Q-BRIDGE: APPLICATION-GUIDED PHYSICAL KV-CACHE RETENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

An agent often knows which application context it will revisit before the inference server decides which KV blocks to retain. Q-Bridge exposes this decision through a cache-retention interface: a Pi Coding Agent extension sends bounded lifecycle intent to a validating sidecar, which forwards reservations to a version-specific vLLM backend. On Qwen3-4B in FP16 on V100 GPUs, Q-Bridge reduces mean post-bind final-request TTFT from 641.46 to 226.65 ms relative to Cost-only in a constructed 12-task pressure workload. In a separate 160-episode, two-GPU study, its mean final-request TTFT and executed prefill are 30.7% and 46.5% lower than Cost-only on four additional held-out sources. The corresponding workflow mean is 0.65% higher, source-cluster intervals for the unexpanded cohorts include zero, and a same-decision one-bit control reproduces Q-Bridge's prefill. In this study, recorded inputs, outputs, and offline answer scores match across policies within each condition. These results show that application-selected physical retention can reduce local KV recomputation under cache pressure, but do not establish broad end-to-end acceleration or an advantage from richer messages.

1 INTRODUCTION

Agent requests compete for finite KV memory. The application knows which contexts it may revisit; the server sees resident blocks and memory pressure. Existing request interfaces often provide no contract for combining application intent with physical cache state.

We test whether an independent cache controller can use lifecycle intent while leaving application behavior unchanged. Backend acknowledgements establish lease admission, and subsequent backend counters verify physical reuse; matched traces and full workflow timing test whether that reuse offsets communication and control costs.

In Pi Coding Agent 0.85.1, a TypeScript extension binds request identities before provider submission. The application runner selects retention and quotas; an independent Unix-socket sidecar validates and forwards operations, and the vLLM backend admits physical leases (Figure 1).

We hold candidate and verifier outputs, the final branch, and recorded provider payloads fixed across policies. Branch-changing comparisons remain separate diagnostics. Verification and pressure calls are also timed: their cost can outweigh final-request TTFT savings.

Our contributions are:

1. an Agent-to-Infra interface that binds application context identities to physically acknowledged KV-cache leases with explicit ownership and release;
2. an evaluation methodology that separates local reuse from complete workflow cost and audits semantic identity, provider requests, and shared physical blocks; and
3. an empirical characterization of when application-selected retention reduces prefill and final-request TTFT, together with equal-information controls and complete-workflow measurements that delimit this benefit.

The measured benefit is lower prefill work under cache pressure; decode throughput and offline answer quality do not change consistently. We therefore treat one-way lifecycle as an important control for separating the value of the retention decision from the value of the wire protocol.

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.

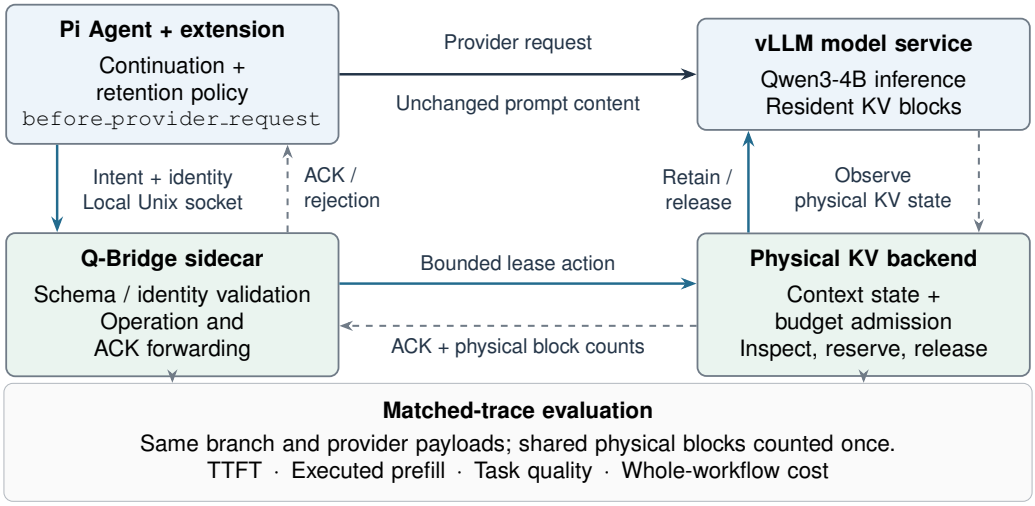

Figure 1: **Q-Bridge request and cache-control paths.** The application selects retention, the sidecar validates and forwards operations, and the backend enforces physical capacity and context identity. Prompt content is unchanged.

Several recent systems connect Agent execution structure to serving-side cache management. KVFlow uses workflow information for prefix-cache reuse (Pan et al., 2025); PBKV uses execution history and prediction for cache guidance (Zheng et al., 2026); and Continuum studies multi-turn scheduling and cache lifetime (Li et al., 2025). TokenCake and Pie provide broader serving abstractions for Agent workloads (Bian et al., 2025; Gim et al., 2025). Q-Bridge focuses on a narrower interface contract: an Agent-originated bounded retention decision is admitted by the physical backend through an independent sidecar and verified against physical vLLM lease effects under matched application traces.

2 System Model

An application identifies a logical context h; after completion, the adapter observes its resident block sequence P_h. The backend maintains a lease set L_h subject to a retained-block capacity C:

$$\left|\bigcup_h L_h\right| \le C, \qquad C = 768 \text{ blocks}. \tag{1}$$

The extension binds a request identity before submission. Completion makes its prefix observable; the runner can then inspect it, request a contiguous quota, collect metrics, and release it. The backend charges shared physical blocks once. An acknowledged reservation does not guarantee a future cache hit. In the measured branch study the runner reserves at most one branch; the global constraint is a safety check, not a tested multi-lease scheduling advantage.

The application owns semantic continuation. The controller cannot choose an answer branch, alter a prompt, or use a benchmark answer. This separation prevents branch choice and answer access from confounding the cache comparison.

2.1 Protocol Semantics

The wire format uses operation-specific JSON fields. Context operations carry `session_id`, `context_id`, and a context `version`. Mutations also carry `event_id`; bind adds `request_id`, while reserve adds `max_blocks`. Metric queries use a request identifier. The sidecar validates the schema and forwards operations to the backend, which maintains context state and enforces reservations. Repeating an event with the same payload returns the previous acknowledgement with

Table 1: Q-Bridge lifecycle operations.

Event	Owner	Required evidence
Bind	Agent extension	controller handle and identity match
Inspect	Adapter	resident physical block count
Reserve	Runner + backend	accepted action and retained count
Metric	Adapter	timestamped KV and timing counters
Release	Extension + backend	release acknowledgement

`duplicate:true`; reusing its identifier with a different payload is rejected. Application branch selection remains in the TypeScript runner.

A successful reserve retains the exact requested quota; otherwise it is rejected without changing the previous reservation. We verify later cache observations before attributing reuse. A zero quota removes pins; release also invalidates idle, unshared cached blocks. Leases require explicit release or reset, with no automatic TTL. Ownership checks enforce handle and session consistency for trusted local clients, not authentication of hostile tenants.

Failure handling depends on the execution path. The optional-intent experiment can abstain from retention when cache advice is malformed. The main tree workflow instead requires a valid application branch selection; a malformed selection raises an error and fails the run. Controller validation failures and rejected physical operations are recorded as errors, not successful leases. The optional-advice fallback therefore does not describe all experiments.

2.2 Division of responsibility

The runner selects retention, the sidecar validates operations, and the backend enforces capacity. Across policies, prompts, application branches, decoding kernels, sampling, and request order remain fixed. Physical acknowledgements and matched traces are both required to attribute a prefill difference to retention.

3 Q-Bridge implementation

3.1 Agent extension

The frozen Pi extension records the active context before each provider request. At the provider boundary it sends a bounded lifecycle frame containing session identity, context identity, version, and request identity. The hook returns the provider payload with the request identity attached. The extension records the bind acknowledgement, first-token timing, provider completion, and metric acknowledgement.

The main experiment uses the application's verifier-selected branch as the retention signal. A one-bit control carries the same selection. The separate optional-intent experiment supports abstention from cache retention; invalid application branch selection in the main workflow instead fails the run.

3.2 Controller and adapter

The sidecar validates message size, operation fields, and acknowledgement identities. The backend tracks pending, completed, and retained contexts and handles duplicate events. A reservation operation changes only a completed resident prefix and returns the exact number of acknowledged retained blocks. Physical errors remain errors; the controller does not report logical success when a reserve call is rejected.

The vLLM adapter reports backend identity, block size, total capacity, current cached blocks, retained blocks, and execution instrumentation. It installs runtime hooks around request registration and block release in the frozen vLLM version. The adapter exposes the backend's cache state to the controller.

3.3 ALLOCATION BASELINES

Cost-only ranks branches by resident block count, treats a one-block difference as a tie, and breaks ties by recency; it is not a calibrated latency model. Workflow-aware first ranks a supplied future-use feature, then applies the same rules. Both features are 1 in the initial workload, so these policies make identical choices there. Transition-aware initially uses configured probabilities 0.6 and 0.4 with alternating branch assignments, not a learned prior. The additional study instead derives priors from disjoint development sources and supplies discriminating workflow features in its public-continuation condition. Stock-prefix makes no reservations; one-way lifecycle retains the live context until release. Comparisons within each experiment share the same budget and arrival trace.

4 EXPERIMENTAL SETUP

The frozen stack uses Pi Coding Agent `@earendil-works/pi-coding-agent` 0.85.1 with commit `71dca871bc80b6bc97be37f0ca3189399d651fff`; Qwen/Qwen3-4B FP16 with a 16K context; vLLM 0.9.0; PyTorch 2.7.0+cu126; Transformers 4.51.3; and a Tesla V100 PCIe 32GB. All policies share synchronous V0 eager execution and a V100 prefix-attention adapter that gathers paged KV for xFormers CUTLASS attention. Its source hash and launch flags are archived. Latency results apply to this adapted stack. The physical pool has 2,048 blocks of 16 tokens; at most 768 blocks may be retained.

A physical PASS records only successful execution and cleanup. Answer quality is measured separately. We report post-bind TTFT from the bind acknowledgement to the first generated token. Where explicitly labeled application TTFT, timing begins at `session.prompt` and includes hook and binding overhead. Episode time runs from root invocation through cleanup on initialized services and a prepared main session; it excludes prior client initialization and model loading.

All assigned requests and abstentions are retained. No gold answer is sent to an online policy. Development, post-hoc repair, and held-out cohorts are kept in separate manifests.

4.1 WORKLOADS

The main workload uses twelve controlled long-context tasks built from real HotpotQA and MATH questions. Source context is repeated three times for HotpotQA and twelve times for MATH. Five 8,192-token pressure requests separate retention from reuse. Each lease is capped at 512 blocks within the global 768-block limit. These are constructed pressure workloads, not unmodified production traces. Each task is replayed under six allocation policies with the same application trace and a shared 768-block budget. The selected condition receives the bounded Agent intent. Blind and wrong conditions deliberately provide unfavorable control signals. Cost-only, Workflow-aware, and Transition-aware are deterministic allocation baselines with the same physical interface. This design makes the comparison a policy test inside one serving path, rather than a comparison of different model servers.

The evaluation uses two non-overlapping four-source held-out sets. The first set also appears in the two-seed replay and the separate pressure-trace check. In the two-GPU study, it is evaluated under expanded contexts, unexpanded contexts, and public continuation, yielding $4 \times 3 \times 5 \times 2 = 120$ episodes. A second set, selected by source-ID order before outcomes were inspected and disjoint from development and the first set, contributes $4 \times 1 \times 5 \times 2 = 40$ unexpanded-context episodes. Each set contains two HotpotQA and two MATH sources; results are reported separately and never pooled. Greedy decoding uses seed 17, method order rotates by source and GPU, and the physical cache is reset before every episode. Public continuation alternates A/B by source index and is exposed to Workflow-aware as a binary feature. Transition priors are estimated only from twelve development sources with add-one smoothing.

4.2 METRICS AND COMPARISON CHECKS

Post-bind TTFT measures time from bind acknowledgement to the first generated token of the designated final request. Application TTFT additionally includes hook and binding overhead. Executed prefill counts prompt tokens processed by the backend and directly measures avoided recomputation.

Decode throughput covers generation after prefill. Whole-episode wall time covers root invocation through cleanup, including lifecycle calls and pressure requests, after client initialization.

Every paired result is first subjected to eligibility gates. The physical gate requires successful bind, completion, metric collection, and cleanup. The semantic gate requires the same candidate, verifier output, final branch, and answer digest after removing only nondeterministic tool-call identifiers. The provider check compares recorded root, candidate, and final request payloads. The main logs do not provide complete input hashes for verifier and pressure calls; the additional study records all ten provider inputs and the six verifier/pressure output hashes. Candidate and final texts are retained; root text is recovered independently from both subsequent branch prompts in server logs. This verifies the root content used downstream, not an independent capture of its original output stream. It does not retroactively repair earlier logs. A pair that fails any gate is kept in the artifact record but is excluded from a causal performance claim. This is why the missing-output pair in the deterministic baseline replay and the historical output-divergent groups are reported explicitly rather than silently dropped from the paper.

For the primary tables, means are computed over eligible paired observations. The held-out uncertainty analysis first averages differences within each of the four source tasks and then enumerates all $4^4 = 256$ ordered bootstrap resamples of the four source tasks. This avoids treating the eight task-seed observations as eight independent tasks. We use the intervals descriptively because the source count is small; the interval including zero for Transition-aware is retained as a limitation.

5 Results

We first verify the physical lifecycle path, then report diagnostic pressure workloads, and finally present the fully audited two-GPU study. We treat the latter as the primary evidence because it records all provider inputs and verifier/pressure outputs and includes equal-information and complete-workflow controls.

5.1 Physical operations

The extension, sidecar, and adapter execute bind, inspect, reserve, metric, and release. The recorded checks report 89 extension tests and 363 Python tests passed, with 26 Python skips. A fresh Python/Pi dependency installation completes four runs on two sources and two policies, with matching outputs within each fresh-install policy pair and four correct offline answers. It reuses the OS, driver, weights, and compilation caches. The injected working directory changes its prompts relative to the main study, so this is a physical-loop check, not cross-installation trace equivalence.

5.2 Diagnostic replay with incomplete output evidence

In a separate 12-trajectory diagnostic, Q-Bridge reduces mean post-bind TTFT from 590.59 to 487.46 ms on 85 observable pairs and total prefill from 158,910 to 97,582 tokens across 86 requests. Episode time changes from 248.55 to 244.27 s. Because one pair lacks output evidence and no quality scores are available, we treat this replay only as corroborating systems evidence. Three alternate arrival traces likewise reduce TTFT by 2.26–18.35% without consistent workflow gains.

5.3 Matched workloads under cache pressure

Table 2 and Figure 2 report the initial pressure workload; the two-GPU study below provides a more complete input audit. All 72 runs on twelve tasks and six policies complete. In this constructed workload, Q-Bridge records a mean post-bind TTFT of 226.65 ms, compared with 641.46, 632.09, and 626.59 ms for Cost-only, Workflow-aware, and Transition-aware. The corresponding observed prefill reduction relative to Cost-only is 78.3%. We use this experiment to illustrate the mechanism rather than to claim workload-level generalization. Task-level answer quality is unchanged across policies.

The held-out workload contains four new tasks and an independently generated pressure schedule. A second run uses the same tasks and schedule with decoding seed 29. All 48 policy runs pass

Table 2: Final-request performance under pressure. Post-bind TTFT: ms; prefill: tokens; decode: tokens/s.

Policy	Mean TTFT	Mean prefill	Mean decode	Exact match
Q-Bridge	226.65	657.9	41.73	6/12
Blind	959.53	5,465.9	40.07	6/12
Wrong-branch	806.85	3,872.6	37.91	6/12
Cost-only	641.46	3,036.6	41.80	6/12
Workflow-aware	632.09	3,036.6	41.12	6/12
Transition-aware	626.59	2,911.2	40.48	6/12

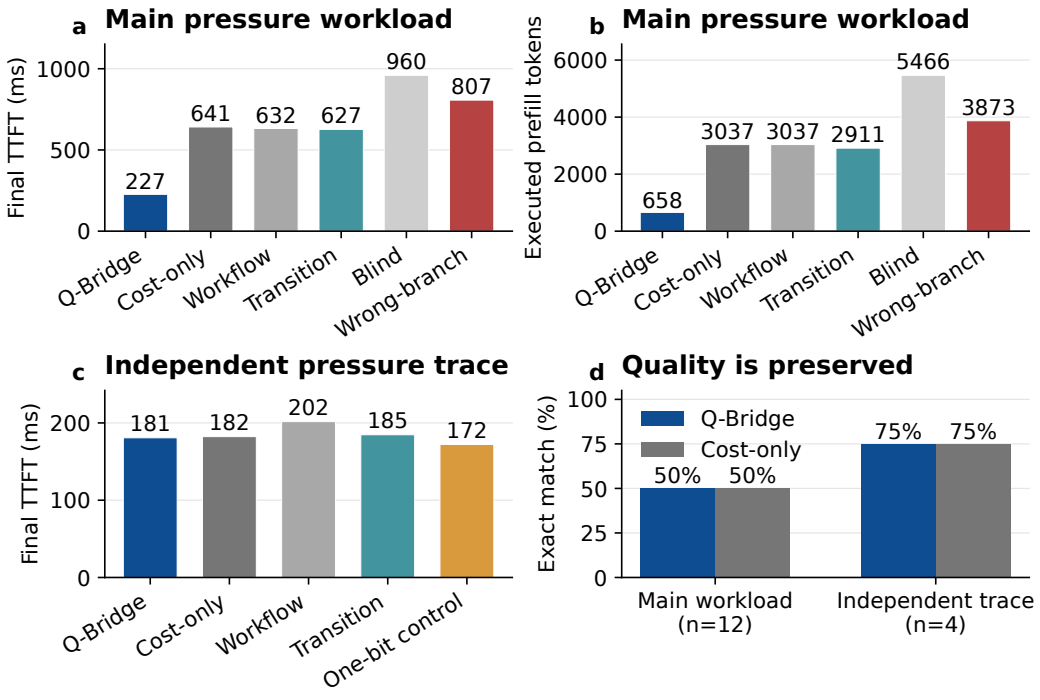

Figure 2: **Cache performance and task accuracy.** Panels a/b show means for the initial pressure workload (12 tasks); c shows the separate pressure-trace check described at the end of this subsection, not the two-seed means in Table 3; d shows exact match. The latter check reuses four held-out sources.

across the two seeds, and every task passes semantic and provider-payload identity gates. Both seeds preserve 75% exact match for every policy.

Q-Bridge has lower final-request TTFT than Cost-only, Workflow-aware, and Transition-aware in 8/8, 8/8, and 6/8 task-seed pairs, respectively. We first average differences within each source, then enumerate all 256 source-cluster bootstrap resamples. The resulting descriptive 95% intervals are [203.12, 620.26], [210.09, 615.41], and [-1.84, 494.11] ms. The Cost-only and Workflow-aware intervals remain positive across all enumerated resamples; the Transition-aware interval crosses zero.

The recency ablation completes 28 runs: selected retention reaches 166.20 ms post-bind TTFT versus 600.42 ms for one-way live-context retention, with 75% exact match for both. This unequal-information diagnostic retains the currently live context and is distinct from the same-decision one-bit control. All 20 runs complete on the second pressure trace, where the earlier advantage largely disappears: TTFT is 180.74 ms for Q-Bridge, 182.26 for Cost-only, 201.68 for Workflow-aware, 184.70 for Transition-aware, and 172.16 for the one-bit control. Each method scores 3/4 exact match. Pressure pattern and information availability both matter.

Table 3: Held-out results: four tasks, two decoding seeds. Units as in Table 2; exact match counts task-seed observations.

Policy	Mean TTFT	Mean prefill	Mean decode	Exact match
Q-Bridge	173.71	504.2	40.70	6/8
Blind	731.54	4,468.2	39.69	6/8
Wrong-branch	589.85	3,216.2	41.09	6/8
Cost-only	585.40	3,216.2	41.11	6/8
Workflow-aware	586.47	3,216.2	38.88	6/8
Transition-aware	394.03	1,924.2	38.67	6/8

Table 4: Two-GPU comparison. Each row averages four sources on two V100s. Post-bind TTFT: final request, ms; prefill: executed tokens; workflow: seconds from root invocation through cleanup.

Policy	TTFT ↓	Prefill ↓	Workflow ↓
Expanded contexts; original four held-out sources			
Q-Bridge	178.67	504.25	64.816
Cost-only	599.81	3216.25	63.400
Workflow-aware	600.74	3216.25	64.736
Transition-aware	191.28	504.25	61.948
One-bit control	188.34	504.25	62.533
Unexpanded contexts; same four sources			
Q-Bridge	140.79	409.00	56.340
Cost-only	181.38	761.00	56.733
Workflow-aware	180.72	761.00	55.456
Transition-aware	172.63	753.00	56.372
One-bit control	130.67	409.00	56.276
Public continuation; expanded contexts, same sources			
Q-Bridge	170.63	498.50	60.301
Cost-only	370.59	1790.50	60.618
Workflow-aware	174.47	498.50	60.423
Transition-aware	404.28	1918.50	60.750
One-bit control	178.25	498.50	60.211
Unexpanded contexts; four additional independent sources			
Q-Bridge	188.38	634.25	73.142
Cost-only	271.66	1186.25	72.669
Workflow-aware	268.95	1186.25	71.788
Transition-aware	232.99	1006.25	71.691
One-bit control	187.46	634.25	72.598

5.4 Complete workflows on two GPUs

The additional study completes all 160 assigned episodes with physical acknowledgements, cleanup, and matched model inputs and outputs. All 80 cross-GPU trace pairs match, and every root request reads zero cached tokens after reset. The eight source tasks, not the GPU repeats or conditions, define the independent sample count. We do not pool these observations with the earlier cohorts.

Local gains and complete costs. Table 4 separates local cache gains from episode cost. Relative to Cost-only, Q-Bridge's mean post-bind TTFT and prefill are lower by 70.2% and 84.3% for expanded contexts, while mean workflow time is 2.23% higher. For the original unexpanded sources, the observed means are 22.4% lower TTFT, 46.3% lower prefill, and 0.69% lower workflow time; for the additional sources they are 30.7% lower TTFT, 46.5% lower prefill, and 0.65% higher workflow time. Offline exact match is unchanged within every condition (75% for the original sources and 50% for the additional sources). HotpotQA EM/F1 and MATH boxed-answer matching use external gold only after execution.

For the unexpanded original and additional source sets, the Cost-only-minus-Q-Bridge TTFT differences are 40.59 and 83.29 ms, with source-cluster 95% intervals of $[-9.46, 111.97]$ and

$[-19.84, 264.07]$ ms. Both intervals include zero; complete-workflow intervals also include zero in the unexpanded sets, while the expanded-context interval is $[-2.10, -0.73]$ s and confirms the observed slowdown.

Equal information and measurement scope. When the continuation is public, Workflow-aware and Q-Bridge both execute 498.5 prefill tokens, and the same-decision one-bit control matches Q-Bridge's prefill in all four conditions. This isolates the source of the measured cache benefit: the retention decision and its physical enforcement, rather than message richness. Because the one-bit encoding was not separately optimized, this experiment does not rank wire encodings.

All 1,600 phase-level first-delta observations are present. Application TTFT on the additional sources, including hook and binding overhead, averages 196.99 ms for Q-Bridge and 280.15 ms for Cost-only. Across conditions, Q-Bridge uses 49.94–63.06 ms of cumulative sidecar service time and exchanges 12.16–13.32 KB of lifecycle traffic per episode. These totals exclude model API traffic and client scheduling. Two directory-creation failures occurred before model requests; their configurations were run at the queue end, with the original errors retained. No completed model run was selected for repetition based on its result.

The final model request accounts for only 0.59–1.37% of mean complete workflow time in Cost-only. Root and candidate generation and pressure calls dominate this workload, so the final-request gains do not imply faster workflows in general. Fresh-install checks also shared host CPU and disk resources during part of the study.

5.5 Shared-prefix lifecycle check

A further check uses 48 new HotpotQA sources in two arrival batches. Each source creates two evidence-review sessions, revisited after all sessions are prepared, without repeated documents or dummy pressure. Four policies execute 768 matched requests with a 64-token output cap. The per-context policy and Q-Bridge execute 114,552 and 114,600 prefill tokens, respectively, with mean batch times of 63.316 and 64.430 s. An independently implemented shared-block greedy control exactly reproduces Q-Bridge's quotas and prefill. Thus physical-union accounting neither improves this workload nor distinguishes Q-Bridge from a simple equal-information allocator. All policies obtain 11/48 and 18/48 exact matches on the two batches' role answers. The artifact reports source selection, instrumented baseline costs, and all paired traces; two batches do not establish timing generalization.

6 Related work

Agent execution and reasoning. ReAct and Toolformer study reasoning with actions and learned tool use (Yao et al., 2022; Schick et al., 2023); AutoGen and MetaGPT organize agent interactions (Wu et al., 2023; Hong et al., 2023). Chain-of-thought, Tree of Thoughts, and Graph of Thoughts structure intermediate reasoning (Wei et al., 2022; Yao et al., 2023; Besta et al., 2023). These methods motivate contexts with distinct future uses. Our evaluation instead fixes the resulting trajectory to isolate cache retention.

Serving and application-level scheduling. PagedAttention manages physical KV blocks, and SGLang exposes structured generation with prefix reuse (Kwon et al., 2023; Zheng et al., 2023b). DistServe separates prefill and decoding, while Sarathi-Serve schedules their interaction (Zhong et al., 2024; Agrawal et al., 2024). FlexGen and DeepSpeed Inference address inference under memory and execution constraints (Sheng et al., 2023; Aminabadi et al., 2022). Parrot exposes application knowledge through semantic variables; Teola represents finer-grained workflow dependencies; and Autellix schedules LLM programs using program-level context (Lin et al., 2024; Tan et al., 2024; Luo et al., 2025). Pie further allows applications to program serving behavior, including explicit KV allocation, release, and sharing (Gim et al., 2025). These systems use application information to guide serving decisions. Q-Bridge addresses the specific problem of admitting a cache lease within a physical block budget and measuring its effect while holding the application trace fixed.

Cache placement, reuse, and lifecycle. Mooncake organizes serving around a distributed KV store, and Preble combines prefix reuse with distributed request scheduling (Qin et al., 2024; Srivatsa et al., 2024). CacheGen compresses and streams cached state; CacheBlend combines cached knowledge chunks; and LMCache provides an external KV-cache layer (Liu et al., 2023b; Yao et al., 2024; Liu et al., 2025). KVFlow uses workflow execution proximity, PBKV predicts future agent invocations, and CacheScout learns execution transitions for eviction and prefetching (Pan et al., 2025; Zheng et al., 2026; Zhang et al., 2026). Continuum retains state through tool pauses with time-to-live, and TokenCake coordinates memory management and scheduling (Li et al., 2025; Bian et al., 2025). Q-Bridge studies an explicit application-selected retention signal admitted against observed physical capacity. Our Cost-only, Workflow-aware, and Transition-aware are local comparison policies rather than full reimplementations of the published systems. The experiments consequently do not compare Q-Bridge with their complete implementations.

Complementary kernel and cache-footprint optimizations. FlashAttention, FlashAttention-2, and FlashInfer optimize attention (Dao et al., 2022; Dao, 2023; Ye et al., 2025). H_2O, SnapKV, and StreamingLLM select KV subsets; KIVI quantizes state (Zhang et al., 2023; Li et al., 2024; Xiao et al., 2023; Liu et al., 2024). We allocate retention across completed contexts, without combining these techniques or claiming their decode-speed benefits.

Evaluation and correctness. AgentBench, SWE-bench, and LongBench evaluate agent completion, repository repair, and long-context understanding (Liu et al., 2023a; Jimenez et al., 2023; Bai et al., 2023). We use small subsets of MATH and HotpotQA (Hendrycks et al., 2021; Yang et al., 2018), not full-benchmark evaluations. Known limitations of LLM judges (Zheng et al., 2023a) motivate our separation of physical completion, unchanged outputs, and externally scored correctness.

7 Failure analysis and limitations

Changing the final reasoning branch would confound cache effects with application behavior. Because the verifier is not an independent correctness oracle, we apply external gold only after execution. The two-GPU study contains eight sources, and the lifecycle check contains two arrival batches. Both use constructed workflows. V100 repeats do not establish distributed or cross-hardware behavior, and equal-information controls match Q-Bridge's reuse. The results show local retention gains, but do not establish stable end-to-end acceleration, a calibrated cost model, or superiority over complete published systems. Earlier missing-output and output-divergent comparisons remain diagnostic only.

8 Conclusion

Q-Bridge provides an implementable path from an application-selected retention decision to a physically enforced KV lease. Across the measured pressure workloads, these leases reduce final-request prefill and often lower mean post-bind TTFT while preserving matched outputs and offline answer scores. Same-decision controls show that the gain comes from retention choice rather than protocol richness, and whole-workflow measurements identify cases where local savings are too small to dominate total execution. The result is a validated interface and a scoped cache-pressure benefit, not a claim of universal end-to-end acceleration.

AI USE STATEMENT

In this work, we used generative AI tools to assist with brainstorming and refining the research methodology, drafting and polishing portions of the manuscript, translating text, and implementing and debugging parts of the software artifact and experiment scripts. We also used such tools to help search and organize related literature and to suggest experimental checks. We did not delegate the final interpretation of the results, the reported claims, or the evaluation decisions to an AI system. All AI-assisted text, code, and analysis were reviewed and tested by the authors, and we take responsibility for the final content of this work, including its text, claims, data processing, and artifacts.

ETHICS STATEMENT

This work involves no human-subject data collection, human participant experiments, personal data, or interaction with human annotators. The evaluation uses publicly available benchmark sources and locally executed language-model inference. Generated model outputs are used only for the controlled application traces and are not presented as human judgments. The reported cache measurements concern system performance and do not expose personal or confidential information.

REPRODUCIBILITY STATEMENT

The frozen agent, model, inference, and infrastructure versions are specified in the configuration files accompanying this submission. Artifacts cover the 160 episodes forming 80 cross-GPU pairs and the 768-request lifecycle check. They include failures, scores, timings, source-level differences, and contract-test mappings. The supplementary core package contains the Pi extension, the validating Infra controller, the Unix-socket protocol schemas, the vLLM adapter boundary, and the smoke-test entry points. A post-review source snapshot adds imported scorers and control dependencies without claiming a pre-run freeze. Root recovery and fresh installation retain the boundaries described in this paper.

REFERENCES

Amey Agrawal et al. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. arXiv:2403.02310.

Reza Yazdani Aminabadi et al. 2022. DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. arXiv:2207.00032.

Yushi Bai et al. 2023. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. arXiv:2308.14508.

Maciej Besta et al. 2023. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. arXiv:2308.09687.

Zhuohang Bian et al. 2025. TokenCake: A KV-Cache-centric Serving Framework for LLM-based Multi-Agent Applications. arXiv:2510.18586v4.

Tri Dao et al. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv:2205.14135.

Tri Dao 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. arXiv:2307.08691.

In Gim et al. 2025. Pie: A Programmable Serving System for Emerging LLM Applications. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles (SOSP)*, 415–430.

Dan Hendrycks et al. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. arXiv:2103.03874.

Sirui Hong et al. 2023. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352.

Carlos E. Jimenez et al. 2023. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. arXiv:2310.06770.

Woosuk Kwon et al. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*, 611–626.

Hanchen Li et al. 2025. Continuum: Efficient and Robust Multi-Turn LLM Agent Scheduling with KV Cache Time-to-Live. arXiv:2511.02230v7.

Yuhong Li et al. 2024. SnapKV: LLM Knows What You are Looking for Before Generation. arXiv:2404.14469.

Chaofan Lin et al. 2024. Parrot: Efficient Serving of LLM-based Applications with Semantic Variable. arXiv:2405.19888.

Xiao Liu et al. 2023a. AgentBench: Evaluating LLMs as Agents. arXiv:2308.03688.

Yuhan Liu et al. 2023b. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. arXiv:2310.07240.

Yuhan Liu et al. 2025. LMCache: An Efficient KV Cache Layer for Enterprise-Scale LLM Inference. arXiv:2510.09665.

Zirui Liu et al. 2024. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. arXiv:2402.02750.

Michael Luo et al. 2025. Autellix: An Efficient Serving Engine for LLM Agents as General Programs. arXiv:2502.13965.

Zaifeng Pan et al. 2025. KVFlow: Efficient Prefix Caching for Accelerating LLM-Based Multi-Agent Workflows. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*.

Ruoyu Qin et al. 2024. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. arXiv:2407.00079.

Timo Schick et al. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761.

Ying Sheng et al. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. arXiv:2303.06865.

Vikranth Srivatsa et al. 2024. Preble: Efficient Distributed Prompt Scheduling for LLM Serving. arXiv:2407.00023.

Xin Tan et al. 2024. Teola: Towards End-to-End Optimization of LLM-based Applications. arXiv:2407.00326.

Jason Wei et al. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.

Qingyun Wu et al. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155.

Guangxuan Xiao et al. 2023. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453.

Zhilin Yang et al. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. arXiv:1809.09600.

Jiayi Yao et al. 2024. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. arXiv:2405.16444.

Shunyu Yao et al. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629.

Shunyu Yao et al. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601.

Zihao Ye et al. 2025. FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving. arXiv:2501.01005.

Rui Zhang et al. 2026. Learning Agent Execution for KV-Cache Management in Agentic Serving. arXiv:2608.14624.

Zhenyu Zhang et al. 2023. H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. arXiv:2306.14048.

Haoyu Zheng et al. 2026. Efficient Serving for Dynamic Agent Workflows with Prediction-based KV-Cache Management. arXiv:2605.06472.

Lianmin Zheng et al. 2023a. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685.

Lianmin Zheng et al. 2023b. SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104.

Yinmin Zhong et al. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. arXiv:2401.09670.