CACHECOHERENT: FREQUENCY-ALIGNED DOCUMENT ORDERING FOR PREFIX-CACHE-EFFICIENT MULTI-TENANT RAG SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-tenant retrieval-augmented generation (RAG) serving has heavy-tailed, highly reused document access, underexploited by prefix caching. We show that corpus-level document frequency is an effective, low-cost ordering prior: placing frequent documents first needs no cache-state history, giving a transient advantage that narrows as an adaptive orderer accumulates history. A strictly causal table retains 76% of the ordering gain under streaming re-estimation and $\approx$99% under a frozen past-only table. An ablation isolates the mechanism: frequency alignment drives the gain, while content-hash ordering adds nothing significant. No lookahead-free adaptive variant we test beats static frequency ordering under the non-causal protocol of that comparison, and it is indistinguishable from static at roughly ten times the ordering cost. On top of frequency ordering, position-weighted WPA routing contributes a further increment, reaching **71.8% HR overall, a** $+23.1$ **pp headline** over the no-reorder baseline (the weakest comparator); the ordering gain reproduces across model families. WPA is a *balanced* router, not the highest-HR one: it trades a little raw HR for load balance, though per-card isolation makes that imbalance cost 20–27% throughput. These cache gains carry a quality cost: on NQ extractive QA, F1 falls from **42.5% to 9.9%** under full frequency ordering, and no evaluated strategy attains both acceptable F1 and HR; no generative task was measured, so the mechanism behind this task- and position-dependent trade-off is not isolated. In the evaluated settings, static historical frequency ordering is the primary mechanism, effective without future information and deployable independently of WPA, with topology- and task-dependent trade-offs; WPA is optional, its benefit depending on topology and load balance. In a matched held-out serving protocol, past-only frequency ordering showed a small positive mean difference over cache-dependent ordering ($+1.0$ pp, $n = 5$), but the comparison is underpowered and does not establish superiority.

1 INTRODUCTION

Multi-tenant retrieval-augmented generation (RAG) workloads exhibit heavy-tailed document access with high cross-user reuse; prior characterization reports the top 5% of chunks serving 60% of requests at 94% reuse Agarwal et al. (2025). Serving engines Kwon et al. (2023); Zheng et al. (2024a) implement radix-tree prefix caching that eliminates redundant KV computation for shared prefixes; when RAG requests share retrieved documents, their prefixes can overlap and be cached. In practice it is underexploited. Cache-aware reordering systems—ContextPilot Jiang et al. (2026), CacheWeaver Tan et al. (2026)—inspect cache state and rearrange documents to match cached prefixes, so they require convergence: they start cold on new deployments, after evictions, and during scaling events, and cannot operate until sufficient cache history accumulates.

We show that static corpus-level frequency statistics provide an alternative requiring no cache-state history: frequency-aligned ordering places the most frequently accessed documents first in every prompt, reaching its operating level without the warmup an adaptive orderer needs. It presupposes a precomputed frequency table, unavailable in a true cold start; we do not measure the cost of building it, and our claims concern cache-state warmup only. Unlike adaptive methods, it is proactive

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.

(dictating what *should* be cached from corpus statistics) rather than reactive (discovering what *was* cached from runtime history). Figure 1 contrasts the fragmentation and consolidation outcomes.

CacheCoherent composes frequency-aligned ordering with position-weighted Weighted Prefix Affinity (WPA) routing, reaching 71.8% HR on the main 2-worker evaluation (1000 req $\times$ 5 seeds), a $+23.1$ pp gain over the no-reorder baseline, of which $+3.3$ pp is attributable to WPA routing. The $+1.1$ pp over cache-dependent greedy reordering is a mechanism-isolation contrast, not a system-level win over prior work, and WPA sits ~ 0.9 pp below consistent hashing on raw HR, trading that for load balance. A causality control retains 76% of the ordering gain under streaming re-estimation (or $\approx 99\%$ frozen); the gain reproduces on Mistral-7B ($+25.29$ pp) as on Qwen ($+25.97$ pp). Our contributions:

- **Frequency alignment as the primary cache-ordering mechanism**: a controlled ablation shows that corpus-frequency ordering drives the cache hit-rate gain while content-hash ordering adds nothing ($p = 0.86$, bounded by a 15-seed cohort). The advantage is transient — a cache-dependent orderer closes the gap as its history accumulates — but frequency ordering needs no cache-state history at all, making it immediately available after a cold start. Under a matched held-out serving protocol, the measured difference over cache-dependent ordering is small and underpowered ($+1.0$ pp, $n = 5$), so no superiority is claimed.
- **Ordering-dominant system with topology-dependent routing**: CacheCoherent reaches 71.8% HR ($+23.1$ pp over the unordered baseline) with ordering contributing $\sim 85\%$ of the gain and WPA routing $\sim 15\%$, approximately additive. WPA is a balanced cache-affinity router, not the strongest one; under per-card isolation it costs 20–27% throughput, so frequency ordering alone (without WPA) is the recommended configuration for isolated deployments, at a total system overhead of 0.069 ms at p99.
- **Quality-cache Pareto frontier on extractive QA**: frequency ordering reduces NQ extractive QA F1 from 42.5% to 9.9% (-32.6 pp), and no tested mitigation simultaneously satisfies F1 $\geq 25\%$ and HR $\geq 55\%$. No generative RAG task was measured, so this penalty is specific to position-sensitive extractive QA; without per-question data the causal mechanism behind the drop is not isolated.

2 Related Work

RAG serving and KV caching. RAG Lewis et al. (2020) creates long prompts with cross-request overlap in multi-tenant settings. Cache-Craft Agarwal et al. (2025) characterizes this overlap (94% reuse) and manages chunk caches; RAGCache Jin et al. (2026), RAGO Jiang et al. (2025a), PipeRAG Jiang et al. (2025b), and FusionRAG Wang et al. (2026a) optimize RAG latency through caching and pipeline parallelism. Prefix caching via radix trees Zheng et al. (2024a); Kwon et al. (2023) is standard, with extensions for modular reuse Gim et al. (2024), compression Du et al. (2026); Liu et al. (2024b), hotness-aware admission Li et al. (2025), and prefetch-enhanced reuse Wang et al. (2026b). Position-independent caching (PIC) systems such as CacheBlend Yao et al. (2025) and EPIC Hu et al. (2025) reuse cached segments at any position and may not pay the ordering-induced quality cost we report; we have no PIC quality measurement, so our frontier is scoped to prefix-cache-compatible reordering. Cache-aware reordering systems ContextPilot Jiang et al. (2026) and CacheWeaver Tan et al. (2026) reorder per-request from cache state, requiring convergence from cold start.

Distributed serving and scheduling. Preble Srivatsa et al. (2025), consistent hashing Karger et al. (1997), and DualMap Yuan et al. (2026) balance cache affinity with load. Serving infrastructure systems NVIDIA (2025); llm-d contributors (2025); Zhong et al. (2024); Patel et al. (2024); Qin et al. (2025); Yu et al. (2022); Sheng et al. (2024); Lin et al. (2024) route on existing prefixes without manipulating ordering; Dexter et al. Dexter et al. (2025) prove prefix-aware scheduling is NP-hard. BatchLLM Zheng et al. (2024b) reorders *requests* by common prefix; CacheCoherent instead fixes the *document order inside each prompt* using corpus-level frequency. A discrimination test (Appendix H for the per-seed detail) finds intra-prompt frequency ordering is the operative lever, while inter-request prefix sorting did not reproduce as a distinct effect. BlendServe Zhao et al. (2026) reorders offline-batch requests; LMCache Liu et al. (2026) changes the cache substrate (a tiered KV store), orthogonal to ordering.

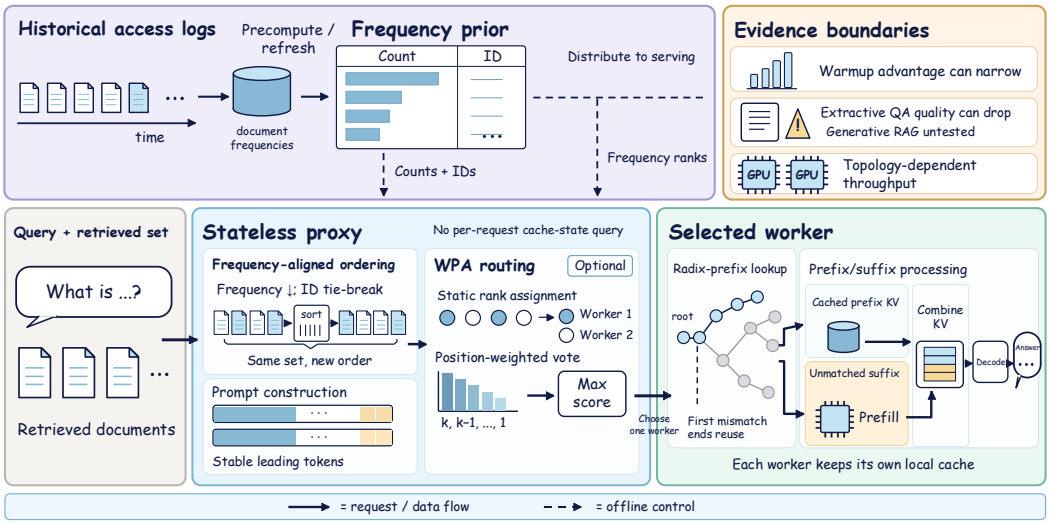


Figure 1: **CacheCoherent overview.** *Problem*: adaptive ordering emits a different order per request, so the same documents land at different prefix positions and each worker's radix-tree prefix cache fragments into duplicate entries. *Method*: an offline global frequency table drives frequency-aligned ordering plus position-weighted WPA routing, both stateless. *Payoff*: identical orderings across requests let each worker's prefix (KV) cache consolidate its reused prefixes into one shared path in its own radix tree, raising hit rate while WPA keeps load balanced across workers.

Position effects and theory. Liu et al. Liu et al. (2024a) document position-dependent quality degradation; OP-RAG Yu et al. (2024) shows preserving original chunk order yields a quality-cache tradeoff as context grows. Our exchange argument Smith (1956) connects to LFU theory Sleator & Tarjan (1985) and online algorithms Borodin & El-Yaniv (1998). Multi-tenant cache security Zhang et al. (2026); Wu et al. (2025); Sun et al. (2026); Wu et al. (2026); Chu et al. (2025) is discussed in Appendix A (for the threat model and proposed mitigations).

3 Method

We present CacheCoherent, a system for distributed prefix cache optimization in multi-tenant RAG serving. CacheCoherent combines frequency-aligned ordering (a stateless document sort by corpus frequency) with Weighted Prefix Affinity (WPA) routing (load-balanced cache-affinity assignment).

3.1 Problem Formulation

Consider a distributed LLM serving system with W inference workers $\{w_1, \ldots, w_W\}$, each maintaining a local radix-tree prefix cache Zheng et al. (2024a). A corpus $\mathcal{D}$ contains $N = |\mathcal{D}|$ documents. Each query q has a retrieved document set $S_q \subseteq \mathcal{D}$ with $|S_q| = k$ (typically $k = 3$–5). The *ordering function* $\pi : 2^{\mathcal{D}} \to \mathcal{D}^*$ determines the concatenation order. The *routing function* $\rho : \mathcal{D}^* \to \{1, \ldots, W\}$ assigns the request to a worker. We maximize system-wide prefix cache hit rate:

$$\max_{\pi,\rho} \mathbb{E}_{q\sim\mathcal{Q}} \left[\mathrm{HR}\left(\pi(S_q),\ \mathcal{C}_{\rho(\pi(S_q))}\right)\right], \tag{1}$$

where $\mathcal{C}_j$ denotes the cache state of worker j and $\mathcal{Q}$ is the query distribution.

3.2 Why Frequency Ordering Outperforms Adaptive Strategies

Under a radix-tree prefix cache, a hit extends only while leading token blocks are identical between the incoming request and a cached entry; a mismatch at position j terminates the hit regardless of deeper matches, so position 1 contributes to every cache hit while position k contributes only when all preceding positions match.

Proactive vs. reactive ordering. Adaptive strategies Jiang et al. (2026); Tan et al. (2026) are reactive: they inspect current cache state $\mathcal{C}$ to decide ordering, discovering what *was* cached from runtime history. Frequency ordering is proactive: it dictates what *should* be cached from corpus statistics, and so requires no cache-state history at all — the frequency table is precomputed before serving begins, so the policy is fully specified from the first request. Its advantage over a cache-dependent orderer is therefore transient: the two converge as that orderer's history accumulates — the gap narrows to $+0.4$ pp at 2000 requests. This concerns *cache-state* convergence only; the frequency table itself must exist before ordering applies.

Exchange argument. Let p_j denote the probability that position j is shared between two independent requests, assuming positions are independent. The expected shared prefix length is

$$\mathbb{E}[L(\pi)] = \sum_{j=1}^{k} \prod_{m=1}^{j} p_m. \tag{2}$$

Since every $p_m \leq 1$, sorting by decreasing p_j maximizes Eq. 2: by an exchange argument Smith (1956), swapping an adjacent pair with $p_a < p_{a+1}$ changes only the depth-a term, by $\prod_{m<a} p_m \cdot (p_{a+1} - p_a) \geq 0$, and leaves deeper terms unchanged. Under Zipfian access this coincides with decreasing document frequency, an analogue of LFU caching Sleator & Tarjan (1985). The bound is tight when all documents are identical. This is a finite-k result under the independence assumption; retrieval-induced correlations may narrow the gap. The argument requires only the weaker conjecture that per-position match probabilities increase monotonically with frequency rank, which the mechanism ablation (Section 4.3) supports empirically.

3.3 Frequency-Aligned Ordering

Frequency-aligned ordering defines π as a deterministic function of the document set:

$$\pi_{\text{freq}}(S_q) = \text{sort}_{d \in S_q}\big(-n_d,\ \text{id}(d)\big), \tag{3}$$

where n_d is the document frequency from access logs and $\text{id}(d)$ provides deterministic tiebreaking. The priority table $\{(-n_d, \text{id}(d))\}$ is precomputed offline and updated at corpus-change intervals (hours to days). Our main results (Section 4.2) fit the table on the complete evaluation trace; the causality control of Section 4.7 re-estimates it from past-only statistics and retains 76% of the ordering gain under streaming re-estimation. Per-request cost: $O(k \log k)$ with $k \leq 5$, yielding 0.069 ms at p99.

3.4 Weighted Prefix Affinity (WPA) Routing

Frequency-aligned ordering produces deterministic prefixes, enabling WPA, a simple cache-affinity heuristic based on static hash-like assignment, to predict worker-prefix affinity without querying cache state. Each document d has a static worker assignment $\text{assign}(d) = \text{rank}_{\text{freq}}(d) \bmod W$, precomputed from the same frequency table. For a prefix $\pi(S_q) = (d_1, \ldots, d_k)$ ordered by decreasing frequency, the routing decision is:

$$w^* = \arg\max_{w \in \{1,\ldots,W\}} \sum_{i=1}^{k} (k+1-i) \cdot \mathbb{1}[\text{assign}(d_i) = w], \tag{4}$$

where i indexes the position in the frequency-sorted prefix: the weight $(k{+}1{-}i)$ gives earlier positions (more frequent documents) higher influence, consistent with the prefix-length argument of §3.2. The $\arg\max$ is computed in $O(W)$ time by iterating over workers. In the 2-worker mechanism ablation (Section 4.3), WPA routing adds $+3.3$ pp, so routing contributes $\sim$15% of the total gain against ordering's $\sim$85%. How that split behaves beyond $W{=}2$ is an open confound: the configurations we hold differ in worker count and card topology at once, so all reported claims are scoped to $W{=}2$.

3.5 CacheCoherent System

CacheCoherent composes frequency-aligned ordering ($O(k \log k)$) and WPA routing ($O(kW)$): sort S_q by the precomputed frequency table, then route via WPA voting. Total overhead: 0.069 ms at

p99 for typical k=5, W=4. The system requires no per-request cache state tracking and operates as a stateless proxy compatible with any radix-tree-based serving engine (vLLM, SGLang).

Deployment. The frequency table is computed offline from access logs and loaded at startup; updates are batched at corpus-change intervals (hours to days) and applied atomically without serving interruption. The WPA assignment is derived from the same table, so both components update together. At inference time, each incoming request passes through two stateless steps: (1) the ordering proxy sorts the retrieved documents by the precomputed priority, and (2) the routing proxy evaluates the WPA score for each worker and forwards the request. Neither step queries the cache, and the proxy can run on the load balancer or as a sidecar. The system presupposes a precomputed frequency table, which is unavailable during a true cold start (before any access logs exist); our claims concern cache-state warmup after the table is available, not the table construction cost itself.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Cohort naming. The *main cohort* is Table 1's 1000-request, 5-seed, two-card configuration; the *C-group* (300–350 req, 3–5 seeds, one shared card) covers ordering dominance and router comparison; the *B-group* (350 req, 3 seeds, 60/20/20 split) is the adaptive-orderer family. Labels C1–C3 name *claims*, not cohorts.

Infrastructure. We deploy vLLM v0.27.1 Kwon et al. (2023) with automatic prefix caching; each worker is a separate vLLM instance serving Qwen2.5-7B-Instruct. The main experiments run on NVIDIA A800-80GB GPUs at `gpu_memory_utilization=0.60` (2 workers); the causality analyses (Section 4.7) run on one RTX PRO 6000 Blackwell card at 0.45, and the leakage analysis on a card whose memory budget was not recorded. Cache hit rate (HR) is the token-ratio `vllm:prefix_cache_hit_rate` (cached over queried prefix tokens).

Workloads. Three categories. (1) *Synthetic*: WildChat-based Zhao et al. (2024) multi-tenant RAG, Zipf(α=1.0) access, 500 documents, 512-token mean length, 3–5 docs/request; 1000 req $\times$ 5 seeds for the main experiments. (2) *Real-document*: Natural Questions (NQ) Kwiatkowski et al. (2019), 200 documents, 10 tenants, 313 req $\times$ 3 seeds (NQ selection pipeline in Section 4.6). (3) *Scenario-calibrated*: search engine (15% overlap), customer support (60%), legal research (35%), each 300 req $\times$ 3 seeds.

Baselines. No Reorder + RR (baseline), Cache-Dependent + RR, Frequency-Aligned + RR, CacheCoherent (Frequency-Aligned + WPA, full system), No Reorder + WPA, CacheWeaver Tan et al. (2026) (greedy LCP), Hybrid (frequency + runtime adaptation), and SGLang-style prefix-tree routing Zheng et al. (2024a).

4.2 MAIN RESULTS

Table 1 presents the main results (1000 req $\times$ 5 seeds): 71.8% HR, +23.1 pp over baseline, from a table fitted on the full evaluation trace. Section 4.7 reports the matched-configuration control scoping the causal reading of this result, and the routing trade-off. All main results are at W=2 workers; scaling beyond this configuration is not validated.

Frequency-aligned ordering gains $+1.1$ pp over cache-dependent reordering on the main-table cohort (Table 1; paired $t = 11.3$, $n = 5$), a narrow margin that we report as an interval rather than leaning on the p-value. On the pooled 15-seed set (5@1000 + 10@500), the per-seed deltas (all 15 positive, mean $+23.99$ pp) are the paper's strongest non-parametric evidence: an exact two-sided sign test gives $p = 6.1 \times 10^{-5}$ at Cohen's $d_z = 3.58$. The strata are pooled because a permutation test does not separate them ($p = 0.72$; Appendix D.2); a separate 2000-request convergence run confirms the transient-decay pattern: the gap narrows to $+0.4$ pp at 2000 requests as the adaptive orderer accumulates history. Figure 2 summarizes the core results.

Table 1: **Consolidated results.** *Panel A*: main results (2-worker vLLM, 1000 req × 5 seeds, Prometheus HR). *Panel B*: mechanism ablation (300 req × 5 seeds, single shared card); frequency alignment is the specific mechanism. SD is the sample standard deviation (ddof = 1). The matched-configuration control spans both the ordering and the worker assignment; Table 4 gives the causal comparison.

Ordering	Routing	HR (%)	SD (%)	Δ (pp)	State
Panel A: Main results (1000 req × 5 seeds, two A800 cards)					
None	RR	48.8	8.5	—	None
Cache-dep.	RR	67.4	2.7	+18.6	Per-worker
Freq-aligned	RR	68.5	2.8	+19.7	None
Freq-aligned	**WPA**	**71.8**	**2.5**	**+23.1**	**None**
None	WPA	52.1	8.6	+3.3	None
Panel B: Mechanism ablation (300 req × 5 seeds, single shared card)					
No reorder (baseline)	RR	38.8	8.8	—	None
Random-fixed	RR	36.9	5.1	−1.8	None
Content-hash	RR	40.4	14.1	+1.7	None
Frequency sort	**RR**	**61.6**	**2.9**	**+22.8**	**None**

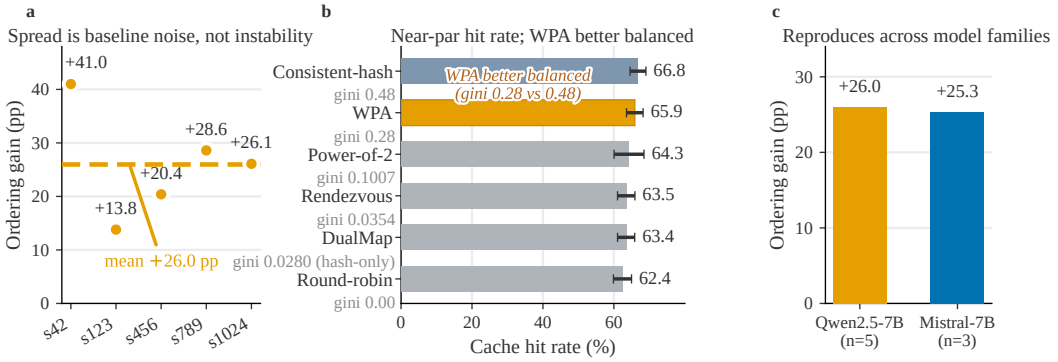


Figure 2: **CacheCoherent core results.** (a) Ordering dominance is a distribution, not a point: the C-group per-seed gain ranges +14 to +41 pp (mean ∼ + 26 pp), the spread being baseline variance rather than method instability. (b) Router comparison on canonical ordering: WPA sits at near-par HR to the highest-HR scheduler (consistent hash) while being *better* balanced (gini 0.28 vs. 0.48), and is not itself the highest-HR router. (c) The gain reproduces across model families (Qwen +26.0 pp, n=5; Mistral +25.3 pp, n=3). Panels (a)/(b) are at the C-group's 350-request scale, not Table 1's 1000-request scale. In panel (b), the power-of-two, rendezvous, and DualMap gini annotations are from the separate router matrix rather than this cohort.

Ordering dominance (C1): mean ± SD across seeds. Throughout, SD denotes the sample standard deviation (ddof = 1). The per-seed ordering gain across the C-group matrix (5 seeds, 350 req, 2 workers) averages ∼ + 26 pp with SD ≈ 10 pp (individual seeds in Figure 2a). This is baseline variance, not method instability: the baseline's SD = 9.0 pp exceeds the method's 2.3 pp, so 79% of the gain variance is baseline and 5% method. At $n = 5$ these results are underpowered; the strong non-parametric claim rests on the 15-seed cohort.

Cross-model robustness (C3). Ordering dominance is model-agnostic: on Mistral-7B-Instruct-v0.3 (350 req × 3 seeds) the gain is +25.29 pp (canonical-WPA 0.665 ± 0.024 vs. no-reorder-RR 0.412 ± 0.103), against +25.97 pp on Qwen2.5-7B ($n = 5$). HR is a token-ratio metric[1], so absolute levels are not cross-model comparable; the *gain* reproduces across both model families.

Variance reduction. The same cohort shows a 3.4× reduction in across-seed spread (8.5 vs. 2.5 pp sample SD; Table 1). This stability property is distinct from the mean gain: a high-variance baseline

[1]All hit rates use `vllm:prefix_cache_hit_rate`: cached prefix tokens / queried prefix tokens. Because it counts tokens rather than documents, values are tokenizer-dependent and not cross-model comparable in absolute terms.

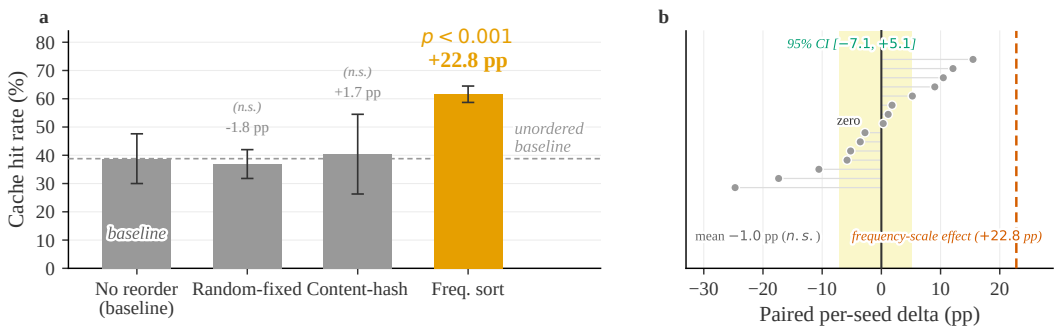


Figure 3: **Mechanism specificity: frequency alignment, not determinism.** (a) The three non-frequency orderings (gray) sit within noise of the unordered baseline (dashed), while frequency sorting (orange) gains $+22.8$ pp ($p < 0.001$). (b) The content-hash null is *bounded*: 15 paired seeds scatter from -24.7 to $+15.5$ pp around a mean of -1.0 pp, and the 95% CI $[-7.1, +5.1]$ excludes the $+22.8$ pp frequency effect (dashed vermillion). Panel (b) is its own α=1.5 cohort.

means operational HR is unpredictable without ordering, whereas CacheCoherent's tight spread delivers a reliable cache benefit across seeds. The reduction is driven almost entirely by the ordering component, since WPA routing contributes only $\sim$15% of the total gain.

Routing analysis (C2): a balanced cache-affinity router, not the strongest one. The C-group matrix (5 seeds, 350 req, canonical ordering fixed) compares seven policies, including round-robin, consistent hashing Karger et al. (1997), power-of-two choices Mitzenmacher (2001), rendezvous hashing Thaler & Ravishankar (1998), DualMap, and CacheWeaver. WPA gains $+2.9$ to $+4.1$ pp over round-robin (mean $+3.5$ pp). Consistent hashing takes the highest raw HR (beating WPA 5/5 by $\sim +0.9$ pp) but concentrates load far more (gini 0.45–0.50 vs. 0.25–0.31, single-worker saturation reaching 350/0). **We do not claim WPA is the strongest router.** A 4-condition decoupling ablation confirms the routing gain reflects generic cache affinity rather than the shared frequency table: substituting a uniform or random assignment table for WPA's frequency-derived one leaves HR within 0.25 pp of the coupled system (66.35/66.12/66.43%), while round-robin sits at 62.73%, so the gain comes from clustering prefix-sharing requests on a worker, not from the specific table that drives the clustering. A separate 300-request, 9-condition matrix confirms the decomposition: consistent hashing concentrates on a dominant worker (occurrence share 0.98 vs. WPA 0.78; Gini 0.48 vs. 0.28; cold-worker reuse 0.11 vs. 0.59), trading balance for hot-prefix collection; ordering dominates routing by an order of magnitude ($\sim$+22.8 pp vs. 1–3 pp). In that matrix, WPA is second-least-balanced of the six affinity-assigning routers: power-of-two choices buy 2.8$\times$ better balance (0.285 $\rightarrow$ 0.101) for 1.45 pp less HR. DualMap is exercised in hash-only form (no SLO/affinity inputs), so WPA's +2.5 pp advantage is against this neutered form, not DualMap's full design.

Cache-aware and prefix-tree baselines. CacheWeaver's greedy longest-common-prefix ordering Tan et al. (2026) matches the preceding request, which under round-robin optimizes the wrong worker's cache; we use its faithful *lookahead* variant (+18.8 pp over no-reorder) rather than the weak, seed-dependent cache-state variant (+4.7 pp). Frequency ordering's +26.1 pp exceeds lookahead by $\sim$7–8 pp (350 req $\times$ 3 seeds); SGLang-style prefix-tree routing Zheng et al. (2024a) adds +6.1 pp.

Adaptive baseline (B-group): a budget artifact with no measurable payoff. Under a 1.0 ms decision budget, an adaptive orderer scores 0.4348 vs. static's 0.6596 (-22.5 pp), missing the budget on 90% of requests (mean decision cost 2.48 ms vs. 0.154 ms) and falling back to the original order. Lifting the budget collapses the deficit to -1.06 pp ($t(2) = -0.57$, 95% CI $[-9.14, +7.02]$): unconstrained, adaptive is indistinguishable from static. A lookahead-assisted variant leads by $+1.64$ pp, but that arm receives both lookahead and adaptivity, so the gain is not attributable to adaptivity; the cost is $\sim$10$\times$ the ordering time for no measurable HR gain. Per-seed values are in Appendix D.4.

Table 2: CacheCoherent advantage across 6 workload scenarios (300 req × 3 seeds, 42/123/456). Rows 1–3 are synthetic workload types, rows 4–6 real-world-calibrated. Δ is canonical-WPA minus baseline over per-seed means. For rows 4–6 the *baseline* column's sample SD (ddof = 1) is 2.6 / 10.5 / 4.5 pp versus the CacheCoherent column's 2.3 / 1.0 / 3.5 pp; per-seed values are in the appendix.

Scenario	Overlap	Baseline (%)	CacheCoherent (%)	Δ (pp)
Synthetic workload types				
Enterprise FAQ	high	61.7	75.9	+14.2
Multi-tenant RAG	medium	25.6	46.4	+20.8
General chat	low	22.0	35.3	+13.4
Real-world-calibrated scenarios				
Search engine	15%	21.1	35.1	+14.0
Customer support	60%	37.8	54.7	+16.9
Legal research	35%	24.6	37.9	+13.3

4.3 Mechanism Ablation

Table 1 (Panel B) decomposes frequency alignment from other deterministic orderings (300 req × 5 seeds; Figure 3). Determinism alone is insufficient: random-fixed ordering imposes a consistent order yet degrades HR, and content-hash ordering adds nothing detectable at frequency scale ($p = 0.86$, $n = 5$; a dedicated 15-seed cohort bounds the null at 95% CI $[-7.06, +5.14]$, excluding the $+22.8$ pp frequency effect). The 2-worker decomposition is approximately additive (interaction < 0.2 pp): ordering contributes $+19.7$ pp (85%) and WPA routing $+3.3$ pp (15%).

Baseline scale note. The no-reorder baseline is lower in the shared-card tables (38.8%/38.7%) than in Table 1 (48.8%). The configurations differ in card topology, memory budget, and request count at once, so no single factor is attributed. The ordering *gain*, however, is consistent across both configurations: frequency alignment adds $+19.7$–$+22.8$ pp regardless of the absolute baseline level, suggesting the mechanism is robust to the serving environment.

4.4 Workload Robustness

Table 2 shows CacheCoherent provides +13.3–20.8 pp across all 6 scenarios, with $\Delta > 0$ on every seed of every real-world-calibrated row: the search-engine delta is the most robust ($t(2) = 5.08$, $p = 0.037$); but customer support's baseline is unstable, of the same order as its $+16.9$ pp delta, so that row is not distinguishable from noise here. The reported run is the second; the first overflowed the window and returned HR $= 0$ (Appendix C).

4.5 Real Workload Validation

On Natural Questions with real documents and natural overlap patterns (313 req × 3 seeds, 10 tenants, 200 documents), CacheCoherent rescues prefix caching from near-collapse: baseline achieves only 6.4% HR versus 20.5% (paired $t = 37.6$, $p \ll 0.001$; TTFT -12.3%; per-seed baseline 6.35/6.71/6.13% and canonical-WPA 19.70/21.25/20.52% for seeds 42/123/456). These are this configuration's HRs, not §4.6's denser-overlap 124-question selection.

4.6 Quality-Cache Pareto Frontier

Table 3: Quality-cache Pareto on NQ extractive QA (Qwen2.5-7B, 124 scored of 500 × 3 seeds). No strategy reaches F1 ≥ 25% and HR ≥ 55%.

Strategy	F1 (%)	HR (%)
No reorder (base)	42.5	13.2
Query-adaptive	30.1	13.1
Top-2 rel. + freq	13.4	26.1
Marked-relevance	11.2	65.7
Full frequency	9.9	69.6

Document reordering trades off against generation quality under this model, prompt format, retrieval pipeline, and NQ subset. From 500 sampled NQ questions, 313 carry gold short-answer annotations and 124 satisfy an answerable-from-context criterion (gold answer appears as a substring in at least one retrieved passage); the quality analysis uses the 124-question subset, while the real-workload

validation (Section 4.5) uses the full 313-question set. We characterize the tradeoff with a 5-strategy ablation on NQ extractive QA (124 scored of 500 sampled $\times$ 3 seeds), shown in Table 3. The per-question artifact underlying these aggregates is no longer recoverable, so the frontier cannot be re-derived from per-question inputs; aggregates are verified cell-by-cell against a surviving gold-rescore artifact.

Full frequency ordering achieves the highest cache HR but reduces extractive QA F1 from 42.5% to 9.9% (-32.6 pp), and no evaluated strategy reaches the desired region (F1 $\geq$ 25% and HR $\geq$ 55% simultaneously). F1 is *set-overlap* F1 against NQ short answers, not SQuAD-style token-F1, and is never merged with the harness's inline response-vs-document metric. The drop is observed under position-sensitive extractive QA with k=5 frequency ordering — it moves the gold-answer document from position 0 to a uniform draw over 0–4, removing the primacy advantage Liu et al. (2024a) — and may differ for generative tasks, which were not measured; the specific mechanism is not isolated without per-question data. The best HR-preserving mitigation, *marked-relevance* (F1 11.2%), recovers only +1.3 pp F1 over full frequency ordering, and that may come from the marker token alone. WPA routing is lexically near-transparent at fixed ordering, while ordering substantially alters content (ROUGE-L 0.157 ± 0.294). Attention-based reordering Peysakhovich & Lerer (2023) and positional-bias calibration Hsieh et al. (2024) remain untested.

4.7 Staleness, Overhead, and Throughput

Table 4: **Causal companion**: five conditions, one self-consistent run on a single A800 card (W=2). The first 600 requests fit the frequency table; the last 200 are the test window (pre-warmed by a 200-request validation). Arms are matched in protocol but not in past-information source.

Condition	HR (%)	TTFT (ms)
No reorder + RR	24.5	144.3
Cache-dep. + RR	39.7	120.3
Past-only freq + RR	40.7	118.8
Past-only freq + WPA	42.9	115.0
Full-trace oracle + WPA	43.0	115.1

Causality control: is the ordering an artifact? The main results fit the frequency table on the full trace. The matched-configuration control measures leakage within a single run: its arms share trace, split, workers, and WPA, differing only in the frequency table — so it spans both ordering and routing rather than isolating either. Retention differs by protocol (streaming 76%; frozen past-only $\approx$99%; Appendices E, G). Table 4 gives the causal control: past-only frequency + RR gains $+16.2$ pp over baseline, WPA adds $+2.2$ pp, and the oracle adds $+0.1$ pp. That table uses a different protocol from Table 1 and *does not validate its headline*. This table establishes the benefit of a strictly causal frequency prior relative to no reordering. A matched held-out comparison against cache-dependent ordering leads in 5/5 seeds (mean $+1.003$ pp, SD 0.621; $t(4) = 3.61$, $p \approx 0.023$), but at $n = 5$ the sign test cannot reach 0.05 (floor 0.0625): the difference is small and underpowered, and no superiority is claimed. The test window is pre-warmed, not a cold start. The routing gain is measured on a shared card, where WPA's imbalance costs nothing; under per-card isolation it loses 20–27% throughput (five seeds; Table 12).

Staleness and training-window sensitivity. A staleness check (frequency table fit on the first 50% of the trace, evaluated on held-out requests) retains 99.4% of the ordering gain. Varying the training window (10/25/50/100% of a 350-request trace) gives $+22.1$–$+22.3$ pp; the 10% window already recovers 99.3% of the oracle gain. Under Zipf(α=1.5) skew this reflects head concentration rather than universal sample efficiency, so deployments with different skew may need larger windows.

5 Conclusion

CacheCoherent uses static corpus-level document frequency to order RAG prompts for prefix-cache efficiency, reaching 71.8% HR ($+23.1$ pp) with ordering contributing $\sim$85% of the gain. The gain reproduces across model families and six workload scenarios ($+13.3$ to $+20.8$ pp). These cache gains carry a quality cost: NQ extractive QA F1 falls from 42.5% to 9.9% under full frequency ordering, with no evaluated mitigation simultaneously preserving quality and cache benefit. Future work should address scaling beyond W=2 workers, long-run behaviour past 2000 requests, generative-RAG quality measurement, and the untested cache side-channel mitigations.

REPRODUCIBILITY STATEMENT

All experiments use publicly available models and open-source infrastructure (vLLM v0.27.1). The main evaluation uses Qwen2.5-7B-Instruct; a cross-model check uses Mistral-7B-Instruct-v0.3, and the quality and real-workload studies use Qwen2.5-7B-Instruct. Request counts and seed counts vary by cohort and are stated with each table and result rather than globally: the main results are 1000 requests $\times$ 5 seeds, the C-group and ablation cohorts 300–350 requests $\times$ 3–5 seeds, and the pooled non-parametric cohort 15 seeds. Cache HR is the token-ratio `vllm:prefix_cache_hit_rate` (cached prefix tokens / queried prefix tokens), a tokenizer-dependent quantity. Frequency-aligned ordering requires only a document frequency table and a deterministic sort. WPA routing assigns documents to workers round-robin by frequency rank and routes by position-weighted prefix affinity votes, where the weight on position i of a k-document prefix is $(k{+}1{-}i)$ (Section 3.4). Hardware: main experiments, mechanism ablation, and quality-cache evaluation on NVIDIA A800-80GB GPUs at `gpu_memory_utilization=0.60` (the throughput comparison's dual-card row also uses two physical A800 GPUs); the causality and window families on a single NVIDIA RTX PRO 6000 Blackwell card, the leakage analysis on a single card whose memory budget was not recorded and therefore cannot be reconstructed, and additionally the same causal-vs-full computation at the main experiments' configuration on two physical A800 cards (one worker each, 0.60) matching the main experiments' request count, corpus, and skew. Workload: Zipf(α=1.0) document access for the main and quality evaluations, with a Zipf(α=1.5) robustness condition; 500-document corpus for the main experiments, 200 documents for the NQ and causal analyses; 512-token mean document length; 3–5 documents per request. Code is planned for release at camera-ready, comprising the vLLM cache-coherent ordering integration and the multi-tenant workload generator.

Scope of the causality and window analyses (Section 4.7, Appendix D.1). These run on a single shared NVIDIA RTX PRO 6000 Blackwell card, 2 workers, `gpu_memory_utilization=0.45`, 200-document corpora, and 350-request traces with a held-out final-175 eval set. The causality family covers Zipf($\alpha = 1.5$) and 1.0, and the window runs use 3 seeds. A matching-configuration computation on the main experiments' own configuration (two physical A800 cards, one worker per card, 1000 requests, 500 documents, Zipf $\alpha = 1.0$, 5 seeds) returns the same system retention, 99.82%, so the retention is a property of corpus rank-order recovery rather than of card-sharing. We make no claim about how retention would differ across card topologies: as Appendix E records, that run's per-seed values coincide exactly with the single-card computation, so it is the same computation at a different configuration, not an independent replication. The earlier leakage runs were executed on a single assigned GPU with no memory-utilization setting recorded, so that configuration cannot be reconstructed; the staleness runs used two physical A800 cards at 0.60 (5 seeds). The Blackwell analyses therefore differ from the main experiments on scale, corpus, and hardware, so any retention figure from them is scoped to that configuration; the A800 check is a separate configuration and its own result, not pooled with it.

ETHICS STATEMENT

CacheCoherent does not modify model weights, training data, or the set of retrieved documents. However, document reordering changes the input context order and can alter model outputs: extractive QA token F1 decreases by up to 32.6 pp under full frequency ordering (Section 4.6). Practitioners should evaluate quality impact for their specific downstream task before deploying frequency-aligned ordering. Security considerations for multi-tenant cache sharing are discussed in Appendix A.

USE OF AI ASSISTANTS

AI assistants were used for the following tasks under author supervision; every AI-assisted output was reviewed, verified, and edited by the authors, who take full responsibility for the content of this paper. *Experiment design and methodology*: proposing and refining the evaluation matrix (baselines, ablations, workload scenarios, and the causality, staleness, and window controls), the experimental parameters, and the protocol comparisons, which the authors reviewed and approved before execution. *Method implementation and code*: writing and editing the ordering, routing, vLLM-integration, workload-generation, and metrics-processing code, including the per-seed aggregation used in the

tables and figures. *Synthetic data and data preparation*: constructing the synthetic multi-tenant RAG workload traces and cleaning and reformatting the retrieved-document and evaluation sets. *Formalization and proof assistance*: proposing and refining the mathematical formulation and the hypotheses, and restructuring the exchange argument in Section 3.2; the authors verified the final statements, their assumptions, and their scope. *Result interpretation*: analyzing and interpreting the experimental results and drafting the corresponding discussion. *Figures*: creating the data figures from the authors' experimental data. *Writing and literature*: drafting and editing prose, suggesting structure and keywords, searching for and summarizing related literature, and preparing and formatting bibliographic entries, which were cross-checked against venue records where available and against the cited preprint for entries without a published venue. AI assistants were *not* used to generate or alter experimental results—these were produced by the described system and independently verified against the raw experiment records—*not* used to translate any material, and no qualitative or thematic coding study was performed for this work. All claims and numbers were checked against the underlying artifacts; the authors bear full responsibility for the final content.

REFERENCES

Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. Cache-Craft: Managing chunk-caches for efficient retrieval-augmented generation. *Proceedings of the ACM on Management of Data*, 2025. doi: 10.1145/3725273. URL https://doi.org/10.1145/3725273.

Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

Kexin Chu, Zecheng Lin, Dawei Xiang, Zixu Shen, Jianchang Su, Cheng Chu, Yiwei Yang, Wenhui Zhang, Wenfei Wu, and Wei Zhang. Selective KV-cache sharing to mitigate timing side-channels in LLM inference. *arXiv preprint arXiv:2508.08438*, 2025. URL https://arxiv.org/abs/2508.08438.

Gregory Dexter, Shao Tang, Ata Fatahi Baarzi, Qingquan Song, Tejas Dharamsi, and Aman Gupta. LLM query scheduling with prefix reuse and latency constraints. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2502.04677.

Chuheng Du, Junyi Chen, Hanlin Tang, Kan Liu, Tao Lan, Lin Qu, Chaoyue Niu, Shengzhong Liu, Guihai Chen, and Fan Wu. C2KV: Compressed and composable KV cache reuse for efficient LLM inference. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 991–1002, 2026. doi: 10.1145/3770855.3817715.

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 (MLSys)*, 2024. URL https://arxiv.org/abs/2311.04934.

Cheng-Yu Hsieh, Yung-Sung Chuang, Chun-Liang Li, Zifeng Wang, Long T. Le, Abhishek Kumar, James Glass, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Found in the middle: Calibrating positional attention bias improves long context utilization. In *Findings of the Association for Computational Linguistics (ACL)*, pp. 14982–14995, 2024. URL https://arxiv.org/abs/2406.16008.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2410.15332.

Wenqi Jiang, Suvinay Subramanian, Cat Graves, Gustavo Alonso, Amir Yazdanbakhsh, and Vidushi Dadu. RAGO: Systematic performance optimization for retrieval-augmented generation serving. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2025a. doi: 10.1145/3695053.3731093.

Wenqi Jiang, Shuai Zhang, Boran Han, Jie Wang, Bernie Wang, and Tim Kraska. PipeRAG: Fast retrieval-augmented generation via adaptive pipeline parallelism. In *Proceedings of the ACM*

SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2025b. doi: 10.1145/3690624.3709194.

Yinsicheng Jiang, Yeqi Huang, Liang Cheng, Cheng Deng, Xuan Sun, and Luo Mai. ContextPilot: Fast long-context inference via context reuse. In *Proceedings of Machine Learning and Systems (MLSys)*, 2026. URL https://arxiv.org/abs/2511.03475.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. RAGCache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 2026. doi: 10.1145/3768628.

David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1997. doi: 10.1145/258533.258660.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 2019. doi: 10.1162/tacl_a_00276.

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 ACM Symposium on Operating Systems Principles (SOSP)*, 2023. doi: 10.1145/3600006.3613165.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Yuhang Li, Rong Gu, Chengying Huan, Zhibin Wang, Renjie Yao, Chen Tian, and Guihai Chen. HotPrefix: Hotness-aware prefix KV cache management for shared-prefix LLM inference serving. *Proceedings of the ACM on Management of Data*, 2025. doi: 10.1145/3749168.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024. URL https://arxiv.org/abs/2405.19888.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 2024a. URL https://arxiv.org/abs/2307.03172.

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. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM Conference*, 2024b. URL https://arxiv.org/abs/2310.07240.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. LMCache: An efficient KV cache layer for enterprise-scale LLM inference. In *Proceedings of Machine Learning and Systems (MLSys)*, 2026. URL https://arxiv.org/abs/2510.09665.

llm-d contributors. llm-d: Kubernetes-native distributed LLM inference. https://github.com/llm-d/llm-d, 2025. Kubernetes-native distributed LLM inference project (CNCF Sandbox; founded by Red Hat, Google Cloud, IBM Research, CoreWeave, and NVIDIA); open-source system, not indexed in academic venues.

Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001. doi: 10.1109/71.963420.

NVIDIA. NVIDIA Dynamo: A datacenter-scale distributed inference serving framework. https://github.com/ai-dynamo/dynamo, 2025. NVIDIA open-source datacenter-scale inference serving framework (Apache-2.0); industry system, not indexed in academic venues.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2024. URL https://arxiv.org/abs/2311.18677.

Alexander Peysakhovich and Adam Lerer. Attention sorting combats recency bias in long context language models. *arXiv preprint arXiv:2310.01427*, 2023. URL https://arxiv.org/abs/2310.01427.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pp. 155–170, 2025. URL https://arxiv.org/abs/2407.00079.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-LoRA: Serving thousands of concurrent LoRA adapters. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024. URL https://arxiv.org/abs/2311.03285.

Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. doi: 10.1145/2786.2793.

Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. doi: 10.1002/nav.3800030106.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for LLM serving. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2407.00023.

He Sun, Shinan Liu, Siyuan Ma, Junhao Li, Mingjun Xiao, and Wenhao Jiang. Agent-assisted side-channel attacks on non-prefix KV cache in RAG. *arXiv preprint arXiv:2606.21842*, 2026. URL https://arxiv.org/abs/2606.21842.

Kaizhen Tan, Rong Gu, and Mingyuan Li. CacheWeaver: Cache-aware evidence ordering for efficient grounded RAG inference. *arXiv preprint arXiv:2606.19667*, 2026. URL https://arxiv.org/abs/2606.19667.

David G. Thaler and Chinya V. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Transactions on Networking*, 6(1):1–14, 1998. doi: 10.1109/90.663936.

Jiahao Wang, Weiyu Xie, Mingxing Zhang, Boxin Zhang, Jianwei Dong, Yuening Zhu, Chen Lin, Jingqi Tang, Yaochen Han, Zhiyuan Ai, Xianglin Chen, Yongwei Wu, and Congfeng Jiang. From prefix cache to fusion RAG cache: Accelerating LLM inference in retrieval-augmented generation. *Proceedings of the ACM on Management of Data*, 2026a. doi: 10.1145/3786655.

Wenfeng Wang, Xiaofeng Hou, Peng Tang, Hengyi Zhou, Jing Wang, Xinkai Wang, Chao Li, and Minyi Guo. PCR: A prefetch-enhanced cache reuse system for low-latency RAG serving. *arXiv preprint arXiv:2603.23049*, 2026b. URL https://arxiv.org/abs/2603.23049.

Guanlong Wu, Zheng Zhang, Jianyu Niu, Weili Wang, Yao Zhang, Ye Wu, and Yinqian Zhang. I know what you asked: Prompt leakage via KV-cache sharing in multi-tenant LLM serving. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2025.

XiangFan Wu, Lingyun Ying, Guoqiang Chen, Yacong Gu, and Haipeng Qu. Cache me, catch you: Cache related security threats in LLM serving frameworks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2026. URL https://www.ndss-symposium.org/ndss-paper/.

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 European Conference on Computer Systems (EuroSys)*, 2025. URL https://arxiv.org/abs/2405.16444.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-based generative models. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 521–538, 2022.

Tan Yu, Anbang Xu, and Rama Akkiraju. In defense of RAG in the era of long-context language models. *arXiv preprint arXiv:2409.01666*, 2024. URL https://arxiv.org/abs/2409.01666.

Ying Yuan, Pengfei Zuo, Bo Wang, Zhangyu Chen, Zhipeng Tan, and Zhou Yu. DualMap: Enabling both cache affinity and load balancing for distributed LLM serving. *arXiv preprint arXiv:2602.06502*, 2026. URL https://arxiv.org/abs/2602.06502.

Yichi Zhang, Zhiqi Wang, Huan Zhang, and Yuchen Yang. HijackKV: New threat in position-independent KV cache reuse. In *Proceedings of the USENIX Security Symposium*, 2026. URL https://arxiv.org/abs/2607.19957.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024.

Yilong Zhao, Shuo Yang, Kan Zhu, Lianmin Zheng, Baris Kasikci, Yifan Qiao, Yang Zhou, Jiarong Xing, and Ion Stoica. BlendServe: Optimizing offline inference with resource-aware batching. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 255–273, 2026. doi: 10.1145/3779212.3790133. URL https://doi.org/10.1145/3779212.3790133.

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 (NeurIPS)*, 2024a. URL https://arxiv.org/abs/2312.07104.

Zhen Zheng, Xin Ji, Taosong Fang, Fanghao Zhou, Chuanjie Liu, and Gang Peng. BatchLLM: Optimizing large batched LLM inference with global prefix sharing and throughput-oriented token batching. *arXiv preprint arXiv:2412.03594*, 2024b. URL https://arxiv.org/abs/2412.03594.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024. URL https://arxiv.org/abs/2401.09670.

A Security and Privacy Considerations

Threat model. Multi-tenant KV cache sharing exposes three attack surfaces relevant to CacheCoherent: (1) *predictable cache patterns*: deterministic frequency-based ordering makes cache contents predictable to an adversary who knows the frequency table, enabling targeted cache probing; (2) *timing side channels*: TTFT differences between cache hits and misses allow reconstruction of whether a specific prefix was cached, leaking information about other tenants' queries Wu et al. (2025); (3) *cross-tenant shared state*: shared prefix caches across tenants enable content extraction attacks on cached KV entries Sun et al. (2026).

Known attacks. HijackKV Zhang et al. (2026) achieves 94% attack success on position-independent cache reuse, PROMPTPEEK Wu et al. (2025) demonstrates 99% prompt reconstruction via TTFT timing, SpliceLeak Sun et al. (2026) extracts cached content in RAG settings, and Wu et al. Wu et al. (2026) catalog six attack vectors. SafeKV Chu et al. (2025) proposes selective cache sharing.

Proposed mitigations (unverified). CacheCoherent's deterministic ordering exacerbates the predictability surface relative to random or adaptive orderings. We identify two mechanism-level mitigations: (1) noise injection into the frequency table (adding random perturbations to document frequencies to reduce ordering predictability) and (2) randomized prefix padding (inserting random padding tokens between the system prompt and the first document to prevent deterministic prefix alignment). **These mitigations are proposed as architectural directions only; we have not implemented or experimentally validated either.** Security is not an empirical contribution of this work. Deployers should conduct independent threat assessment before enabling cross-tenant prefix sharing.

B Convergence Trajectory

Figure 4 plots the *cumulative* hit rate over the trace for both orderings, given a precomputed frequency table. It is a cumulative measure and should be read as such: it shows that the frequency-aligned curve stays above the cache-dependent curve at every point of the trace, not that either ordering reaches its asymptote at a particular request count. The measured quantity that separates them is the cross-request-count gap, which narrows to +0.4 pp at 2000 requests as the adaptive method accumulates sufficient history, so the advantage is transient rather than a steady-state level difference.

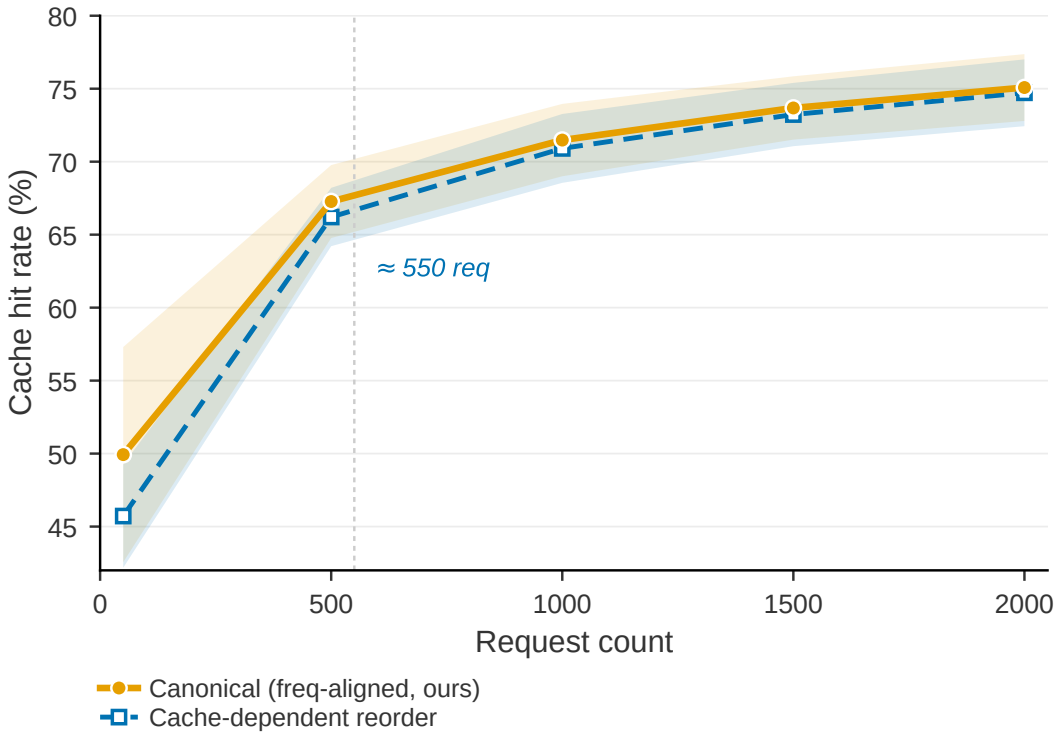


Figure 4: Cumulative hit rate over the trace for frequency-aligned (Canonical, orange) and cache-dependent (blue) ordering, both given a precomputed frequency table. The frequency-aligned curve remains above the cache-dependent curve throughout; because both are cumulative running averages, this figure does not show the instantaneous rate at any request count. Shading: ±1 SD over 3 seeds (2000 req).

C Additional Details

Per-seed detail for the workload table. The real-world-calibrated rows of Table 2 come from the real-workload run (300 req $\times$ 3 seeds, seeds 42/123/456, 3 conditions). Per-seed HR — baseline / canonical-WPA — is 0.2293/0.3458, 0.2225/0.3313, 0.1810/0.3761 (search engine); 0.4498/0.5540, 0.2574/0.5365, 0.4270/0.5518 (customer support); 0.2795/0.3557, 0.1952/0.4202, 0.2642/0.3624 (legal research). Paired per-seed deltas (WPA minus baseline, pp) are $+11.7/+10.9/+19.5$ (search engine, $t(2) = 5.08$, $p = 0.037$), $+10.4/+27.9/+12.5$ (customer support, $t(2) = 3.07$, $p = 0.092$), and $+7.6/+22.5/+9.8$ (legal research, $t(2) = 2.87$, $p = 0.10$). The three deltas are positive on every seed in all three scenarios; an exact two-sided sign test at $n = 3$ cannot go below $p = 0.25$, so none of the per-scenario results reaches significance by a non-parametric route, and the customer-support baseline is the least stable of the six conditions (sample SD 10.5 pp). CacheCoherent's sample SDs are 2.3/1.0/3.5 pp respectively, so the method's across-seed spread is smaller than the baseline's in every scenario. The synthetic rows come from the workload-characterization run (same request count and seeds; only means and across-seed SDs were retained, and those use the population convention, so no sample SD is quoted for them above).

Real-workload validation detail. The real-workload validation (Section 4.5) runs on the full 313-question NQ set with real documents, 10 tenants, and a 200-document corpus, and both conditions are evaluated on the same requests. Per-seed HR is 6.35/6.71/6.13% (baseline) and 19.70/21.25/20.52% (canonical-WPA) for seeds 42/123/456. On corpora with limited cross-tenant overlap, absolute hit rates are correspondingly lower: the method's value scales with available prefix sharing, and the $+14.1$ pp figure is specific to this corpus and tenant mix rather than a universal gain.

D Per-Seed Data and Additional Tables

D.1 Per-Seed Frequency Leakage

The frequency-leakage analysis (full-trace vs. past-only frequency table, $n = 5$ seeds) shows a per-seed leakage of $+6.37$, $+5.13$, $+5.38$, $+3.65$, $+4.38$ pp, mean $+4.98$ pp (paired $t = 10.81$, $p < 0.001$), with the past-only table retaining 76% of the ordering gain.

D.2 Per-Seed Non-Parametric Detail

Table 5 gives the per-seed ordering gain (canonical-WPA minus no-reorder-RR) for each family, with non-parametric summaries. The mixed 15-seed pool comprises the 1000-request family (seeds 42–1024) and the 500-request family (seeds 2024–2033); an exact two-sided permutation test on the stratum means does not separate them (difference -1.36 pp, 3003 permutations, $p = 0.72$), so pooling is defensible. Bootstrap percentile intervals are reported for completeness but are cited nowhere in the body: at $n = 5$ they are descriptive only and are not independent inferential support, and the $n = 5$ intervals can even be narrower than the corresponding t-intervals, a small-sample artefact rather than precision. The primary uncertainty statements are the t-intervals and the exact sign tests; only the pooled 15-seed cohort clears a non-parametric test. Two provenance caveats: the C-group's $n = 5$ is not a clean single-run cohort (seed 1024 was re-run after an initial failure, same output target), and the three causal batches all reuse batch 1's shared `no_reorder` baseline, so their deltas share a denominator — the gains are valid but those rows are not fully independent draws. Absolute gains are skew-dependent ($\approx$20.1 vs. $\approx$26.9 pp at $\alpha = 1.0$ vs. 1.5), so the causal restriction is essentially free at both skews.

D.3 C-Group Router Advantage, Per Seed

Table 7 gives the per-seed support for the C-group range and mean quoted in Section 4.2 and the abstract.

Router-matrix instrumentation caveat. The locality-derived figures reported in Section 4.2 (dominant-worker occurrence share, cold-worker reuse) come from a document-to-worker placement proxy, not an independent per-request cache-hit measurement: per-request locality instrumentation was not available on this matrix, so per-request locality is a proxy here, and the HR values

Table 5: Per-seed ordering gain (pp), canonical-WPA minus no-reorder-RR, with non-parametric summaries. Sign test is exact two-sided binomial against zero; d_z is Cohen's paired effect size.

Family	n	Mean (pp)	95% t-CI	Sign p	d_z
1000-req cohort	5	$+23.08$	$[12.10, 34.06]$	0.0625	2.61
500-req stratum (10 seeds)	10	$+24.44$	$[20.25, 28.64]$	0.0020	4.17
C-group (350 req)	5	$+25.97$	$[13.36, 38.59]$	0.0625	2.56
Leakage oracle	5	$+20.73$	$[8.37, 33.09]$	0.0625	2.08
Leakage past-only	5	$+15.74$	$[3.74, 27.75]$	0.0625	1.63
Causal α=1.0 (system)	5	$+20.12$	$[13.71, 26.54]$	0.0625	3.89
Pooled 15-seed	15	$+23.99$	$[20.28, 27.70]$	0.000061	3.58

Table 6: Per-seed values underlying each family in Table 5 (pp gains, one column per seed).

Family	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10
1000-req cohort	36.58	13.45	17.22	24.91	23.23	—	—	—	—	—
500-req stratum (10 seeds)	27.63	20.66	18.26	20.66	36.47	30.15	27.89	19.01	21.47	22.21
C-group (350 req)	41.01	13.75	20.44	28.61	26.06	—	—	—	—	—
Leakage oracle	35.69	9.29	14.86	23.41	20.38	—	—	—	—	—
Leakage past-only	29.32	4.16	9.48	19.76	16.00	—	—	—	—	—
Causal α=1.0 (system)	27.28	13.20	18.68	19.17	22.28	—	—	—	—	—

themselves are the Prometheus deltas. Figure 2(b) draws its HR bars and its WPA and consistent-hash gini annotations from the C-group cohort, but the power-of-two, rendezvous, and DualMap gini annotations shown there are the 300-request matrix's values ($0.1007/0.0354/0.0280$) rather than this cohort's; no claim depends on those three annotations.

Content-hash cohort non-conflation. Two cohorts bound the content-hash null and must not be conflated: the 5-seed ablation gives $\Delta = +1.7$ pp ($t(4) = 0.19$, $p = 0.86$), and the dedicated 15-seed tightening cohort gives -0.96 pp with 95% CI $[-7.06, +5.14]$. The signs differ ($+1.7$ vs. -0.96), so averaging them would be meaningless; the body reports each separately.

D.4 Adaptive-Baseline Detail (Per-Seed)

Two paired runs differ only in the per-request decision budget (350 requests, 3 seeds, 2 workers on one card, `gpu_memory_utilization=0.45`, zero lookahead, identical WPA router and frozen train-window frequency table; 60/20/20 split).

E Causality Control Detail

Re-estimating the frequency table from a strictly causal (past-only) view delivers $+15.7$ pp over no-reorder ($n = 5$ seeds) against the $+20.7$ pp full-trace oracle, retaining 76% of the ordering gain. At the paper's operating skew a strictly causal table at $\alpha = 1.0$ retains 99.37% of the full-table gain at the system level and 99.67% at the ordering level (vs. 98.98% and 100.18% at $\alpha = 1.5$), both at 350 requests. All retention fractions use one convention, the *ratio of mean gains* (mean causal gain / mean full-table gain, each against the same baseline); the per-seed-mean alternative is a different quantity and is not cited as a headline. The 76% and $\approx$99% figures are not nested: 76% is the cold-start cost of a streaming table against the full-trace oracle, whereas the $\approx$99% layer values above are a frozen past-only table versus the full-trace table on the same held-out window. Those 350-request ratios leave scale, corpus, and hardware unmatched to Table 1; the same causal-vs-full contrast computed at that table's own configuration (two physical A800 cards, one worker per card, 1000 req, 500 docs, $\alpha = 1.0$, 5 seeds) returns an identical system retention of 99.82%. We make no claim about how the two card topologies would differ: see the per-seed identity noted below, which shows this is the same computation at a different configuration rather than an independent replication. Table 9 gives the per-seed dispersion behind these aggregates. Ordering adds 0.069 ms at p99; WPA is $O(kW)$.

The two 1000-request cohorts' per-seed retention vectors are identical to displayed precision (delta 0.0000 pp). We report this as measured rather than smoothing it, and we read it as a *weaker*, not stronger, form of evidence: bit-identical per-seed values across two hardware configurations mean

Table 7: WPA minus round-robin HR (pp), C-group cohort (350 requests, one shared card, canonical ordering fixed), system level. The body's $+2.9$–4.1 pp range and $+3.5$ pp mean are computed over these five seeds (min 2.88, max 4.09, mean 3.468). Provenance: for seeds 42/456/789 the value is each run's recorded WPA-minus-round-robin delta; for seed 1024 it is the seed-1024 routing-baseline run; **for seed 123 it is derived from the run's per-condition hit rates** ($0.628836 - 0.589790$) because that run does not record a precomputed delta — its component hit rates are recorded, so the value is derivable rather than missing.

	s42	s123	s456	s789	s1024
WPA − RR (pp)	2.88	3.90	4.09	3.45	3.02

Table 8: Adaptive versus static frequency ordering, per seed (test-window HR). The budgeted adaptive arm misses the 1.0 ms budget on 90% of decisions.

	Static `freq+WPA`		Adaptive `+WPA`		
Seed	HR	ord. (μs)	HR	ord. (μs)	budget
42	0.6645	152.3	0.6401	1316.2	off
123	0.6380	46.1	0.6645	2549.5	off
456	0.6762	152.5	0.6422	3508.0	off
Mean	0.6596	116.9	0.6489	2457.9	off

the second run is not an independent replication, so the matched-configuration causal result rests on a single computation rather than two. Identity is consistent with the mechanism — each worker maintains its own KV cache, and worker assignment and document ordering both derive from the same seed-deterministic tables, so the cache-hit pattern does not depend on how requests are distributed across cards — but we do not make any claim about how retention would behave on card topologies we did not independently measure. The frequency table nevertheless drives both mechanisms: the matched-configuration control varies ordering and assignment together rather than isolating either.

F Restart-Recovery Measurement

The main results fit the frequency table on the full trace; the causality control (Appendix E) re-estimates it from past-only statistics. Both leave open whether the ordering advantage survives an *actual* cache-loss event. We therefore ran a direct mid-trace restart experiment: a warmup phase to a steady state, a forced eviction, then a recovery phase, across three ordering conditions and three seeds (42/123/456), 9/9 cells complete. The registered falsifier was that cache-dependent ordering would recover no slower than frequency-aligned ordering; it returned *against* that hypothesis at 3/3 seeds under both of two independent metrics.

The companion metric fixes a common threshold and asks when each arm's hit rate crosses it. At $F = 0.55$, cache-dependent ordering reaches the crossing later than frequency-aligned ordering by $+50/+60/+40$ requests (seeds 42/123/456). The registered relative metric instead gives each arm its own threshold ($0.9\times$ its pre-eviction hit rate) and asks when it reaches 90% recovery; the same direction holds, $+40/+40/+30$ requests. We report per-seed values only — a mean of three crossing times would present three integers as a continuous quantity — and every crossing travels with its margin (a bare crossing request is not a measurement: the checkpoint grid is $\sim$10 requests).

Three qualifications bound this result. (1) *Different topology.* It ran on one RTX PRO 6000 card with 2 workers, whereas the throughput sweeps use two A800 cards; it must not be read as the same configuration, and its absolutes are not transferable. (2) *Thin margins.* The thinnest crossing anywhere is 0.000054 (five parts in a hundred thousand, cache-dependent ordering, seed 123, at the fixed companion threshold $F = 0.60$); near-threshold crossings are not robust to baseline variance, so the crossing request is always reported with its margin. (3) *Discriminating power and a boundary.* Seed 123's baseline plateau sits only 0.052 below the companion threshold; and the companion discriminates at $F = 0.55/0.60/0.65$ but not at $F = 0.50$, where the baseline reaches the threshold at seed 456, so that seed is excluded at that setting. The three repetitions within a cell are byte-identical, which shows determinism rather than statistical power; all inference runs on the 3-seed axis. We do not cite the eviction guard's pass verdict as evidence the cache actually went cold:

Table 9: Per-seed causal retention (% of the full-table gain, ratio of mean gains). These are dispersion, not headline figures: the aggregates above are the reported quantities, and the per-seed *mean* is a different quantity that must not be cited as either. The first two rows are the matched pair the identity comparison concerns (both 1000 req, 500 docs, α=1.0, system level, differing only in card topology). The third row is a *different* cohort — 350 requests on one shared card — included for completeness; it is not the comparison the identity result concerns.

Cohort	s42	s123	s456	s789	s1024
Same-topology, 2-card, 1000 req (system)	98.14	106.93	99.63	96.50	98.68
Single-card, 1000 req (system)	98.14	106.93	99.63	96.50	98.68
350-req single card (different cohort)	97.52	98.95	99.73	100.47	100.69

a worker restart zeroes the Prometheus counters, so that check passes for any restart. The real guarantee is the conjunction of a tracked-PID process-group kill, verified GPU-memory cleanup, and verified ports-free. No stream disconnected during this run, so the harness's per-attempt reconnect path never fired: the per-attempt TTFT-clock fix that path carries remains verified by code inspection only, and the clean TTFT figures here are not evidence that it was exercised.

What recovery saves in latency. The crossings above convert to an elapsed-latency result. Over the 500-request recovery window, frequency-aligned ordering saves the *cumulative recovery TTFT* shown below, per seed, against the unordered baseline. We report it as a range rather than a headline figure, because the saving is seed-dependent: $\approx$4.8–17.7 s of aggregate TTFT per recovery window ($\approx$17,669 ms at seed 42, $\approx$4,787 ms at seed 123, $\approx$7,242 ms at seed 456). The robust comparison is the tail: p95 TTFT falls by 8.3/8.3/19.8 ms (canonical vs. baseline, seeds 42/123/456; $\approx$8–37 ms across the p95/p99 family), with no reversal in any repetition at any seed. The contrast against *cache-dependent* ordering is much thinner and must be read with its margin, per the discipline above. Table 10 gives the per-seed cumulative and per-request savings over the 500-request recovery window; the range is 1.1–3.9 ms/req, and it carries one repetition-level reversal at seed 456 (-80 ms).

Table 10: Cumulative TTFT saving of canonical over cache-dependent ordering, per seed, over the 500-request recovery window.

Seed	Cumul. saving (ms)	Per-request (ms/req)
42	1,971	3.94
123	1,515	3.03
456	554	1.11

Two properties bound the reading. First, the quantity is *cumulative TTFT saving*, not prefill time: the instrumentation exposes no prefill/decode split and no queue-time field, so no such attribution is available. Second, the magnitude is driven by the *unordered baseline's* own cross-seed draw — seed 42's baseline sits far above its levels at the other seeds while its three repetitions are tight among themselves — so this is the third quantity, after hit rate and throughput, in which the unordered baseline carries the cross-seed spread, and these savings must never be pooled across seeds. The latency figures come from a two-worker run on the same host and protocol as the experiment above, which reproduces it exactly (18/18 comparable crossings, all three baselines, and a cell matching counter-to-counter); that makes it a configuration-identity check, not a replication.

G Throughput Under Concurrent Load

Table 12 shows frequency ordering provides $\sim$1.2–1.6$\times$ throughput over no-reorder across concurrency 1–8, and WPA delivers slightly *higher* throughput than frequency+round-robin at concurrency $\geq$ 2 (+4 to +14%). Those rows are measured on a single RTX PRO 6000 Blackwell card with 2 workers sharing it; WPA's load imbalance (gini 0.28) gives no throughput penalty on this shared card (its p99 TTFT is 0.92–0.99$\times$ of frequency+RR). Under per-card isolation on two physical A800 GPUs (`cuda:0,1`), by contrast, WPA's imbalance concentrates load on one card and idles the other, and the penalty is now measured at five seeds (Table 12, dual-card rows; mean $\pm$ sample

Table 11: Restart recovery (one RTX PRO 6000 card, 2 workers, 3 seeds; a different configuration from the two-card throughput sweeps). Positive values mean cache-dependent ordering reached the recovery point later than frequency-aligned ordering, i.e. frequency-aligned ordering recovered sooner. Companion metric: requests to cross a fixed threshold $F = 0.55$. Relative metric: requests to reach 90% of each arm's own pre-eviction hit rate.

Seed	Companion (F=0.55)	Relative (90% of own pre-HR)	Margin at crossing
42	+50	+40	+0.00248 / +0.00723
123	+60	+40	+0.00276 / +0.00091
456	+40	+30	+0.00395 / +0.00351

SD, ddof $= 1$). Pooling tightens rather than weakens the finding: the five-seed means are *more* negative than the original single-seed values. Two consequences are worth stating. First, seed 42 is a low baseline draw — pooling raises the baseline column by $+3.4$ to $+4.1$ rps while the ordered arms move by at most 0.8 rps — so a reader comparing the new baseline column against the old single-seed row would otherwise see an unexplained jump. Second, and as a finding rather than a silent correction, the published single-seed dual-card ratios (1.67–1.83) sat *above* the single-card band of 1.2–1.6, whereas the pooled five-seed ratios (1.40–1.49) sit *inside* it: after the fold the two topologies agree.

Table 12: Throughput (req/s) at concurrency 1–8. Single-card rows: 3 seeds (42/123/456, 300 req/condition), 2 workers sharing one RTX PRO 6000 Blackwell card; the $\pm$ spread is that of the three-seed aggregate; its SD convention is not recorded, so it is reported as an unspecified spread rather than asserted as a sample or population SD. Dual-card rows: $n = 5$ seeds (42/123/456/789/1024) pooled across two protocol-identical runs (the combine was gated on a per-key config diff, which passed), two physical A800 GPUs with per-card isolation, `gpu_memory_utilization=0.60`; the $\pm$ is the sample SD (ddof $= 1$) over the five seeds, matching the convention of the rest of the paper.

Concurrency	No-reorder	Freq+RR	Freq+WPA	
1	9.9 ± 1.5	11.9 ± 0.5	12.5 ± 0.3	
2	13.6 ± 2.7	18.1 ± 0.5	19.7 ± 0.2	
4	15.1 ± 3.4	21.4 ± 0.3	23.9 ± 0.4	
8	15.8 ± 3.7	22.7 ± 0.3	25.8 ± 0.7	
Dual-card, per-card isolation ($n = 5$ mean $\pm$ sample SD), WPA penalty vs. Freq+RR				
2	16.6 ± 2.3	22.9 ± 0.6	18.2 ± 0.4	(-20.5%, range -22.6 to -19.3)
4	18.3 ± 2.8	26.2 ± 1.0	19.5 ± 0.5	(-25.4%, range -27.8 to -22.9)
8	18.5 ± 3.0	26.9 ± 0.9	19.5 ± 0.5	(-27.2%, range -30.6 to -24.8)

Per-seed values for the dual-card rows. Baseline (no-reorder) at c2/c4/c8: $13.2/14.2/14.1$ (s42), $19.6/21.9/22.4$ (s123), $17.1/19.2/19.4$ (s456), $16.2/17.8/18.0$ (s789), $16.8/18.4/18.4$ (s1024). Freq+WPA: $17.7/19.5/19.5$ (s42), $18.3/19.7/19.7$ (s123), $18.2/19.2/19.3$ (s456), $17.9/19.0/18.9$ (s789), $18.8/20.3/20.3$ (s1024). The WPA penalty is negative at every seed and every level (5/5), consistent with the closed per-card-isolation result rather than a reinterpretation of it.

H Prefix-Sorting Discrimination Test

Related work delimits our intra-prompt frequency ordering from *inter-request* prefix sorting (BatchLLM-style). We test that delimitation directly with a four-condition experiment that varies the two levers independently while holding routing fixed at round-robin in all four conditions, so the arm contrast is not confounded by router differences: *baseline* (no sorting), *arm A* (intra-prompt frequency ordering, our mechanism), *arm B* (inter-request prefix sorting, the named prior lever), and *both*. Three seeds (42/123/456), one host and install, so the comparison is internally valid but its absolute numbers are not transferable across hosts. Note that "lever" is used in two distinct senses in this paper: here it denotes the ordering-vs-sorting pair (arm A vs. arm B), whereas the separate $\sim$85%/$\sim$15% decomposition reports an ordering-vs-routing pair at W=2; the two are different comparisons and neither speaks to the other.

Arm B implements only the prefix-sorting component of BatchLLM, which as published also schedules by estimated decode-to-prompt length ratio so decoding overlaps later prefill chunks; our arm does not reproduce that scheduling half, nor BatchLLM's own setting. The null below therefore bounds the *prefix-sorting component* in our setup, and we do not read it as evidence against BatchLLM's system as a whole. This lever pair (intra-prompt ordering vs. inter-request sorting) is distinct from the ordering-vs-routing decomposition reported elsewhere in this paper ($\sim$85%/$\sim$15% at W=2, Section 4.3); a null here leaves that decomposition unaffected.

Arm A is strongly validated, and we report it per seed rather than as a single multiple: the per-seed HR gains are $+36.83$/$+9.66$/$+15.75$ pp — a $3.8\times$ spread, largest exactly where the baseline is weakest (seed 42's baseline HR is the lowest). Read as one aggregate it would misstate the effect.

Arm B is null on three observation channels within a single experiment across all three seeds, with measured decision latency confirming it executed rather than being skipped. The three quantities come from the same three runs and therefore corroborate rather than constitute independent evidence. The hit-rate Δ is -0.224 pp (per-seed -0.52/-0.27/$+0.12$), an *absolute* sample SD (ddof $= 1$) of 0.33 pp — we state the null with its absolute SD because the relative SD of a near-zero mean is numerically meaningless; throughput moves $+0.0097$ rps against arm A's $+3.638$ (SD 0.052 rps on a 7–11 rps base); and mean TTFT moves $+0.098$ ms against arm A's -34.88 ms. The escape hatch for a null is that the arm never ran, so we report the latency control: arm B costs 1.9–3.1 μs versus the baseline's 0.3 μs, a 16–$24\times$ cheaper lever than arm A's 45–53 μs, and its measured cost is a small fraction of the 2.0 ms decision budget (zero budget misses in every cell). The interaction term is measured at $+0.41$ pp on a *3-seed aggregate* — small, and we do not read it as an independent finding — and the both-arms condition is essentially arm A alone ($+20.93$ vs. $+20.75$ pp over baseline). We therefore report one operative lever plus a clean negative on the other, not two complementary mechanisms.

Table 13: Prefix-sorting discrimination test (3 seeds 42/123/456, routing fixed round-robin in all four conditions, one host and install). Deltas are over the no-sorting baseline. SD denotes the sample standard deviation (ddof $= 1$) of the per-seed values.

Condition	HR Δ (pp)	Throughput Δ (rps)	Mean TTFT Δ (ms)
Arm A (intra-prompt freq. ordering)	$+20.75$	$+3.638$	-34.88
Arm B (inter-request sorting)	-0.22 (SD 0.33)	$+0.010$ (SD 0.052)	$+0.098$
Both arms	$+20.93$	$+3.648$	-34.94

I Quality-Cache Pareto Frontier

Table 3 reports the 5-strategy quality-cache ablation underlying Section 4.6; Figure 5 plots the same tradeoff. The published aggregates are verified cell-by-cell against a surviving gold-rescore artifact (5 conditions $\times$ 3 seeds, 124 matched questions per cell, population SD). The gold-answer file (313 NQ short-answer records) survives separately, and the scoring denominator (124/313 matched) is confirmed by exact-integer EM counts. However, the per-question detail files — individual model responses, prompts-as-served, retrieved document sets, and per-question F1/EM scores — are no longer available on any project server, so the frontier cannot be re-derived from per-question inputs. A rebuild with full per-question artifact preservation is designed and staged but requires NQ trace data not currently available in the offline environment. This frontier is therefore a historical measurement whose aggregates are verified but whose per-question basis cannot be audited.

Three properties of the surviving aggregate constrain any future replication: (1) the scoring denominator is exactly 124 questions per cell (confirmed by integer EM counts); (2) the *marked-relevance* condition records identical EM counts across the three seeds (6/124); the count alone does not identify *which* questions were answered, so this does not by itself establish deterministic behavior; (3) refusal-to-answer responses appear in the baseline arm; whether abstention rates differ across conditions, and how any such difference affects the measured delta, cannot be established without per-question data. A future rebuild should preserve per-question artifacts and count abstentions per arm. The three mitigations are defined as follows: *query-adaptive* applies frequency ordering only when the retrieved documents' aggregate corpus-frequency rank exceeds the median; *top-2 relevance + freq* places the two highest-relevance documents at positions 0–1 and orders the rest by

frequency; *marked-relevance* prepends a `[RELEVANT]` marker to the highest-relevance document while maintaining full frequency ordering. F1 is scored against NQ short answers using token-set overlap, $\mathrm{F1}(p, G) = \max_{g \in G} 2|P \cap G_g|/(|P| + |G_g|)$, where P and G_g are the token sets of the predicted and gold answer. Lexical agreement across conditions: canonical+RR vs. canonical+WPA share 96.4% identical responses with ROUGE-L 0.987 ± 0.094 over the 939 pairs, whereas ordered vs. unordered conditions give ROUGE-L 0.157 ± 0.294 — routing is near-transparent at fixed ordering, ordering is not, and neither number measures answer quality. The quality evaluation uses a fixed two-shot prompt: a system instruction, two worked question–answer exemplars (a factual question answered by a short phrase, in the same format as the task), and then the real request; generation is capped at 32 tokens, and F1 is scored against NQ short answers as the set-overlap $\max_g 2|P \cap G_g|/(|P| + |G_g|)$ defined in Section 4.6. These three settings (two-shot prompt, 32-token cap, gold-short-answer F1) determine the 42.5% baseline and are held fixed across all five conditions. ROUGE-L and BLEU-4 against the NQ gold answers are near-zero across all conditions, including the baseline, because both the gold answers and the model responses are very short (the model answers with a short phrase matching the answer-only prompt), leaving little for n-gram overlap metrics to measure.

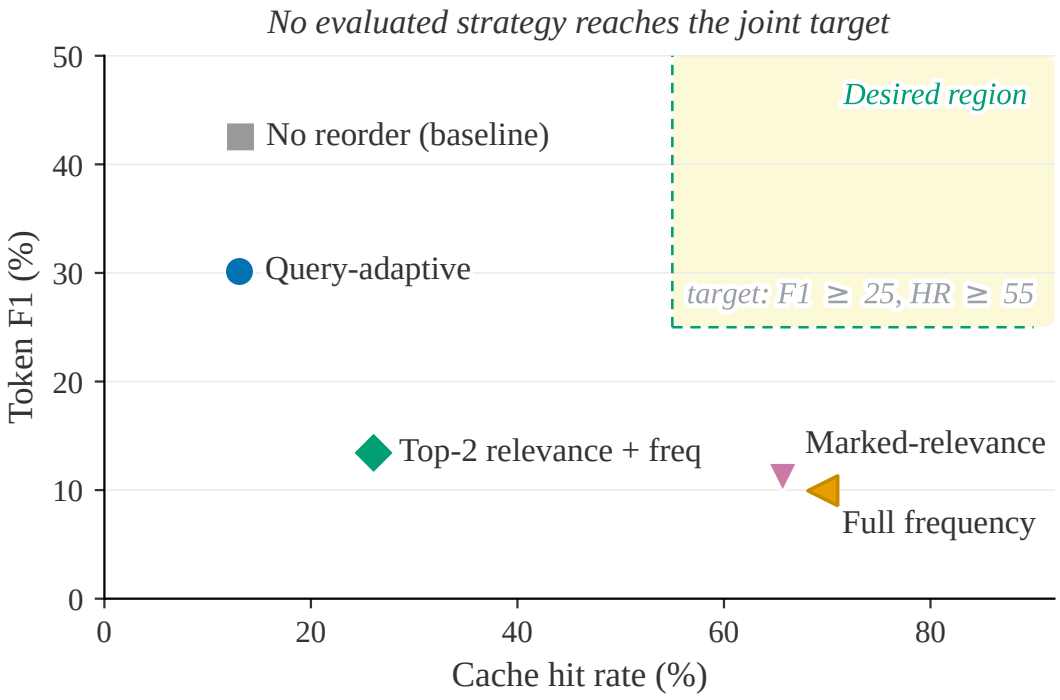


Figure 5: Quality-cache Pareto frontier. Among strategies evaluated on NQ with Qwen2.5-7B-Instruct, none reaches the desired region (F1 $\geq$ 25%, HR $\geq$ 55%). The tradeoff is observed under this model, prompt, and dataset.

J Overlap and Cache Sensitivity

Figure 6 shows the advantage is overlap-invariant (+23.8 to +25.8 pp across 10–80% document overlap, 3 seeds), a confound-free contrast: both arms share the same routing, so the gap is attributable to ordering alone.

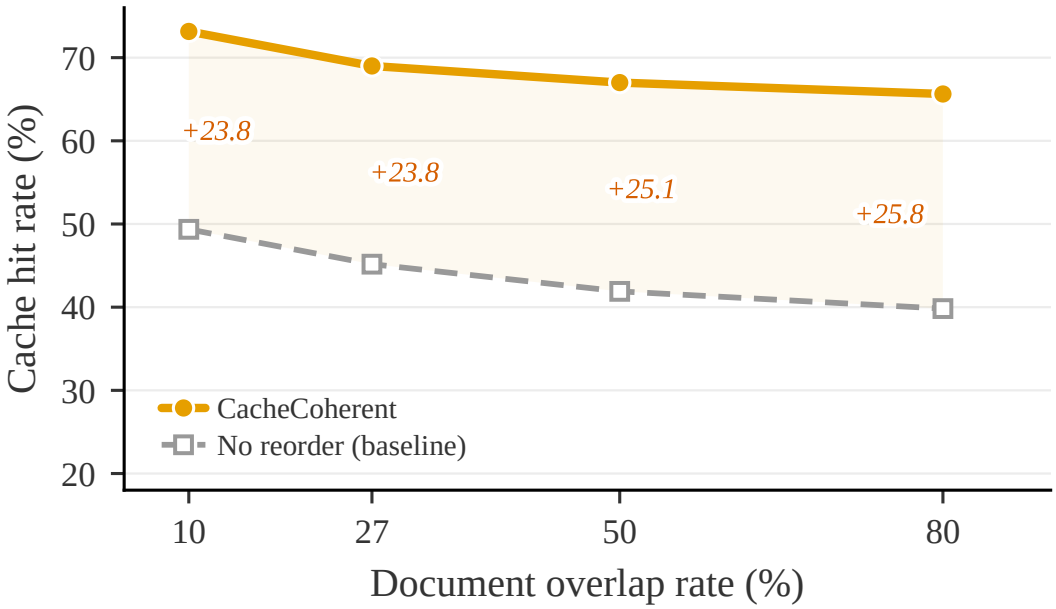


Figure 6: Overlap sensitivity (500 req × 3 seeds). CacheCoherent's advantage is overlap-invariant, +23.8 to +25.8 pp across 10–80% document overlap. The registry stores per-condition means only, so no error bars are shown.