13/−0.5	1.22/−1.8	1.29/−3.2
	Spec. decoding	Draft length	1.10/0.0	1.17/0.0	1.19/0.0
	PGCR	Match/gate	1.17/−0.2	1.27/−0.6	1.34/−1.3

At the central PGCR points, paired speedup 95% intervals are [1.20,1.26], [1.16,1.22], and [1.24,1.30]. The corresponding speculative intervals are [1.19,1.23], [1.16,1.20], and [1.15,1.19]. PGCR's SWE-bench-Lite quality-change interval is [-1.0,-0.2]; exact speculation remains at zero by construction. The speed intervals separate on ToolBench-Recurring, overlap on Code, and place the methods in the same speed region on the less recurrent SWE stream.

B LEARNED-COMPONENT AND EXECUTION CONTRACT

Table 6 collects the interfaces that remain fixed from offline fitting through evaluation. It makes explicit which state is learned, which state is observed online, and which boundary protects native execution.

Table 6: PGCR contract used in every central experiment.

Component	*Input*	*Output / objective*	*Online role*
Key encoder	4096-d target summary	256-d normalized key; InfoNCE with 30% hard negatives	Segmented nearest source
Package codec	Native residual and pre-RoPE KV writes	512-d prototype; residual plus downstream next-token loss	Complete block hand-off
Governor	Ordered target/source descriptors and timing	Binary counterfactual utility label	Route score at threshold 0.5
Support rule	Provenance, subroutine, request ID, clean-prefix flag	Completed cross-request entries only	Candidate eligibility
Fallback	Current target state	Native layers $b{+}1{:}b{+}s$ and exact KV writes	Default execution and bank update

Offline fitting uses training families and templates only; validation selects the match threshold under the accepted-route regression-incidence constraint. Evaluation starts with an empty online bank, excludes the current request from lookup, and never imports evaluation traces into an offline bank. These boundaries apply to the template-held-out, family-disjoint, shift, and replication protocols.

C HELD-OUT CONSTRUCTION AND RAW TRACE

Partitions are created from normalized templates before full-forward traces are generated. Template-held-out evaluation shares the provenance segment and family label but shares no normalized request template. Family-disjoint evaluation shares the suite-level provenance vocabulary but withholds every family used to fit the key, codec, governor, or thresholds.

Table 7: Raw construction trace for the disjoint-transfer table. Eligible requests satisfy the suite's ordinary evaluation criteria; candidates are mean compatible entries available at query time.

Suite	*Split*	*Eligible*	*Mean cand.*	*Full successes*	*Shortcut*
Code	Template held out	150	38.6	64	0.30
Code	Family disjoint	125	17.9	54	0.32
Tool	Template held out	120	41.2	62	0.27
Tool	Family disjoint	100	15.6	51	0.31

Using the rounded mean per-seed counts in Table 3, Wilson 95% reference intervals for PGCR accepted-request regression incidence are [4.7,26.6], [2.8,30.1], [3.1,23.6], and [3.5,36.0]; corresponding similarity-cache intervals are [10.7,32.9], [13.4,39.5], [9.5,31.9], and [11.4,39.8]. These quantify representative-seed binomial uncertainty rather than a pooled three-seed interval. Paired quality/speed intervals are [-2.3,-1.1]/[1.13,1.19], [-1.0,-0.2]/[1.16,1.22], [-4.0,-2.4]/[1.11,1.17], [-1.3,-0.3]/[1.08,1.14], [-2.5,-1.3]/[1.16,1.22], [-1.1,-0.3]/[1.20,1.26], [-3.6,-2.0]/[1.10,1.16], and [-1.4,-0.2]/[1.07,1.13].

D FAMILY-SHIFT COUNTS AND UNCERTAINTY

Table 8 pairs the in-support stream with a family-disjoint stream under the frozen routing rule.

Table 8: Family-shift test on CodeContests-Edit. Each cell is seen/unseen family; the unseen partition contributes no offline fitting data.

Method	*Speedup*	Δ *pts.*	*Accept inst. (%)*	*Match reject (%)*	*Reg./acc. (%)*
Similarity cache	1.19/1.14×	$-1.8/-3.4$	34.2/27.2	65.8/72.8	18.7/29.4
PGCR	1.23/1.11×	$-0.7/-0.9$	26.3/15.2	53.4/66.4	10.7/10.5

The seen partition contains 500 requests per seed and the held-out family partition contains 125. PGCR accepts 132/128/134 requests with 14/13/15 regressions across the three seen streams; under shift it accepts 18/20/19 with 2/2/2. Its paired speedup intervals are [1.20,1.26] and [1.08,1.14]; paired quality-change intervals are [-1.0,-0.4] and [-1.3,-0.5]. No-candidate rates are 20.3/18.4, completing the acceptance/rejection partition and showing where coverage is spent under shift.

E RECURRENCE CONSTRUCTION AND COMPLETE RESPONSE

Recurrence is defined before outcome evaluation as the geometric mean of normalized family/schema frequency and full-forward boundary-key density inside the validation-selected radius. Equal-count tertiles are then frozen. RQ3 uses dedicated randomized stratification streams, so its bins test association rather than partitioning Table 4; ToolBench uses the unfiltered parent pool.

Paired speedup 95% intervals from low to high recurrence are [1.01,1.05]/[1.15,1.21]/[1.26,1.32], [0.99,1.03]/[1.07,1.13]/[1.19,1.27], and [1.00,1.04]/[1.13,1.19]/[1.27,1.35]. All three high-recurrence intervals exclude parity, whereas the low-recurrence intervals remain at or near it; the monotone speed response follows acceptance rather than rising regression incidence.

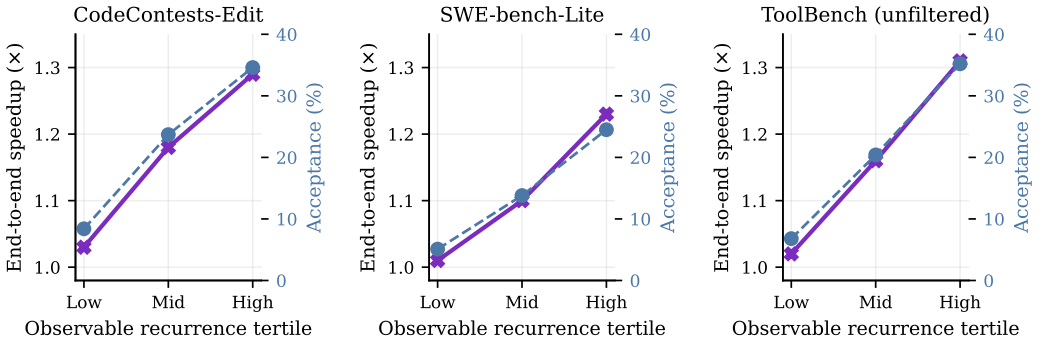

Figure 3: Performance versus a pre-outcome continuation-recurrence score. Tertiles combine observable family/schema frequency and boundary-key neighborhood density. Curves show three-seed mean speedup and acceptance.

Table 9: Complete PGCR recurrence response. Values are denominator/acceptance (%)/accepted-request regression incidence (%)/signed quality change/speedup.

Suite	*Low recurrence*	*Middle recurrence*	*High recurrence*
Code	167/8.4/10.2/−0.2/1.03	166/23.7/10.8/−0.6/1.18	167/34.6/10.5/−0.8/1.29
SWE	100/5.1/14.0/−0.1/1.01	100/13.8/12.8/−0.4/1.10	100/24.5/11.9/−0.8/1.23
Tool (unfiltered)	267/6.8/11.7/−0.2/1.02	266/20.4/9.8/−0.5/1.16	267/35.2/9.1/−0.7/1.31

F Replication Protocol Details

Qwen2-7B-Instruct has 28 transformer layers, so the replicated boundary and span preserve the main protocol's depth fractions with $b = 14, s = 7$. The H100 replication retains the Llama architecture and $b = 16, s = 8$. Every row uses batch size 8, maximum sequence length 2048, three seeds, identical requests and decoding limits, and paired synchronized timing. Rounded mean per-seed PGCR accepted/flip counts are 119/13, 109/11, 126/13, and 106/10; table rates average seed-wise proportions. Paired speedup intervals are [1.19,1.25], [1.22,1.28], [1.17,1.23], and [1.20,1.26].

Table 10: Orthogonal replication. Qwen2-7B-Instruct uses $b = 14, s = 7$ on A100; Llama uses $b = 16, s = 8$ on H100. Entries are speedup/signed quality change/accepted-request regression incidence.

Model/hardware	*Suite*	*Spec. decoding*	*Similarity cache / PGCR*	
Qwen2-7B, 8×A100	Code	1.20/0.0/–	1.18/−1.7/17.9	1.22/−0.8/11.2
Qwen2-7B, 8×A100	Tool	1.16/0.0/–	1.20/−1.9/17.3	1.25/−0.7/9.8
Llama-3.1-8B, 8×H100	Code	1.18/0.0/–	1.17/−1.8/18.3	1.20/−0.7/10.5
Llama-3.1-8B, 8×H100	Tool	1.15/0.0/–	1.20/−1.8/16.4	1.23/−0.6/9.6

Across both replications, PGCR remains faster than the similarity route and has lower regression incidence. Its Tool speed also exceeds exact speculation on Qwen/A100 and Llama/H100, matching the recurrence-heavy ordering in the central study.

G Online Bank Dynamics

Table 11 resolves the utility terms in Eq. 2 before Figure 4 follows bank growth from a cold start.

The measured routing overhead is 1.30–1.42 ms per accepted step, leaving 7.68–9.10 ms after subtracting keying, lookup, decoding, and maintenance from native block time. Figure 4 connects that local margin to request-stream behavior from an empty bank.

Table 11: Latency and bank-cost decomposition at $M = 4096$. Times are milliseconds per accepted decoding step, with stream-wide bank maintenance amortized over accepted routes; memory is resident bank storage in MiB. Values are means over three randomized streams.

Suite	t_{skip}	t_{key}	t_{look}	t_{dec}	t_{maint}	*Net saved*	*MiB*
Code	9.80	0.31	0.18	0.74	0.09	8.48	8.7
SWE	9.10	0.32	0.21	0.78	0.11	7.68	8.4
Tool	10.40	0.30	0.17	0.75	0.08	9.10	8.9

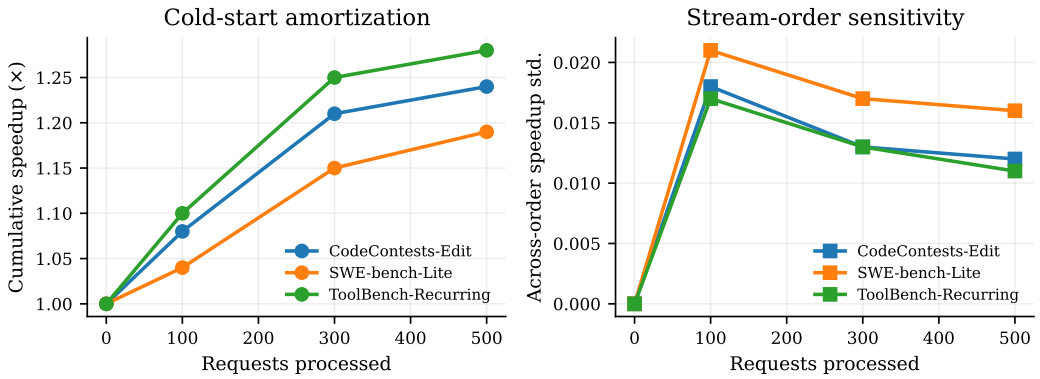

Figure 4: Online bank dynamics on dedicated 500-request arrival streams from empty-bank start. Left: cumulative end-to-end speedup as compatible entries accumulate. Right: standard deviation across independently randomized stream orders. All timings include keying, lookup, decoding, update, and eviction.

The dynamics experiment samples separate 500-request arrival streams from each suite's evaluation-family distribution; these streams are distinct from the fixed main-table denominators. At requests 100/300/500, cumulative speedups are 1.08/1.21/1.24, 1.04/1.15/1.19, and 1.10/1.25/1.28. Across-order speedup standard deviations are 0.018/0.013/0.012, 0.021/0.017/0.016, and 0.017/0.013/0.011. Alternative capacities $M = 1024/2048/8192$ require 2.6/4.7/16.8 MiB and produce 1.17/1.21/1.24 speedup. The plateau beyond $M = 4096$ shows that central utility comes from compatible recurrence rather than unbounded bank growth.

H Provenance and Bank-Integrity Matrix

The protected object is the continuation package; the enforcement boundary comprises metadata validation, segmented retrieval, match filtering, the utility governor, and fallback. Observable provenance metadata and the full-forward target model are trusted. The bank may contain stale, high-similarity, or incompatible candidates, and metadata may be noisy.

Table 12: Systems robustness matrix. Results are rejection rate/signed quality change/overhead change. PGCR is compared with similarity-based activation reuse under the same perturbation.

Perturbation	*Capability*	*Similarity cache*	*PGCR*	*Observed failure mode*
20% metadata noise	Misroute family label	4.1/−3.7/+0.1%	78.4/−1.0/+1.8%	Valid but mislabeled requests fall back
Incompatible insertion	Add foreign candidates	8.7/−4.2/+0.3%	96.1/−0.8/+2.0%	Same-segment matches define residual exposure
Hard-negative burst	Surface-match flood	12.6/−3.0/+0.6%	84.7/−0.9/+2.2%	Dense ambiguous neighborhoods abstain
Stale prototypes	Delay bank refresh	9.8/−2.6/+0.2%	71.9/−1.1/+1.5%	Old but compatible keys reduce coverage

PGCR rejects 71.9–96.1% of perturbed candidates and keeps signed quality change between -1.1 and -0.8 points. The similarity route rejects only 4.1–12.6% and loses 2.6–4.2 points, directly linking segmented support and governance to conservative behavior under bank faults.

I Mechanism Sensitivity

The deletion study isolates the learned estimator's components, and the following sweep varies its principal operating axes.

Table 13: Ablation on CodeContests-Edit. Signed quality change is in points; regression incidence uses all accepted instances as its denominator.

Variant	*Speedup*	Δ *pts.*	*Reg./acc. (%)*	*Shortcut*
Full PGCR	1.23×	−0.7	10.7	0.28
− Contrastive keys (cosine)	1.21×	−1.4	16.2	0.47
− Utility governor (always reuse)	1.31×	−2.1	24.8	0.54
− Hard negatives	1.22×	−1.1	13.1	0.43
− Downstream codec loss ($\beta = 0$)	1.23×	−1.3	11.4	0.29
− Provenance control	1.24×	−0.9	12.1	0.31

Removing the governor produces the largest speed increase and the largest regression increase; removing contrastive keys produces the largest shortcut increase after that always-route variant. The remaining rows isolate package fidelity, hard-negative supervision, and provenance support.

Table 14: CodeContests-Edit sensitivity. Results are speedup/signed quality change/acceptance/accepted-request regression incidence.

Axis	*Conservative*	*Central*	*Aggressive*
Match threshold	1.16/−0.3/17.2/8.9	1.23/−0.7/26.3/10.7	1.29/−1.5/36.8/17.6
Gate threshold	1.17/−0.4/18.1/9.2	1.23/−0.7/26.3/10.7	1.30/−1.8/39.4/20.9
Prototype dim.	1.24/−1.1/26.0/13.8	1.23/−0.7/26.3/10.7	1.21/−0.6/26.5/10.2
Skip span	1.14/−0.3/28.1/8.8	1.23/−0.7/26.3/10.7	1.28/−1.6/21.9/18.4
Boundary	1.18/−1.0/20.7/15.1	1.23/−0.7/26.3/10.7	1.17/−0.4/30.2/9.7

Threshold and span sweeps expose the expected speed–regression exchange, while prototype dimension and boundary position show diminishing returns. The central operating point is the only displayed setting shared unchanged across all three suites.

J Raw Denominators and Uncertainty Convention

Table 2 averages 500 requests per seed. Mean accepted/regression counts are 171/32, 111/14, 143/19, and 132/14. In every pair, the first count is all accepted requests and the second is the subset with a 1-to-0 full-forward-to-routed outcome. Dividing the second count by all 500 requests gives the reported all-request incidences 6.4%, 2.8%, 3.8%, and 2.8%. Table 4 uses 500, 300, and 400 requests per seed for CodeContests-Edit, SWE-bench-Lite, and ToolBench-Recurring; pooled PGCR accepted/regression totals are 395/42, 121/15, and 354/33. Main-table rates average seed-wise proportions rather than recomputing a pooled ratio. Table 13 uses the same 500 CodeContests requests and three independently randomized streams.

Paired request bootstrap intervals resample request identities and retain each method's paired latency and outcome before recomputing speedup or quality change. Wilson reference intervals use rounded mean per-seed accepted instances as the denominator and flips as the numerator where stated. The paired-bootstrap convention is used for frontier, shift, recurrence, and replication comparisons; Table 4 standard deviations remain seed-level variation.