ATXSERVE: TOWARDS TRANSACTION-AWARE INFERENCE SERVING FOR MULTIAGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language model agents increasingly collaborate in shared workspaces to solve complex tasks. However, concurrent updates by multiple agents can compromise consistency or invalidate costly inference. A natural approach is to model these operations as database transactions and apply conventional concurrency control. In agent workloads, prolonged model inference dominates the interval between read and write operations. Optimistic validation detects conflicts only after expensive inference has been performed, while pessimistic locking can suppress useful concurrency throughout inference. This mismatch creates a fundamental tension between preserving consistency and GPU efficiency. We address this mismatch by shifting the optimization focus from state access to model execution. Specifically, we reformulate agent concurrency control as a GPU scheduling problem and introduce ATXSERVE. ATXSERVE constructs a conflict graph before main inference, orders conflicting requests, and admits compatible requests together. After admission, requests validate their planning reads when binding current snapshots. The resulting protection covers planning and execution dependencies through commit. This design prevents inference invalidation by declared conflicts between managed transactions while allowing compatible transactions to execute concurrently. To evaluate ATXSERVE across diverse concurrency scenarios, we develop RMWBench, which emulates multiagent workloads with controllable conflicts. On the Commit0 benchmark, ATXSERVE shows mean useful throughput gains of 27.3–49.5% over optimistic concurrency control.

1 INTRODUCTION

Large language model agents are increasingly deployed as collaborative systems in which multiple agents divide complex tasks and make progress concurrently (Hong et al., 2024; Wu et al., 2024; Qian et al., 2024; Chen et al., 2024). In domains such as software development and document editing, these agents coordinate through shared workspaces. However, sharing state also creates conflicts. Two agents may read the same function and generate edits independently, and once one edit commits, the other may no longer apply to the updated version (Figure 1). A stale edit then fails at commit even though the model has already consumed substantial GPU time. Coordinating shared state therefore also means avoiding wasted computation. A natural remedy is concurrency control from database systems, such as optimistic (OCC) and pessimistic (PCC) protocols (Kung & Robinson, 1981; Bernstein et al., 1987). But agent workloads differ from classical database transactions, since the gap between reading state and writing an update is dominated by seconds of inference. Under OCC, changes to bound inputs are detected at commit, so each such rejection wastes an entire model call. PCC protects shared state throughout inference, but first-in, first-out (FIFO) admission can reduce concurrency among compatible requests. These make conventional database protocols difficult to apply directly to agent workloads, where expensive inference dominates the interval between reading and writing. **This mismatch demands a fundamental rethinking of concurrency control for agent workloads.**

Conflicts should be scheduled, not merely validated. Conventional concurrency control focuses on shared state, while agent transactions often spend much of their time in model inference between reads and writes. As a result, state-level conflict handling does not directly control which inference requests should run together. This gap suggests bringing conflict awareness into GPU admission. At admission time, conflicts indicate which transactions should be ordered and which can run together. We therefore

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.

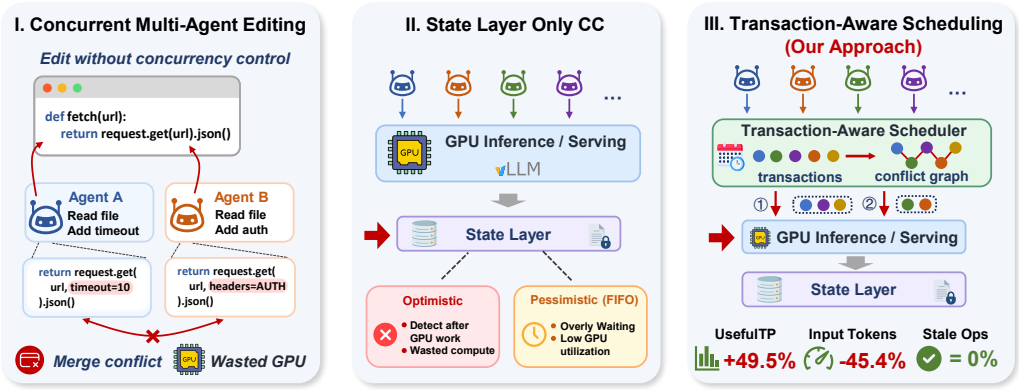


Figure 1: **Transaction-aware admission resolves conflicts before main inference.** (i) Concurrent edits conflict. (ii) OCC can waste inference; FIFO protection can delay compatible requests. (iii) ATXSERVE orders conflicts, batches compatible requests, and protects snapshots through commit.

propose transaction-aware serving, which uses these conflicts to guide inference admission. This design orders conflicting transactions before main inference, so conflict handling can shape GPU execution.

Our approach. ATXSERVE uses workspace conflicts to coordinate both state protection and GPU admission. ATXSERVE represents the declared read and write dependencies of concurrent transactions as a conflict graph. The graph identifies conflicting transactions that must be ordered and compatible transactions that can execute together. This admission policy orders conflicting transactions before inference while preserving concurrency among compatible transactions. After admission, ATXSERVE validates the planning versions when binding the execution snapshot. Once the snapshot is bound, the grant protects both planning and execution dependencies. The grant and admission logic remain outside the serving engine, leaving vLLM's internal scheduling unchanged (Kwon et al., 2023).

To evaluate ATXSERVE, we develop RMWBench to systematically study transaction-aware serving under diverse contention patterns. We further evaluate ATXSERVE on software development workloads Commit0 (Zhao et al., 2025) with real source code. The controlled benchmark shows how contention affects serving efficiency, while the Commit0 workloads measure these gains on real source code.

Contributions. This work makes three contributions. **(i)** We reformulate multiagent concurrency control as a GPU admission problem, where workspace conflicts become scheduling signals for expensive model inference. **(ii)** We design and implement ATXSERVE without changing the serving engine. It orders conflicting transactions before inference, admits compatible transactions concurrently, and protects their snapshots through commit. **(iii)** We develop RMWBench to study transaction-aware serving under controlled contention and evaluate ATXSERVE on Commit0 workloads with real source code. Across three Commit0 repositories, Qwen3 4B experiments show mean useful throughput gains over OCC of 27.3% at Medium contention and 49.5% at High contention.

2 PROBLEM FORMULATION

Transaction Model. We consider multiple agents operating on a shared workspace composed of versioned blocks $\mathcal{B} = \{b_1, \ldots, b_m\}$, where each block represents a file or an independently updated region. Each transaction T_i performs model inference over shared state and may update part of the workspace. Before the main inference begins, it exposes a bounded declaration $D_i = (\mathcal{R}_i^D, \mathcal{W}_i^D)$ that specifies the blocks it may read and write during the current stage. The declared reads also cover mutable workspace blocks observed during planning. Execution is restricted to this declaration, while longer tasks proceed through a sequence of such bounded stages. These declarations determine which transactions must be ordered. Two transactions conflict whenever one may write a block that the other may read or write,

$$\text{Conflict}(i,j) \iff \left(\mathcal{W}_i^D \cap (\mathcal{R}_j^D \cup \mathcal{W}_j^D) \neq \emptyset\right) \vee \left(\mathcal{W}_j^D \cap \mathcal{R}_i^D \neq \emptyset\right). \quad (1)$$

In contrast, transactions that only read the same blocks do not conflict and may execute concurrently. Because execution remains within the declared sets, this relation covers all accesses allowed during the stage. After admission, a transaction binds the current workspace state only if the versions recorded during planning remain current. Let s_i denote the successful binding time and c_i its commit time. We define its protected interval as $\tau_i = [s_i, c_i]$, which includes the main model inference. To keep the bound state valid until commit, conflicting transactions must have disjoint protected intervals,

$$\mathsf{Conflict}(i, j) \implies \tau_i \cap \tau_j = \emptyset. \tag{2}$$

This protection is logical, so physical workspace locks are needed only during atomic snapshot binding and publication. Binding checks the recorded planning versions under the same locks used to read the snapshot.

Objective. We call an operation *useful* only if it commits correctly, passes task-specific semantic checks, remains fresh at commit, and causes no lost update. Let $u_i = 1$ when transaction T_i satisfies these conditions and $u_i = 0$ otherwise. We define USEFULTP (useful throughput) as the number of useful committed operations per GPU hour. For a run allocated one GPU, let E denote end-to-end wall-clock time in hours, including planning, waiting, and retries. We then define

$$\text{USEFULTP} = \frac{\sum_i u_i}{E}. \tag{3}$$

Thus, USEFULTP measures how efficiently GPU time is converted into valid committed work. Maximizing USEFULTP requires avoiding invalid work while preserving concurrency among transactions that do not conflict. At time t, the scheduler observes waiting transactions $\mathcal{Q}_t$, active protected transactions $\mathcal{A}_t$, and an admission capacity K that limits the number of active logical grants. It selects a set $S_t \subseteq \mathcal{Q}_t$ whose transactions are mutually nonconflicting and do not conflict with any active transaction, subject to $|\mathcal{A}_t| + |S_t| \leq K$. The objective is to maximize USEFULTP by ordering conflicting transactions while allowing nonconflicting transactions to execute concurrently.

3 ATXSERVE DESIGN

ATXSERVE controls transaction admission before main inference and leaves token scheduling to the serving engine. It uses workspace conflicts to admit compatible transactions together and order conflicting ones before inference. Once admitted, each transaction receives a logical grant that protects its workspace dependencies through commit. To maintain progress, ATXSERVE gives priority to transactions repeatedly bypassed by conflicting requests. Figure 3 summarizes the workflow.

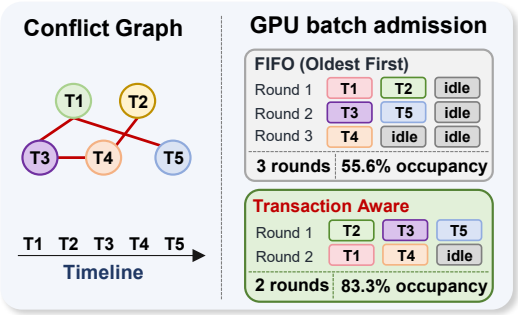


Figure 2: **Conflict-graph admission preserves compatible overlap.** Here, ATXSERVE reduces three FIFO rounds to two and raises admission-slot occupancy from 55.6% to 83.3%.

3.1 CONFLICT-AWARE ADMISSION

Declare the access scope before inference. Before main inference, ATXSERVE uses a short planning request to select a target and declare the blocks it may read or write. This declaration specifies *where* the transaction may act, while main inference determines *what* update to produce. The planning request reads workspace state but does not modify it or acquire a grant. During planning and resource resolution, the runtime maps declared resources to concrete blocks and records each observed block and version in H_i. These recorded blocks are added to the declaration's read set before admission. Admission uses resolved declarations rather than access hints. Execution then remains within the resolved scope, and the workspace records actual reads for commit validation. If execution needs an undeclared resource, the stage stops and replans with a new declaration.

Build the admission graph. The admission manager builds a conflict graph from these declarations, including the recorded planning reads. Its vertices represent waiting transactions, and its edges connect pairs that conflict under Equation 1. Read–read overlap alone creates no edge. The manager first identifies transactions compatible with all active grants,

$$F = \{i \in \mathcal{Q} \mid \forall j \in \mathcal{A},\ \neg\mathsf{Conflict}(i, j)\}\,. \tag{4}$$

Transactions outside F wait for conflicting active grants to be released; those inside F may still conflict with one another. Let $G_F = (F, E_F)$ be the conflict graph induced by F. Each admission round seeks a largest independent set of G_F, a set of pairwise compatible transactions, within the remaining capacity:

$$S^\star \in \arg\max_{S \subseteq F} |S|, \qquad \text{subject to } |S| + |\mathcal{A}| \le K, (i,j) \notin E_F \quad \forall i, j \in S,\ i \neq j. \tag{5}$$

Finding the optimal admission set is NP-hard, since the problem becomes the standard maximum independent set problem when capacity is unrestricted (Halldórsson & Radhakrishnan, 1997).

To approximate this objective, ATXSERVE uses minimum-degree greedy selection to exclude as few remaining candidates as possible at each step. It selects a vertex with minimum degree, breaking ties by arrival order, activates its grant, and removes the vertex and its neighbors from the candidate graph. Degrees are then updated, and selection continues until capacity is full or no candidate remains. The resulting set S_{g} is compatible with active work and internally compatible. For a fixed feasible graph without reservation overrides, it is maximal under the conflict and capacity constraints.

Preserve compatible parallelism. Conflict-free admission can still leave capacity unused because an early choice can block several otherwise compatible transactions. Figure 2 illustrates this with five transactions and three admission slots.

FIFO first selects T_1, which excludes T_3 and T_5 from the round. Adding T_2 also excludes T_4, leaving the third slot idle. The remaining transactions require two more rounds, $\{T_3, T_5\}$ and $\{T_4\}$. Minimum-degree selection instead starts with T_2, the earliest of the degree-one vertices, excluding only T_4. On the remaining graph, it selects T_3, which excludes T_1 but leaves T_5 available. The first round therefore fills all three slots with $\{T_2, T_3, T_5\}$, and the second admits $\{T_1, T_4\}$. Both schedules enforce the same conflict constraints; the gain comes from ordering transactions to preserve compatible choices within each round.

Algorithm 1 Compatible admission with conflict-specific reservation

Require: Waiting $\mathcal{Q}$; active $\mathcal{A}$; capacity K; threshold B
$S \leftarrow \emptyset$
while $|\mathcal{A}| < K$ **do**
  $F \leftarrow \{i \in \mathcal{Q} : \forall j \in \mathcal{A}, \neg\mathsf{Conflict}(i, j)\}$
  $P \leftarrow \{i \in \mathcal{Q} : o_i \ge B\}$
  $r \leftarrow$ oldest transaction in P, or $\perp$
  if $r \neq \perp$ and $r \in F$ **then**
    $p \leftarrow r$
  else
    if $r \neq \perp$ **then**
      $F \leftarrow \{i \in F : \neg\mathsf{Conflict}(i, r)\}$
    if $F = \emptyset$ **then break**
    $p \leftarrow$ min-degree vertex in F, oldest on ties
  for each older $i \in \mathcal{Q}$ with $\mathsf{Conflict}(i, p)$ **do**
    $o_i \leftarrow o_i + 1$
  Issue grant p
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{p\}$; $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{p\}$; $S \leftarrow S \cup \{p\}$
return S

We adopt minimum-degree selection for compatible admission (Halldórsson & Radhakrishnan, 1997). For a fixed feasible graph of maximum degree Δ_F with reservation disabled, this heuristic satisfies $|S_{\mathrm{g}}| \ge \min\{1, 3/(\Delta_F + 2)\}|S^\star|$ under the capacity cap.

3.2 PROTECTED EXECUTION

Prioritize repeatedly bypassed requests. Minimum-degree selection can repeatedly bypass older transactions in favor of newer ones with fewer conflicts. Algorithm 1 combines the minimum-degree selection in Section 3.1 with a conflict-specific reservation rule to protect these waiters. ATXSERVE tracks these conflicting overtakes with a counter o_i for each waiter T_i, incremented only when a later conflicting transaction is admitted first. At each admission decision, the oldest waiter with $o_i \ge B$ becomes the current reservation. If feasible, it takes priority over minimum-degree selection; otherwise, new transactions that conflict with it are withheld while independent transactions remain eligible. Each grant is recorded immediately, so subsequent selections account for earlier admissions in the same round.

While a transaction remains the current reservation, its active conflict set cannot grow. If active grants finish or expire in finite time and the reservation remains current until these conflicts clear, it is selected at the next admission decision with available capacity (Appendix B). The threshold B activates this conflict-specific protection; it does not bound a waiter's total overtakes across changes in the current reservation.

Protect bound state through commit. The state used to choose a target during planning must remain valid when execution begins. In Figure 3, the client obtains a logical grant from the admission manager (step 2) before binding its snapshot in the shared workspace (step 3). Under the binding locks, the workspace verifies that the grant is valid and covers the declaration, then checks every version in H_i. These checks and the snapshot read form one atomic operation, so binding succeeds only if the recorded planning versions remain current. If a planning version has changed, binding returns `STALE_PLAN`. The client then releases the grant, discards the plan and its recorded observations, and replans before issuing any main inference request.

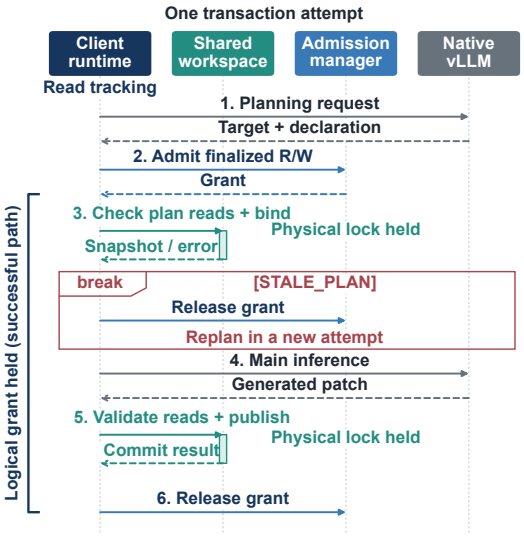


Figure 3: **A logical grant protects planning and execution dependencies through commit.**

After successful binding, the client sends the selected execution inputs from the snapshot to vLLM (step 4). The grant protects planning-only blocks without adding them to execution inputs. The grant covers the full declaration through commit, so managed transactions cannot modify either planning or execution dependencies. At patch publication (step 5), the workspace briefly reacquires physical locks to verify the grant and validate both H_i and the actual execution reads. The workspace publishes the update atomically only if all checks succeed. Validation also detects changes to those blocks by writers outside the admission manager, provided they are tracked by the versioned workspace. After commit or termination, the client releases the grant so compatible waiters can proceed (step 6).

3.3 Protocol Guarantees

Correctness Guarantees. Successful binding validates the planning observations against the bound snapshot, and the declared read set covers these observations and execution reads. With complete, enforced declarations and grants held through commit, conflicting managed transactions cannot overlap from binding to publication. This nonoverlap preserves their relative order, and atomic binding and publication therefore yield a conflict-serializable history. Compatible transactions can still overlap, so the protocol preserves concurrency when conflicts are absent. This compatibility also gives a simple completion bound for a fixed batch. With reservation disabled, no replanning, and equal protected duration p, repeated maximal admission finishes at most $p\Delta$ later than the offline optimum, where Δ is the maximum conflict graph degree. Appendix B gives the formal proofs, and Appendix B.1 extends the completion bound to unequal durations.

Serving integration. ATXSERVE decides which transactions enter main inference, while vLLM retains its native batching, token scheduling, and memory management. This division lets the prototype set the admission cap K to vLLM's `max_num_seqs` (Appendix B.3). Each grant occupies one admission slot until release or expiration, so K limits the number of active logical grants. Within these grants, vLLM continues to schedule sequences and tokens at each engine step. Short planning requests run before admission, and their cost is included in the measured runtime and model work.

4 RMWBench

RMWBench evaluates concurrent read-modify-write operations over a shared versioned workspace. Each operation reads workspace state and produces an update that is useful only if correct and valid at commit. Table 1 summarizes the benchmark capabilities. SWE-bench, Commit0, and τ-bench include task checks, but their cited protocols do not include conflict control, read-freshness checks, or USEFULTP. RMWBench combines these dimensions to distinguish correct task outputs from updates that remain valid under concurrent execution.

Table 1: Benchmark capabilities.

Benchmark	Task checks	Conflict control	Read freshness	USEFULTP
SWE-bench (Jimenez et al., 2024)	✓	✗	✗	✗
Commit0 (Zhao et al., 2025)	✓	✗	✗	✗
τ-bench (Yao et al., 2024)	✓	✗	✗	✗
RMWBench	✓	✓	✓	✓

✓ included in the cited protocol; ✗ not included.

RMW Task Structure. As shown in Figure 4a, the workspace contains independently versioned source, catalog, and target blocks. Source blocks provide evidence, the catalog maps a routing key to a target, and the target stores the mutable summary. Figure 4b shows the corresponding task flow. A task resolves its target, reads the required workspace state, produces a short patch, and validates its reads before commit. Multiple tasks execute concurrently, so their read and write sets may overlap. RMWBench supports policy comparisons under fixed task semantics. It controls access overlap to create shared reads, read–write (RW) conflicts, and write–write (WW) conflicts.

Deterministic Scoring. RMWBench records each operation's read blocks and versions, then evaluates its update at commit. Deterministic checks cover routing, semantic correctness, read coverage, freshness, preservation of unrelated content, and lost updates. For example, given a valid `red` fact and a newer invalid `amber` fact, the expected patch contains `red` and the earlier timestamp. Even a semantically correct patch fails validation if a read block changes before commit. Only committed updates passing all checks contribute to USEFULTP, with costs including planning, retries, and main inference. Repository workloads apply the same protocol and scoring to real source context (Appendix C).

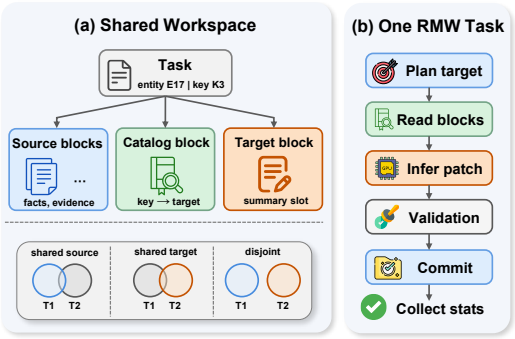


Figure 4: **RMWBench and task workflow.**

5 EVALUATION

5.1 EXPERIMENTAL SETUP

The main experiments serve Qwen3 4B (Qwen Team, 2025). The transaction-control comparisons replay fixed tasks, targets, and declarations with a shared planner and declaration enforcement. Their time and token totals include planning and retries (Appendix C). We compare OCC validation at commit, OCC Backoff with jitter after failed validation, PCC-FIFO with compatible admission in arrival order, and ATXSERVE with $B = 8$ reservation. Additional controls isolate admission order, reservation, and placement.

RMWBench provides controlled RW and WW conflicts, while Commit Focus uses access patterns from Commit0 repositories' commit histories (Zhao et al., 2025). Its 96 fixed tasks are grouped by write target into 12 eight-task waves. Dispersed selects one task per queue per pass; Focused fills waves from the longest remaining queues. Mixed takes up to four tasks from the longest queue, then fills remaining slots as in Dispersed. Both use one MetaX C550; comparisons share a GPU and measurement epoch, with cross-host timings reported separately. Unless noted otherwise, throughput changes use ratios of mean per-run rates. Tables mark best and second-best measured values in **bold** and underline, including ties.

5.2 END-TO-END RESULTS

Resolving conflicts before inference improved useful throughput once contention made optimistic execution costly. Across 128 operations from eight agents, ATXSERVE's mean USEFULTP was 0.4% below OCC at Low contention and 8.3–11.2% higher at Medium and High (Figure 5a). At High, it also exceeded PCC-FIFO by 11.8%, showing the value of preserving compatible parallelism alongside state protection. Commit Focus tests conflict concentration by regrouping the same 96 tasks

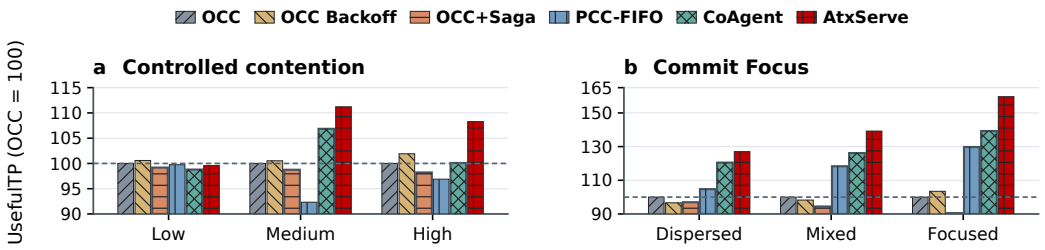


Figure 5: UsefulTP normalized to OCC (=100). (a) Controlled; (b) Commit Focus.

and declarations into Dispersed, Mixed, and Focused waves. The mean gain over OCC increased from 27.0% in Dispersed to 59.6% in Focused (Figure 5b). Concentrating updates on shared targets thus increased the benefit of ordering conflicting requests before main inference.

Source-editing tasks connect this throughput gain to the cost of failed attempts and recovery (Figure 6a,b). Across three serialized-work fractions, every policy passed nine of 48 evaluations per setting, comprising 16 tasks repeated three times. With this matched success count, ATXSERVE achieved 22.8% higher successful-task throughput than OCC Backoff and 6.0% higher throughput than PCC-FIFO at 50% serialized work. Its extra input/output token cost over Serial was 1.2k per passing evaluation, compared with 25.3k for OCC Backoff, including failed attempts and recovery. Pre-inference coordination therefore reduced the model work needed to obtain the same number of successful outcomes.

Candidate ordering contributed beyond simply skipping blocked requests (Figure 6c). With reservation disabled and other admission settings matched, minimum-degree selection raised mean USEFULTP over PCC-FIFO by 4.9% at Controlled High and 12.3% at Focused. Across three runs per rule and workload, useful completion counts were unchanged. Minimum-degree selection preserves compatible admission choices by excluding fewer remaining candidates at each step.

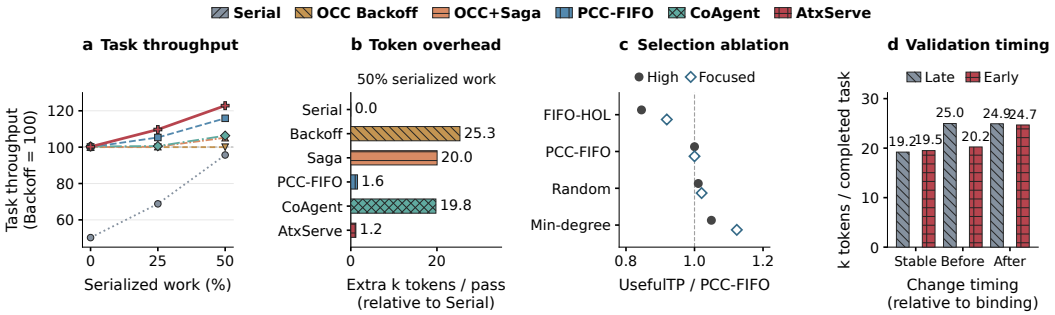


Figure 6: End-to-end efficiency and mechanism ablations. (a) Successful-task throughput (Backoff = 100 per setting). (b) Extra tokens per passing evaluation versus Serial. (c) Selection-rule UsefulTP relative to PCC-FIFO. (d) Tokens per completed task under early versus late planning validation.

5.3 INFERENCE WORK

Table 2 separates inference savings from admission efficiency. Including planning and all attempts, ATXSERVE used 45.4–53.0% fewer input tokens than OCC across Focused and Controlled High. In Focused, stale operations, retries, and aborts fell to zero; waste dropped from 77.0% to 29.6% (Table 9). Remaining waste included semantically incorrect commits. At Controlled High, higher USEFULTP than OCC reflected more useful completions despite longer elapsed time. PCC-FIFO matched ATXSERVE's input-token totals but took longer in both workloads, supporting an additional benefit from admission order.

5.4 ACCESS ROBUSTNESS

Using resolved declarations ties the admission graph to the enforced execution scope. We perturbed access hints with tasks and enforced declarations fixed (Figure 7a,b). The w/o Resolution control builds its graph directly from hints, so false positives serialize compatible work and false negatives leave conflicts unscheduled. At Low contention, FP50 (50% precision, full recall) reduced this

Table 2: Inference accounting, including planning and all attempts. Useful counts operations passing all checks.

	Controlled High (128 ops)			Commit Focus: Focused (96 ops)		
Policy	Input (M) ↓	Useful ↑	Elapsed (s) ↓	Input (M) ↓	Useful ↑	Elapsed (s) ↓
OCC	0.734	82.0	142.2	1.431	44.3	209.1
OCC Backoff	0.709	83.0	**141.1**	1.416	45.3	206.6
OCC+Saga	0.734	82.0	144.5	1.431	41.0	214.3
PCC-FIFO	**0.345**	97.0	173.4	**0.782**	51.0	184.9
CoAgent	0.502	97.3	168.5	1.073	52.3	177.3
ATXSERVE	**0.345**	**97.7**	156.4	**0.782**	**57.0**	**168.3**

control's throughput by 65.4% without stale operations or aborts. At High, FN50 (approximately 50% recall, full precision) produced means of 39.0 stale operations and 10.0 aborts. ATXSERVE built its graph from the fixed resolved declarations and recorded zero stale operations, retries, or aborts across tested profiles (Appendix E.1).

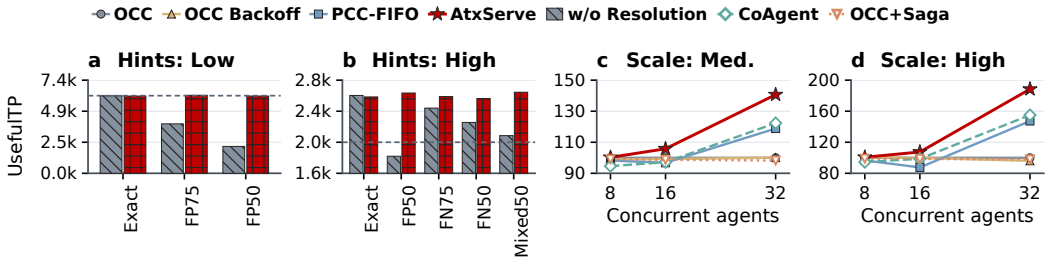


Figure 7: Hint robustness and concurrency scaling. (a,b) Graph-source ablation at Low/High contention with absolute UsefulTP ticks in k ops/GPU-hour. (c,d) Scaling at Medium/High contention with OCC = 100 within each setting.

Conservative Declarations. Declaration scope and model inputs affect different parts of execution. At a fixed snapshot, expanding declarations alone preserved a mean of 124/128 correct answers; adding those resources to model inputs reduced it to 119/128 at FP50. Under concurrency, broader declarations reduced active grant occupancy and lowered useful completion rates by 26.9% at Low and 24.8% at High. All 128 tasks still committed, and the High rate remained 1.5% above OCC (Appendix E.3). Broader declarations thus restricted parallelism even when all tasks committed; expanding model inputs separately affected semantic accuracy.

Dynamic Access. The source-editing case allows execution to discover dependencies and trigger replanning. Validating planning reads at binding (Early) tests whether these dependencies remain current before model work begins (Figure 6d). When dependencies changed before binding, Early used 18.9% fewer tokens per completed task than validation only at commit (Late), rejecting stale plans before main inference. Each path completed 48 tasks per condition, with token costs normalized by completions independently of task-test outcomes. Costs were similar with stable dependencies and with external versioned changes after binding. These timings distinguish the two checks: binding validation avoids inference on stale plans, while final validation detects subsequent external changes. Managed conflicting writers wait for grants and cannot introduce such changes during protected execution.

5.5 CONTENTION AT SCALE

Sustained Load. Longer context and generation extend the period during which concurrent updates can invalidate inference. With 512 operations from 32 agents at High contention, ATXSERVE and PCC-FIFO committed every operation, while OCC left some operations aborted (Table 3). Across three seeds, ATXSERVE delivered 103.7% higher mean USEFULTP than OCC, combining complete commits

Table 3: Completion and throughput at High contention.

Policy	Commits ↑	Aborts ↓	USEFULTP ↑
OCC	365.0	147.0	1430.5
OCC+Saga	365.0	147.0	1397.7
PCC-FIFO	**512.0**	**0.0**	1221.1
CoAgent	511.0	1.0	2169.7
ATXSERVE	**512.0**	**0.0**	**2913.6**

with compatible parallelism. A separate 60-second arrival test examined reservation's trade-off between waiter progress and aggregate stage throughput. With $B = 8$, the maximum observed conflicting overtakes fell from 97 to 8, at the cost of lower completed-stage throughput and more unfinished stages (Appendix F.3).

Concurrency scaling. In the 1,024-operation workloads, gains over OCC rose from below 1% with eight agents to 40.7% (Medium) and 88.4% (High) with 32 (Figure 7c,d). This trend is consistent with greater conflict overlap favoring pre-inference coordination. GPU assignments were matched within settings but could differ across agent counts.

6 Related Work

Agent coordination. MetaGPT, AutoGen, ChatDev, and AgentVerse coordinate agents through roles, messages, and tools (Hong et al., 2024; Wu et al., 2024; Qian et al., 2024; Chen et al., 2024). ReAct and SWE-agent connect inference to actions (Yao et al., 2023; Yang et al., 2024), while SWE-bench evaluates software agents (Jimenez et al., 2024). ATXSERVE instead coordinates shared workspace access, where updates can invalidate another agent's inference inputs.

Model serving. Orca, vLLM, and SGLang improve scheduling, attention memory, and structured execution (Yu et al., 2022; Kwon et al., 2023; Zheng et al., 2024). DistServe, Sarathi-Serve, and Fast-Serve optimize prefill/decode scheduling (Zhong et al., 2024; Agrawal et al., 2024; Wu et al., 2026), while fair serving addresses client allocation (Sheng et al., 2024). Nexus partitions prefill/decode resources (Shi et al., 2025), BROS manages request priorities and KV memory (Wan et al., 2026), and Preble balances prefix reuse with distributed load (Srivatsa et al., 2025). MARS considers resource pressure and session progress (Wang et al., 2026), while KAIROS controls power through agent context (Yuan et al., 2026). WISP coordinates speculative drafting and verification for service objectives (Li et al., 2026). Other agent serving systems schedule workflows, regulate concurrency through KV memory pressure, or retain prefix state (Guo et al., 2026; Chen et al., 2026; Ni et al., 2026a). Chimera routes workflows across heterogeneous LLMs using semantic scores, output-length predictions, and congestion estimates (Ni et al., 2026b). These systems optimize execution or serving state, whereas ATXSERVE uses workspace conflicts to avoid inference for stale updates.

Concurrency control. Locking and optimistic validation protect shared state (Bernstein et al., 1987; Kung & Robinson, 1981), while Silo and Calvin improve throughput through validation and ordering (Tu et al., 2013; Thomson et al., 2012). Sagas recover from failures by compensating completed actions (Garcia-Molina & Salem, 1987). CoAgent coordinates long-running agent transactions through a fixed serialization order, speculative writes, agent-guided repair, and tool inverses (Lyu et al., 2026). ATXSERVE instead uses bounded, enforceable declarations to order conflicts before main inference and holds grants through commit, preventing declared conflicts rather than repairing their effects.

7 Discussion

The results show that workspace dependencies can guide inference admission without changing the serving engine's scheduler. Admission control complements the serving engine's existing batching, preemption, and memory management.

Limitations. The approach requires bounded, enforceable stage declarations and recorded planning versions. Declaration breadth trades concurrency for scope, and the planner remains responsible for obtaining sufficient task information. ATXSERVE protects managed, versioned workspace state; applications retain responsibility for authorization and recovery for irreversible external actions. The prototype stores coordination metadata in memory. Production deployments require durable, replicated storage.

8 Conclusion

ATXSERVE coordinates shared-state protection with inference admission. It orders declared conflicts before main inference while preserving compatible parallelism. Experiments on controlled and repository-based workloads show that this coordination reduces repeated inference under contention and increases the rate of valid committed updates. For agents with bounded, enforceable access declarations, ATXSERVE jointly addresses consistency and serving efficiency. By co-designing the state and serving layers, ATXSERVE offers a foundation for efficient and consistent multiagent execution.

AI USE STATEMENT

We used AI tools to assist with manuscript writing, language polishing, and improvements to clarity and concision. The authors retain full responsibility for the manuscript's content.

REPRODUCIBILITY STATEMENT

The ATXSERVE implementation, benchmark configurations, and experiment scripts are available at `https://anonymous.4open.science/r/atxserve-1FDD/`. The appendices detail the experimental settings, workload construction, and evaluation metrics.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation*, pp. 117–134, 2024. URL `https://www.usenix.org/conference/osdi24/presentation/agrawal`.

Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987. URL `https://www.microsoft.com/en-us/research/people/philbe/book/`.

Qiaoling Chen, Zhisheng Ye, Tian Tang, Peng Sun, Boyu Tian, Guoteng Wang, Shenggui Li, Yonggang Wen, Zhenhua Han, and Tianwei Zhang. CONCUR: High-throughput agentic batch inference of LLM via congestion-based concurrency control. *arXiv preprint arXiv:2601.22705*, 2026. URL `https://arxiv.org/abs/2601.22705`.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=EHg5GDnyq1`.

Hector Garcia-Molina and Kenneth Salem. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pp. 249–259, 1987. doi: 10.1145/38713.38742.

Dongxin Guo, Jikun Wu, and Siu Ming Yiu. SAGA: Workflow-atomic scheduling for AI agent inference on GPU clusters. *arXiv preprint arXiv:2605.00528*, 2026. URL `https://arxiv.org/abs/2605.00528`.

M. M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997. doi: 10.1007/BF02523693.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=VtmBAGCN7o`.

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

H. T. Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, 1981. doi: 10.1145/319566.319567.

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

Xiangchen Li, Jiakun Fan, Qingyuan Wang, Dimitrios Spatharakis, Saeid Ghafouri, Hans Vandierendonck, Deepu John, Bo Ji, Ali R. Butt, and Dimitrios S. Nikolopoulos. WISP: Waste- and interference-suppressed distributed speculative LLM serving at the edge via dynamic drafting and SLO-aware batching. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 10(2):1–31, 2026. doi: 10.1145/3805653.

Hongtao Lyu, Dingyan Zhang, Mingyu Wu, Xingda Wei, and Haibo Chen. CoAgent: Concurrency control for multi-agent systems. *arXiv preprint arXiv:2606.15376*, 2026. URL https://arxiv.org/abs/2606.15376.

Hongqiu Ni, Han Tian, Chi Zhang, Guopeng Li, and Haisheng Tan. TOPAS: Workflow-aware prefix-state scheduling for multi-agent LLM serving. *arXiv preprint arXiv:2608.25523*, 2026a. URL https://arxiv.org/abs/2608.25523.

Kangqi Ni, Wenyue Hua, Xiaoxiang Shi, Jiang Guo, Shiyu Chang, and Tianlong Chen. Chimera: Latency- and performance-aware multi-agent serving for heterogeneous LLMs. *arXiv preprint arXiv:2603.22206*, 2026b. URL https://arxiv.org/abs/2603.22206.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 15174–15186, 2024. doi: 10.18653/v1/2024.acl-long.810.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation*, pp. 965–988, 2024. URL https://www.usenix.org/conference/osdi24/presentation/sheng.

Xiaoxiang Shi, Colin Cai, Junjia Du, and Zhihao Jia. Nexus: Proactive intra-GPU disaggregation of prefill and decode in LLM serving. *arXiv preprint arXiv:2507.06608*, 2025. URL https://arxiv.org/abs/2507.06608.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for LLM serving. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5bc342f48de8264779952fac378f96dc-Abstract-Conference.html.

Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. Calvin: Fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 1–12, 2012. doi: 10.1145/2213836.2213838.

Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. Speedy transactions in multicore in-memory databases. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, pp. 18–32, 2013. doi: 10.1145/2517349.2522713.

Borui Wan, Juntao Zhao, Chenyu Jiang, Chuanxiong Guo, and Chuan Wu. BROS: Efficient LLM serving on hybrid real-time and best-effort requests. In *IEEE INFOCOM 2026 - IEEE Conference on Computer Communications*, pp. 1–10, 2026. doi: 10.1109/INFOCOM59046.2026.11571463.

Yifei Wang, Hancheng Ye, Yechen Xu, Cong Guo, Chiyue Wei, Qinsi Wang, Dongting Li, Tingjun Chen, Hai "Helen" Li, Danyang Zhuo, and Yiran Chen. MARS: Efficient, adaptive co-scheduling for heterogeneous agentic systems. *arXiv preprint arXiv:2604.26963*, 2026. URL https://arxiv.org/abs/2604.26963.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. FastServe: Iteration-level preemptive scheduling for large language model inference. In *23rd USENIX Symposium on Networked Systems Design and Implementation*, pp. 57–74, 2026. URL https://www.usenix.org/conference/nsdi26/presentation/wu-bingyang.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Proceedings of the First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024. doi: 10.52202/079017-1601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

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 *16th USENIX Symposium on Operating Systems Design and Implementation*, pp. 521–538, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Yichao Yuan, Mosharaf Chowdhury, and Nishil Talati. KAIROS: Stateful, context-aware power-efficient agentic inference serving. *arXiv preprint arXiv:2604.16682*, 2026. URL https://arxiv.org/abs/2604.16682.

Wenting Zhao, Nan Jiang, Celine Lee, Justin T. Chiu, Claire Cardie, Matthias Gallé, and Alexander M. Rush. Commit0: Library generation from scratch. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=MMwaQEVsAg.

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

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 *18th USENIX Symposium on Operating Systems Design and Implementation*, pp. 193–210, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

A Cross-Platform Transfer

We evaluate Qwen3 4B on an NVIDIA RTX A6000 using matched workloads and serving configurations. Table 4 compares the A6000 measurements with the reported C550 transaction-control rates from Table 14. Each A6000 entry is the mean useful throughput over three experimental runs.

Table 4: Cross-platform useful throughput with Qwen3 4B. Rates are useful operations per GPU hour; measured A6000 values average three runs.

	Controlled High		Commit Focus Focused	
Policy	C550	A6000	C550	A6000
OCC	2079.4	1321.2	763.7	523.6
OCC Backoff	2119.2	1373.3	790.1	547.1
PCC-FIFO	2014.7	1403.6	991.1	688.6
AtxServe	2251.6	1511.8	1219.2	868.4

B Protocol Analysis

We formalize the guarantees in Section 3 for blocks b with content x_b and monotonically increasing versions $\mathsf{ver}(b)$.

Proposition 1. *Each admission round returns a maximal set under its conflict, capacity, and reservation constraints. If fewer than K slots are occupied, every remaining waiter conflicts with an active grant, a selected transaction, or the current reservation.*

At capacity, no further request fits. Below capacity, Algorithm 1 removes selected requests, their neighbors, and conflicts of an infeasible reservation. Thus, termination leaves no admissible candidate, proving maximality rather than maximum cardinality.

For completion analysis, fix a graph G of n transactions available at time zero. Their declarations include planning reads and permit successful binding without replanning. For maximum degree Δ, clique number ω, and capacity K, define $h = \lceil n/K \rceil$ and $\ell = \max\{h, \omega\}$. Let $q^\star$ and q_A count rounds under optimal scheduling and repeated minimum-degree selection, respectively. Each round contains at most K compatible transactions.

Proposition 2. *Assume no future arrivals, external writes, or reservation, and zero planning, validation, and admission overhead. Each round begins after the preceding grants finish. Then $q_A \le h + \Delta$, while the offline optimum satisfies $q^\star \ge \ell$.*

Choose v from the last round. At most $\lfloor (n-1)/K \rfloor$ earlier rounds are full; each other round contains a distinct neighbor of v by maximality. Hence $q_A \le h + \Delta$. Capacity and clique size give $q^\star \ge \ell$, so equal protected duration p yields

$$0 \le C^A_{\max} - C^\star_{\max} \le p(h + \Delta - \ell), \qquad \frac{C^A_{\max}}{C^\star_{\max}} \le \frac{h + \Delta}{\ell}. \tag{6}$$

Since $\ell \ge h$, the gap is at most $p\Delta$. For unequal durations, the corresponding capacity and clique lower bounds are

$$C^\star_{\max} \ge \max\left\{ \frac{1}{K} \sum_i p_i, \max_{C \text{ clique}} \sum_{i \in C} p_i \right\}. \tag{7}$$

These bounds apply to the fixed batch specified above.

For consistency, H_i records all mutable planning and resolution observations, including repeated reads. Their versions advance on every mutation and are never reused after deletion or recreation. Atomic binding validates H_i against the execution snapshot, so each observation can be treated as a read at binding. The declaration includes these reads, and the grant protects them through commit.

Proposition 3. *Assume exact stage declarations that include planning reads, successful binding, atomic publication, and a valid logical grant held from binding through commit. The committed history of managed transactions is conflict serializable. No managed transaction can invalidate another admitted transaction's basis during its protected interval.*

By Equation 2, conflicting protected intervals are disjoint. Their precedence edges advance in bind time, so a topological order preserves every conflict. The same interval separation prevents managed writes from invalidating protected reads. For external writes tracked by the workspace, publication checks recorded versions. Section B.2 extends this argument to conservative declarations.

Proposition 4. *Assume a valid grant excludes every managed transaction that conflicts with T_i until commit. Successful atomic binding returns a consistent execution snapshot and validates all recorded planning observations at the same logical time. Those values remain protected from managed conflicting writes until c_i.*

Binding verifies the grant and declaration under block locks acquired in fixed order. Under the same locks, validation of H_i and snapshot reads share one instant s_i. After lock release, the grant maintains protection until termination or atomic publication, which rechecks the grant, versions, and scope.

Proposition 5. *Assume active leases finish or expire in finite time. While T_r remains the current reservation, Algorithm 1 excludes new conflicting grants. If T_r remains reserved until its active conflicts drain, the next decision with available capacity admits it.*

Exclusion lets existing conflicts drain while independent requests remain eligible. Each decision recomputes the oldest waiter above B; the proposition describes progress while that reservation remains current.

B.1 Unequal Durations

Now give each grant in a nonempty fixed graph $G = (V, E)$ a positive duration p_i independent of scheduling. Retain the fixed batch assumptions above, with complete enforced declarations, capacity $K \geq 1$, no reservation or external writes, and zero protocol overhead. Admission is maximal initially and after each completion. For closed neighborhoods $N[i] = \{i\} \cup \{j : (i, j) \in E\}$, define

$$W = \sum_{i \in V} p_i, \qquad \Gamma = \max_{i \in V} \sum_{j \in N[i]} p_j, \tag{8}$$

Here W is total duration, and Γ is the largest duration sum over a closed neighborhood. The completion times of AtxServe and the optimum under identical constraints satisfy

$$C^A_{\max} \leq \frac{W}{K} + \left(1 - \frac{1}{K}\right)\Gamma, \qquad \frac{C^A_{\max}}{C^\star_{\max}} \leq 1 + \frac{(K-1)\Gamma}{W}. \tag{9}$$

Proof of Equation 9. Let v finish last and $m(t)$ count active grants. When $m(t) < K$, maximality requires v or an active neighbor to occupy a slot. Their intervals cover all underfilled time, with total length at most Γ and at most $K - 1$ idle slots. Since active slot time equals W, idle slot time satisfies

$$KC^A_{\max} - W = \int_0^{C^A_{\max}} (K - m(t))\, dt \leq (K-1)\Gamma. \tag{10}$$

Division by K gives the first bound; $C^\star_{\max} \geq W/K$ gives the ratio. □

For $\varepsilon \geq 0$, $(K-1)\Gamma \leq \varepsilon W$ therefore implies a $(1+\varepsilon)$ completion ratio for any maximal compatible admission rule. This fixed graph analysis leaves replanning costs to empirical measurement. If all $n = |V|$ transactions are useful and durations are in seconds on one GPU, it also gives

$$\text{USEFULTP}_A \geq \frac{3600n}{W/K + (1 - 1/K)\Gamma}. \tag{11}$$

B.2 Conservative Declarations

Conservative declarations add permissions to a complete scope without changing the intended task or target. Let $\mathcal{R}^{\text{act}}_i$ contain the blocks actually read during planning and execution, and $\mathcal{W}^{\text{act}}_i$ contain the blocks actually written. For a reference declaration $D_i = (\mathcal{R}^D_i, \mathcal{W}^D_i)$ and an expanded declaration $\widehat{D}_i = (\widehat{\mathcal{R}}_i, \widehat{\mathcal{W}}_i)$, write

$$D_i \preceq \widehat{D}_i \quad \Longleftrightarrow \quad \mathcal{R}^D_i \subseteq \widehat{\mathcal{R}}_i \ \wedge\ \mathcal{W}^D_i \subseteq \widehat{\mathcal{W}}_i. \tag{12}$$

Read and write permissions must separately cover actual execution,

$$\mathcal{R}_i^{\mathrm{act}} \subseteq \widehat{\mathcal{R}}_i, \qquad \mathcal{W}_i^{\mathrm{act}} \subseteq \widehat{\mathcal{W}}_i. \tag{13}$$

Coverage must hold for the resulting execution, even if added permissions change its accesses. The planner remains responsible for obtaining sufficient task evidence.

Proposition 6 (Conflict monotonicity under declaration expansion)**.** *Fix a set of transaction vertices and two families of declarations with $D_i \preceq \widehat{D}_i$ for every i. Let $G_D = (V, E_D)$ and $G_{\widehat{D}} = (V, E_{\widehat{D}})$ be their conflict graphs under Equation 1. Then $E_D \subseteq E_{\widehat{D}}$. In particular, under Equation 13, every actual conflict is a declared conflict.*

Proof. Each nonempty intersection in Equation 1 remains nonempty when its constituent sets expand, proving $E_D \subseteq E_{\widehat{D}}$. Applying this inclusion to actual access sets proves the final statement. □

Expansion can only remove feasible independent sets for fixed vertices, active transactions, and capacity. Online throughput can still vary because expansion changes ordering, binding times, and execution durations.

Proposition 7 (Safety under conservative declarations)**.** *Assume managed transactions without external writes satisfy Equation 13, atomic planning validation, and admission under Equation 2. Each attempt reads from its bound snapshot and retains a valid grant until termination or atomic publication; failed attempts publish nothing. Then commits are conflict serializable, and managed writes preserve protected read values even when declarations are conservative.*

Proof. Proposition 6 implies

$$\mathsf{Conflict}_{\mathrm{act}}(i,j) \Longrightarrow \mathsf{Conflict}_{\mathrm{decl}}(i,j) \Longrightarrow \tau_i \cap \tau_j = \emptyset. \tag{14}$$

Represent validated planning observations as reads at binding. Every actual precedence edge advances in bind time, so a topological order preserves every conflict. Equation 14 also excludes managed writes to $\mathcal{R}_i^{\mathrm{act}}$ during τ_i, proving snapshot protection as in Proposition 4. □

B.2.1 Semantic Correctness

For task and target q_i at state x, let $Y_i(D, x; \xi) = \mathsf{Exec}_i(q_i, x, D; \xi)$ denote the proposed update, with failure $\perp$. Its predicate $\mathsf{Sem}_i(q_i, x, y)$ checks the answer and target preservation, with $\mathsf{Sem}_i(q_i, x, \perp) = 0$. An expansion is *execution noninterfering* if it preserves execution at fixed x. This requires identical task inputs, prompts, tools, model, decoding, randomness distribution, parsing, and update construction. It also requires unchanged truncation and serving effects on outputs, including cancellation.

Proposition 8 (Semantic preservation at a fixed snapshot)**.** *Fix q_i and x. If $D_i \preceq \widehat{D}_i$ preserves execution under the conditions above, then the two executors have the same output distribution. In particular,*

$$\begin{aligned} &\Pr_{\xi}\Big[\mathsf{Sem}_i(q_i, x, Y_i(\widehat{D}_i, x; \xi)) = 1\Big] \\ &\quad = \Pr_{\xi}[\mathsf{Sem}_i(q_i, x, Y_i(D_i, x; \xi)) = 1]\,. \end{aligned} \tag{15}$$

This equality compares attempts at the same snapshot; scheduling and commit selection can change aggregate accuracy.

Proof. Coupling identical execution procedures and inputs with the same ξ gives $Y_i(\widehat{D}_i, x; \xi) = Y_i(D_i, x; \xi)$, including $\perp$. Applying the same semantic predicate proves Equation 15. □

Proposition 9 (Conditional semantic validity at commit)**.** *Assume Proposition 7 and an execution noninterfering expansion. Suppose the reference executor returns only valid updates at every state reachable under either policy, and protected reads cover all semantic dependencies. Then published updates are semantically valid immediately before publication.*

Algorithm 2 Planning, binding, and publication for one attempt

Require: Task q_i; versioned workspace; admission manager
defer discard all records of this attempt
$(target, inputs, D_i, H_i) \leftarrow$ PLANANDRESOLVE(q_i)
$\mathcal{R}_i^D \leftarrow \mathcal{R}_i^D \cup \{b : (b, v) \in H_i\}$
$g \leftarrow$ ADMIT(D_i) ▷ Algorithm 1
defer RELEASE(g)
Atomic binding under workspace locks
if grant g is invalid or does not cover D_i **then**
  return `TERMINATE`
if $\exists (b, v) \in H_i : \mathsf{ver}(b) \neq v$ **then**
  return `REPLAN(STALE_PLAN)`
if *inputs* exceed the declared read scope **then**
  return `REPLAN(UNDECLARED_ACCESS)`
$(snapshot, reads) \leftarrow$ READSNAPSHOT$(inputs)$
Release workspace locks; retain grant g
$patch \leftarrow$ EXECUTE$(q_i, target, snapshot, D_i)$
if execution requests an undeclared resource **then**
  return `REPLAN(UNDECLARED_ACCESS)`
if execution produces no patch **then**
  return `TERMINATE`
Atomic publication under workspace locks
if grant g is invalid or does not cover D_i **then**
  return `TERMINATE`
if any planning or execution read version has changed **then**
  return `REPLAN(STALE_READ)`
if actual accesses or *patch* exceed D_i **then**
  return `REPLAN(UNDECLARED_ACCESS)`
PUBLISH$(target, patch)$
return `COMMITTED`

Proof. Proposition 8 preserves validity at bound state x. Protection preserves every semantic dependency until the state x^- immediately before publication, so

$$\mathsf{Sem}_i(q_i, x, Y_i) = \mathsf{Sem}_i(q_i, x^-, Y_i) = 1.$$

Atomic publication applies that valid update. □

These dependencies include target content that the update must preserve. Coverage of external services and time requires explicit modeling, and semantic validity assumes executor correctness. For example, extra context may cause the executor to answer for entity B instead of A despite fresh reads. This change violates execution noninterference, which Appendix E.3 tests separately from hint noise in Appendix E.1. Missing permissions instead trigger replanning when accessed; a necessary input never requested can still cause a semantic failure.

B.3 RUNTIME PROTOCOL

Algorithm 2 specifies one attempt against unmodified vLLM. Planning records mutable observations in H_i from prompt construction through resource resolution and adds their blocks to declared reads. The client submits this declaration to the admission endpoint, which groups arrivals over one millisecond and applies Algorithm 1. Admission returns grant identifiers and wait/bypass counters; grants and expiration deadlines remain in memory. Each grant occupies a slot until release or expiration, with $K =$`max_num_seqs`.

Binding and publication use the same block lock order and synchronize grant checks with lease cleanup. Binding returns selected contents and read handles; the client sends only those contents to vLLM while the grant protects all declared dependencies. Deferred grant release clears protection before attempt records are discarded, including on exceptions and cancellation. Cleanup returns the failure reason to the retry policy, and failed attempts publish nothing. Traces associate attempts with planning records, grants, binding, model calls, commits, and release. These records distinguish planning invalidation from commit rejection and account for all attempts.

C EXPERIMENTAL METHODOLOGY

Table 5 specifies the shared task protocol; workloads vary in payload, conflicts, and concurrency. Additional studies examine candidate selection, dynamic access, and paired policy comparisons (Appendices D.1, G, and D.4). We report results by configuration, with arrival stress measured in completed model stages rather than useful updates.

Table 5: Detailed experimental configuration

Item	Setting
Model / accelerator	Qwen3 4B/8B/14B, BF16/TP1; one 64 GB MetaX C550 per trial
Serving engine	vLLM 0.19 development build; eager, FCFS, no prefix cache
Workflow	Short target plan; declared scope; main request; validated commit
Controlled / stress	128 ops, 8 agents / 512 ops, 32 agents
Scale sweep	1024 ops; 8, 16, and 32 agents; Medium and High
Commit0 / Commit Focus	64 ops per repository / 96 commits in 12 waves of eight
Contention (WW / RW)	Low 0 / 0; Medium 0.5 / 0.25; High 1.0 / 0.5
Facts / targets	4 candidates, 2 relevant / 16 planning candidates
Workspace / retry budget	OCC on actual reads / three retries
Core and progress seeds	7, 17, 27
Model/scaling seeds	7, 17, 27

C.1 WORKLOADS

Versioned evidence blocks contain entity status facts, timestamps, validity markers, and distractors. Catalog blocks route tasks to targets, whose mutable summary slots sit among immutable anchors. Each task exposes its entity, sources, routing key, and 16 target candidates but hides the correct target and answer. The planner selects a target and declares its scope; main inference then returns the latest valid status and timestamp. The client inserts this patch into the summary slot and submits a full target replacement. Workload waves vary overlap independently of these facts. Low, Medium, and High use write conflict/read after write rates of 0/0, 0.5/0.25, and 1.0/0.5, respectively. WW and RW rates specify the fractions of operations assigned the corresponding conflict patterns, not the fraction of conflicting transaction pairs. An operation may participate in both patterns, so these fractions need not sum to one.

C.2 METRICS

Success counts published writes; abort counts exhausted retries or nonretryable validation errors. Stale counts operations that encounter a target conflict, and retry counts operations with multiple main attempts. Planning invalidation instead returns `STALE_PLAN` before main inference, with its cost included in runtime and model work. Model waste includes inference from failed attempts and committed but nonuseful results. Useful results require the expected status and timestamp, complete read coverage, freshness at commit, and no lost update. Freshness checks both H_i and execution handles, including the catalog, supplied sources, and previous target content. Routing and anchor checks also test target selection and preservation of unrelated content. Read amplification divides all workspace reads by successful operations. Table 6 includes planning and every main attempt in token totals and elapsed time. Its useful counts require all checks, which traces associate with planning, binding, model calls, publication, and release. Across matched tasks and seeds, relative changes use ratios of policy means. Latency P95 conditions on successful operations, so completion counts accompany it.

C.3 REPOSITORY WORKLOADS

Commit0 replaces generated context with 32 sorted Python files from pinned Babel, Jinja, and Marshmallow repositories. Each file becomes a versioned block; 960 tokens of source context retain compact anchors for exact scoring. The shared planner chooses among 16 targets for 64 operations from eight rolling agents, with identifiers and declarations fixed across policies. These comparisons cover Qwen3 4B, 8B, and 14B. The larger models use independent BF16/TP1 servers on four 64 GB

Table 6: Inference accounting. M/k denotes millions/thousands of tokens, and TP denotes USEFULTP.

	Controlled High, 128 operations					Commit Focus Focused, 96 operations				
Policy	Input (M) ↓	Output (k) ↓	Useful ↑	Elapsed (s) ↓	TP ↑	Input (M) ↓	Output (k) ↓	Useful ↑	Elapsed (s) ↓	TP ↑
OCC	0.734	42.34	82.0	142.2	2079.4	1.431	29.78	44.3	209.1	763.7
OCC Backoff	0.709	41.28	83.0	141.1	2119.2	1.416	29.46	45.3	206.6	790.1
OCC+Saga	0.734	42.34	82.0	144.5	2042.3	1.431	29.78	41.0	214.3	690.7
PCC-FIFO	0.345	19.80	97.0	173.4	2014.7	0.782	14.78	51.0	184.9	991.1
CoAgent	0.502	28.84	97.3	168.5	2081.4	1.073	20.26	52.3	177.3	1063.4
ATXSERVE	0.345	19.80	97.7	156.4	2251.6	0.782	14.76	57.0	168.3	1219.2

MetaX C550 GPUs on a second host, with one worker per GPU. Each policy comparison shares a GPU; assignments may differ across models.

Commit Focus also derives targets and context from the 32 latest commits per repository that modify one extant Python file by at most 300 lines. Selection excludes merges and follows reverse chronology independently of conflicts. Each transaction writes the modified file and reads it plus up to three direct internal imports resolved at the pinned anchor. The resulting 96 transactions retain bounded source context and fixed scoring across 12 waves of eight. Tasks are grouped by write target and shuffled within each queue using seeds 7, 17, and 27. Dispersed scans nonempty queues in target-key order, taking one task from each and repeating the scan until eight tasks are selected. Mixed first takes up to four tasks from the longest queue, then fills the wave using dispersed selection, excluding that target on the first scan when alternatives exist. Focused repeatedly selects the longest remaining queue and takes as many tasks as fit until the wave is full. Queue-length ties use target-key order. Only wave assignments change; tasks and declarations remain fixed. These placements yield conflicting pair rates of 22.3%, 26.2%, and 42.3%; write conflicts rise from 5.7% to 35.4% between the extremes. Across Dispersed, Mixed, and Focused, ATXSERVE reduces mean elapsed time relative to OCC by 17.4%, 27.3%, and 19.5%, respectively. The corresponding USEFULTP gains are 27.0%, 39.2%, and 59.6%, using ratios of policy means.

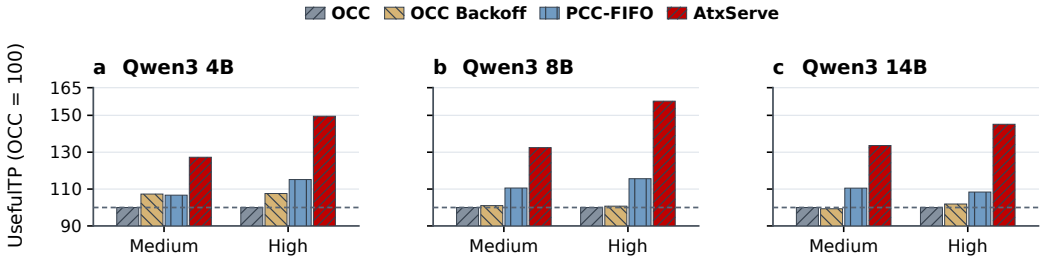


Figure 8: Commit0 results relative to each setting's OCC mean (=100). Panels (a), (b), and (c) average Qwen3 4B, 8B, and 14B results over Babel, Jinja, and Marshmallow.

Across six model/contention aggregates, ATXSERVE gains 27.3% to 57.7% over OCC and exceeds PCC-FIFO in mean throughput. Across these runs, both protected policies commit all 64 operations and record no stale work, retries, or aborts. Table 8 separates these OCC comparisons by repository.

C.3.1 WRITE CONTENTION

Figure 9 shows independent conflict groups of six and two transactions. OCC may run conflicting writers together and retry stale updates; ATXSERVE admits one writer per group until commit releases its grant. This order preserves overlap between groups while serializing writes within each group.

Table 7: Commit0 controls averaged over three repositories. n counts runs per repository; TP denotes UsefulTP.

Model	Policy	n	Medium TP ↑	Medium S/R/A ↓	High TP ↑	High S/R/A ↓
4B	OCC	3	1787.4	25.7/30.0/3.3	1475.8	38.0/39.3/11.0
4B	OCC Backoff	3	1917.6	23.7/28.0/4.0	1587.8	34.3/36.3/9.3
4B	PCC-FIFO	3	1906.9	**0.0/0.0/0.0**	1699.6	**0.0/0.0/0.0**
4B	AtxServe	3	**2274.7**	**0.0/0.0/0.0**	**2206.6**	**0.0/0.0/0.0**
8B	OCC	3	1015.0	24.7/29.0/4.7	940.9	38.7/39.3/11.0
8B	OCC Backoff	3	1025.4	21.3/25.0/6.7	947.8	35.0/35.7/9.7
8B	PCC-FIFO	3	1122.3	**0.0/0.0/0.0**	1087.9	**0.0/0.0/0.0**
8B	AtxServe	3	**1344.8**	**0.0/0.0/0.0**	**1483.8**	**0.0/0.0/0.0**
14B	OCC	3	1150.7	21.7/25.7/5.0	935.4	39.3/41.7/8.3
14B	OCC Backoff	3	1142.9	20.0/24.0/5.0	953.1	34.7/37.0/7.7
14B	PCC-FIFO	3	1271.6	**0.0/0.0/0.0**	1013.9	**0.0/0.0/0.0**
14B	AtxServe	3	**1537.6**	**0.0/0.0/0.0**	**1357.8**	**0.0/0.0/0.0**

Table 8: Commit0 by repository. Arrows show OCC to AtxServe UsefulTP, and changes use ratios of means. n counts runs per policy and repository.

Model	Repository	n	Medium TP	Change	High TP	Change
4B	Babel	3	1909.5 → **2344.6**	+22.8%	1496.9 → **2159.6**	+44.3%
4B	Jinja	3	1797.8 → **2386.1**	+32.7%	1527.8 → **2308.5**	+51.1%
4B	Marshmallow	3	1654.8 → **2093.5**	+26.5%	1402.7 → **2151.6**	+53.4%
8B	Babel	3	1113.3 → **1353.1**	+21.5%	940.9 → **1641.7**	+74.5%
8B	Jinja	3	872.7 → **1246.8**	+42.9%	1008.7 → **1325.6**	+31.4%
8B	Marshmallow	3	1059.2 → **1434.5**	+35.4%	873.2 → **1484.0**	+70.0%
14B	Babel	3	1139.8 → **1498.9**	+31.5%	883.7 → **1407.1**	+59.2%
14B	Jinja	3	1137.5 → **1581.0**	+39.0%	1043.3 → **1366.3**	+31.0%
14B	Marshmallow	3	1174.9 → **1532.9**	+30.5%	879.3 → **1299.9**	+47.8%

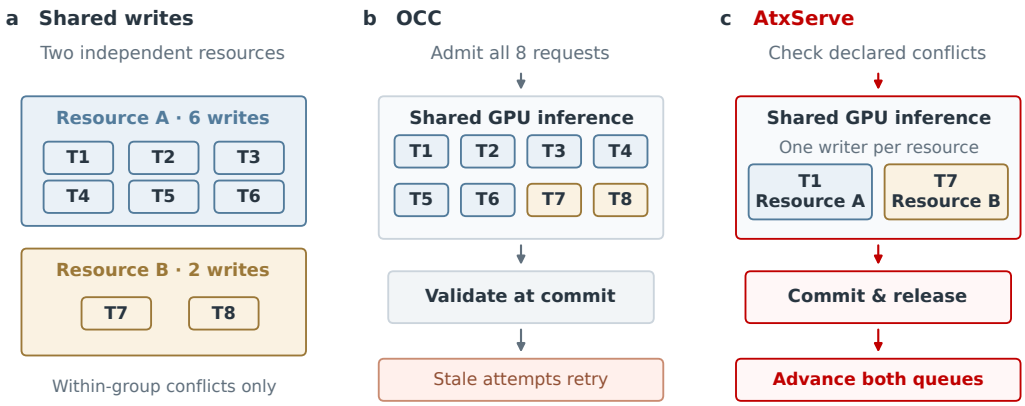


Figure 9: A concentrated write wave. Six transactions update resource A and two update resource B. OCC validates after inference; AtxServe admits one writer per resource together.

Table 9: Commit Focus outcomes (96 operations per setting). S/R/A counts stale/retrying/aborted operations; waste measures model request time yielding no useful work.

Policy	Useful ↑	Commits ↑	S/R/A ↓	Waste (%) ↓	Elapsed (s) ↓
Dispersed, offered conflicting pair rate 22.3%					
OCC	54.0	94.0	9.0/35.0/2.0	59.5	162.4
OCC Backoff	54.7	94.0	9.0/36.3/2.0	59.3	170.1
PCC-FIFO	51.7	96.0	0.0/0.0/0.0	33.7	148.6
CoAgent	54.3	96.0	4.7/11.3/0.0	38.4	135.8
OCC+Saga	53.0	94.0	9.0/35.0/2.0	60.3	164.1
ATXSERVE	56.7	96.0	0.0/0.0/0.0	28.0	134.1
Mixed, offered conflicting pair rate 26.2%					
OCC	56.0	95.0	29.0/43.3/1.0	66.1	204.0
OCC Backoff	55.7	95.0	29.0/43.0/1.0	65.9	206.4
PCC-FIFO	52.0	96.0	0.0/0.0/0.0	34.8	160.4
CoAgent	53.3	96.0	15.3/15.0/0.0	45.9	153.5
OCC+Saga	53.7	95.0	29.0/43.3/1.0	67.4	207.5
ATXSERVE	56.7	96.0	0.0/0.0/0.0	29.3	148.3
Focused, offered conflicting pair rate 42.3%					
OCC	44.3	83.0	49.0/56.0/13.0	77.0	209.1
OCC Backoff	45.3	83.0	49.0/55.7/13.0	76.5	206.6
PCC-FIFO	51.0	96.0	0.0/0.0/0.0	36.5	184.9
CoAgent	52.3	95.7	25.3/17.0/0.3	52.9	177.3
OCC+Saga	41.0	83.0	49.0/56.0/13.0	78.7	214.3
ATXSERVE	57.0	96.0	0.0/0.0/0.0	29.6	168.3

C.4 WORKED EXAMPLES

RMWBench update. Let B_j denote block j in the example workspace file. In a compact seed-7 instance, task `op_00000000` reads sources B_0, B_6 and resolves `route_0000` through catalog B_4 to target B_0. Its declaration is $\mathcal{R}_i^D = \{B_0, B_4, B_6\}$ and $\mathcal{W}_i^D = \{B_0\}$. For the matching task and entity, the relevant fact and a newer distractor are:

```
FACT ts=1000000 status=red valid=1
FACT ts=1000001 status=amber valid=0
patch = {"status":"red","timestamp":1000000}
```

The newer fact is invalid, so the patch selects `red` at timestamp 1000000. The client sets summary slot 0 to these values and submits a full replacement of B_0, preserving the other slots and anchors.

Block-level concurrency. Table 10 separates three access patterns; omitted sources are independent. For WW, replacements based on the same old B_0 can overwrite each other's slots; version validation rejects the stale replacement. For RW, B_0 may remain unchanged while the read source B_1 changes, so publication must validate the read basis.

Table 10: Representative RMWBench access pairs. Each row is a separate example.

Relationship	Task A	Task B	Compatibility
WW	Update B_0, slot 0	Update B_0, slot 1	Conflicting
RW	Read B_1; write B_0	Write B_1	Conflicting
Shared reads	Read B_4; write B_2	Read B_4; write B_3	Compatible

Commit-derived concurrency. Table 11 identifies examples from Babel, whose source snapshot is pinned at `f91754b`. In Focused seed 7, wave 2, the tasks from `221c937` and `a87422f` both write the date-formatting module. They conflict despite distinct cache and exception-handling edits in the source history. The `d2c84b7` task updates the package setup script. Its only shared dependency with either date-formatting task is a read of the package initialization module, allowing compatible

admission. Dispersed seed 7, wave 2 includes the tasks from `139dc11` and `a87422f`. The former writes the support module and reads the date-formatting module through an import dependency; the latter writes the date-formatting module. Their different targets therefore still create an RW conflict. Protecting only the two write targets would leave the support task's read dependency exposed.

Table 11: Babel source commits underlying the concurrent task examples.

Commit	Target component	Source change
`221c937`	Date formatting	Replace the pattern cache with a least-recently-used cache.
`a87422f`	Date formatting	Fix the unbound exception variable `exc`.
`d2c84b7`	Package setup	Move development dependencies to a package extra.
`139dc11`	Support utilities	Fix string conversion in `_locales_to_names`.

D SCHEDULING EVALUATION

Table 12 reports scheduling variants, each including planning and execution within declarations. Rows label engine placement and reservation explicitly; Default uses API placement with $B = 8$.

Table 12: Scheduling results.

Method	Conflict	Elapsed s ↓	P95 ms ↓	USEFULTP ↑	Success ↑	Abort ↓	Stale ↓	Retry ↓
OCC	Low	**56.163**	**3690.8**	**6154.6**	**128.0**	**0.0**	**0.0**	**0.0**
OCC	Medium	108.951	**11362.2**	2863.4	117.3	10.7	51.3	59.0
OCC	High	148.019	13793.1	1979.3	105.7	22.3	84.3	87.7
PCC-FIFO (Engine)	Low	57.269	3786.5	6097.6	**128.0**	**0.0**	**0.0**	**0.0**
PCC-FIFO (Engine)	Medium	128.888	11374.5	2653.9	**128.0**	**0.0**	**0.0**	**0.0**
PCC-FIFO (Engine)	High	162.848	**13036.7**	2184.5	**128.0**	**0.0**	**0.0**	**0.0**
ATXSERVE (Engine, no res.)	Low	56.462	3768.2	6142.6	**128.0**	**0.0**	**0.0**	**0.0**
ATXSERVE (Engine, no res.)	Medium	**103.754**	12795.8	**3353.9**	**128.0**	**0.0**	**0.0**	**0.0**
ATXSERVE (Engine, no res.)	High	**139.099**	16184.0	**2544.7**	**128.0**	**0.0**	**0.0**	**0.0**
ATXSERVE (Default)	Low	56.549	3728.9	6114.3	**128.0**	**0.0**	**0.0**	**0.0**
ATXSERVE (Default)	Medium	108.572	12861.8	3173.4	**128.0**	**0.0**	**0.0**	**0.0**
ATXSERVE (Default)	High	140.765	15646.7	2480.6	**128.0**	**0.0**	**0.0**	**0.0**

All policies complete Low without stale work; contention introduces repeated inference under OCC. Protection removes this waste, and minimum-degree selection preserves compatible overlap.

D.1 CANDIDATE SELECTION

Table 13 isolates selection under shared resolution, validation, grants, capacity, and collection windows, with reservation disabled. FIFO-HOL stops at an incompatible head, whereas PCC-FIFO skips it and Random-compatible samples feasible requests. Minimum-degree selection chooses the lowest-degree candidate and breaks ties by arrival; every selection updates feasibility. These rules use Qwen3 4B with three paired runs on each of Controlled High and Focused, totaling 24 runs. Pairs share tasks and initial state, randomize rule order, and use separate randomness for candidate sampling.

D.2 TRANSACTION BASELINES

Table 14 compares policies on identical operation identifiers, waves, targets, and declarations. OCC Backoff adds seeded random delays after failed OCC validation. PCC-FIFO admits compatible requests in arrival order and holds read/write protection from binding through commit. The underlying records label OCC as Original and PCC-FIFO as Pessimistic FIFO. For these policies, S/R/A denotes stale, retrying, and aborted transactions. Backoff validation uses seeds 37/47/57, separate from test

Table 13: Selection ablation. Values are means of three runs per rule; reservation is off throughout.

Rule	USEFULTP	Useful	Elapsed (s)	Tokens (thousands)	Decision (μs)
Controlled High, 128 operations					
FIFO-HOL	1865.0	97.3	187.9	392.8	19.0
PCC-FIFO	2206.5	97.3	158.8	395.5	28.9
Random-compatible	2230.7	97.3	157.1	401.5	29.5
Minimum degree	2315.1	97.3	151.4	386.5	72.3
Commit Focus Focused, 96 operations					
FIFO-HOL	1006.5	57.0	203.9	845.4	29.3
PCC-FIFO	1095.1	57.0	187.4	865.9	41.0
Random-compatible	1117.8	57.0	183.4	833.7	45.7
Minimum degree	1230.3	57.0	166.7	828.5	97.4

Useful counts valid updates; tokens include planning and all execution attempts. Decision is the mean CPU time per admission decision within each run, then averaged across runs.

seeds 7/17/27 for Controlled, Commit Focus, Commit0, and scaling. The selected base and cap are both 400 ms, so each retry uses a uniform delay from 0 to 400 ms (Appendix F.3).

Table 14: Transaction baseline USEFULTP for Figure 5(a,b).

Workload	Setting	OCC	Backoff	PCC-FIFO	AtxServe	CoAgent	OCC+Saga
Controlled	Low	5963.1	5998.1	5948.1	5937.9	5890.9	5914.3
Controlled	Medium	2868.9	2884.1	2648.9	3190.7	3065.6	2833.2
Controlled	High	2079.4	2119.2	2014.7	2251.6	2081.4	2042.3
Commit Focus	Dispersed	1197.6	1157.2	1254.4	1521.0	1443.1	1161.3
Commit Focus	Mixed	988.2	970.9	1169.4	1375.7	1246.8	933.2
Commit Focus	Focused	763.7	790.1	991.1	1219.2	1063.4	690.7

D.3 POLICY COMPARABILITY

Replay fixes operation order, candidates, targets, declarations, and answers across policies. Online replanning instead recomputes targets and declarations, and checks routing against the catalog at successful binding. The dynamic study therefore pairs initial task groups and seeds but permits different tool traces (Appendix G). Selection tests disable reservation, while paired default tests retain $B = 8$ (Appendix D.4).

These policy comparisons share planning and execution version checks. OCC variants validate H_i at binding and all reads at commit; protected policies additionally retain grants over full declarations. These shared checks isolate admission and protection under equal commit requirements. For ATXSERVE, full protection includes planning dependencies, but only selected execution inputs enter the main prompt. Planning records observed state without a separate probe of every execution input.

D.4 PAIRED COMPARISONS

The paired evaluation compares OCC Backoff, PCC-FIFO, and ATXSERVE with Qwen3 4B across four settings, including Commit0 Babel. Five repetitions per policy and setting give 60 runs (Table 15). Protected policies share API admission, $B = 8$, capacity, collection windows, and validation; OCC Backoff uses the selected 0 to 400 ms delay. Each pair fixes workload, initial state, decoding seed, and GPU, with randomized policy order and independent resets. Policies use the GPU separately and include all planning, attempts, and publication in accounting. Table positions identify repetitions, not seeds; elapsed time equals $3600\,\mathrm{useful}/\mathrm{USEFULTP}$ for each run. These fixed-workload repetitions are separate from the serialized-work sweep in Figure 6a,b. For a zero baseline mean, only absolute differences are defined.

Table 15: Individual paired repetitions. Rates are valid operations per GPU hour; other columns are five-entry means.

Policy	USEFULTP in paired order	Useful	Elapsed (s)	Tokens (thousands)
Controlled Low, 128 operations				
OCC Backoff	5865.1,5984.7,6102.9,5879.0,6047.7	123.6	74.5	139.0
PCC-FIFO	5771.7,5844.1,5954.8,5922.2,5900.3	123.6	75.7	139.0
ATXSERVE	5996.1,6035.5,5841.6,5969.9,5810.3	123.6	75.0	139.0
Controlled High, 128 operations				
OCC Backoff	1931.7,2298.6,2077.2,2151.8,2088.0	83.0	142.1	725.0
PCC-FIFO	2194.6,2149.7,2186.9,2179.6,2195.4	97.8	161.4	388.0
ATXSERVE	2197.6,2329.5,2281.2,2205.6,2500.2	97.8	153.4	385.0
Commit Focus Focused, 96 operations				
OCC Backoff	800.9,801.5,798.0,811.0,774.1	45.4	204.8	1492.0
PCC-FIFO	1063.9,1120.3,1109.2,1129.0,1071.8	57.0	186.9	845.0
ATXSERVE	1226.4,1276.0,1224.7,1251.8,1234.0	57.0	165.1	840.0
Commit0 Babel High, 64 operations				
OCC Backoff	1512.6,1553.0,1479.2,1540.4,1484.9	40.0	95.1	858.0
PCC-FIFO	1945.8,1915.3,1881.7,1989.9,2004.4	50.0	92.5	624.0
ATXSERVE	2137.8,2183.3,2133.9,2240.8,2134.7	50.0	83.1	617.0

Useful counts valid updates. Tokens include planning and all execution attempts. Each run's rate uses its own elapsed time; rates, counts, elapsed times, and tokens are averaged separately.

E ROBUSTNESS DIAGNOSTICS

E.1 HINT ROBUSTNESS

This study perturbs hints after planning and fixes tasks, candidates, targets, declarations, prompts, and answers. Exact retains perfect precision and recall; FP75/FP50 lower precision to 75%/50% at full recall. FN75/FN50 instead lower recall to approximately 75%/50% at full precision, and Mixed50 lowers both to approximately 50%. The w/o Resolution control bypasses resolved declarations when constructing the graph and uses the perturbed hints instead. ATXSERVE constructs the graph from the unchanged resolved declarations. Both paths retain resource resolution, complete grant declarations, and scope and version checks at binding and publication. Thus, missing graph edges can permit conflicting snapshots before publication. Final validation rejects their stale updates after inference. Constructing the graph from resolved declarations prevents that overlap.

Table 16: Hint ablation at Low contention.

Profile	OCC USEFULTP	w/o Resolution USEFULTP	AtxServe USEFULTP	Elapsed (s) w/o Resolution → AtxServe
Exact	**6154.6**	6147.6	6127.0	**55.8 → 55.8**
FP75	6154.6	3922.1	**6184.3**	88.3 → **56.1**
FP50	**6154.6**	2130.8	6128.1	161.2 → **55.8**

Table 17: Mean stale/retry/abort (S/R/A) counts at High contention over three runs.

Hint profile	w/o Resolution S/R/A ↓	AtxServe S/R/A ↓
Exact	**0.0/0.0/0.0**	**0.0/0.0/0.0**
FP50	**0.0/0.0/0.0**	**0.0/0.0/0.0**
FN75	13.7/21.3/3.0	**0.0/0.0/0.0**
FN50	39.0/44.7/10.0	**0.0/0.0/0.0**
Mixed50	34.0/40.7/4.3	**0.0/0.0/0.0**

At Low, all policies commit 128 operations without stale work, retries, or aborts (Table 16). False edges instead serialize compatible work, causing a 65.4% throughput loss for w/o Resolution under FP50. At High, missing edges cause stale inference, with FN50 producing the largest stale and abort counts among profiles with only one error type. Mixed50 combines both effects, whereas ATXSERVE records complete basis coverage and freshness without stale operations, retries, or aborts across tested profiles. Its Low throughput stays within 0.5% of OCC, and its five High means exceed OCC.

Table 18: ATXSERVE relative to OCC across prediction profiles

Conflict	Profile	OCC USEFULTP ↑	AtxServe USEFULTP ↑	USEFULTP change ↑	OCC S/R/A
Low	Exact	**6154.6**	6127.0	−0.4%	0.0/0.0/0.0
Low	FP75	6154.6	**6184.3**	+0.5%	0.0/0.0/0.0
Low	FP50	**6154.6**	6128.1	−0.4%	0.0/0.0/0.0
High	Exact	1979.3	**2557.3**	+29.2%	84.3/87.7/22.3
High	FP50	1979.3	**2607.1**	+31.7%	84.3/87.7/22.3
High	FN75	1979.3	**2562.1**	+29.4%	84.3/87.7/22.3
High	FN50	1979.3	**2535.5**	+28.1%	84.3/87.7/22.3
High	Mixed50	1979.3	**2617.2**	+32.2%	84.3/87.7/22.3

E.2 SNAPSHOT REUSE

Table 19: Workspace read amplification for the deterministic routing diagnostic.

Method	Exact	FP50	FN75	FN50	Mixed50
Full reread	8.00	8.00	8.00	8.00	8.00
Validated reuse	**6.03**	**5.82**	**6.01**	**5.97**	**6.00**

A deterministic routing diagnostic probed all candidate evidence without writes before admission. After admission, it either reread the evidence or reused the snapshot after version validation. We report read amplification for this fixed routing workflow separately from the main planning workflow. Validated reuse reduces workspace reads per success from eight to approximately six at High while retaining correctness.

E.3 DECLARATION SENSITIVITY

This study expands declarations at full recall and precision of 100%, 75%, or 50%, denoted Exact, FP75, and FP50. Fixed-snapshot comparisons separate protection-only changes (Scope-only) from changes that also expand model inputs (Context-added). Separate concurrent comparisons evaluate OCC, PCC-FIFO, and ATXSERVE at Low and High. Each case uses 128 tasks and three runs; semantic percentages divide mean correct counts by 128. Useful rates equal $3600\,\overline{N}_{\text{useful}}/\overline{T}$ for mean time $\overline{T}$ in seconds. We report these ratios of means descriptively; they differ from mean per run rates.

Table 20: Diagnostics at a fixed snapshot. Counts average three runs of 128 tasks; input tokens are per-task means.

Path	Profile	Prompt matches	Input tokens	Semantic correct	Correct to wrong	Wrong to correct
Reference	Exact	128.0/128	1060.0	124.0/128	0.0	0.0
Scope-only	FP75	128.0/128	1060.0	124.0/128	0.0	0.0
Scope-only	FP50	128.0/128	1060.0	124.0/128	0.0	0.0
Context-added	FP75	0.0/128	1317.0	121.0/128	4.0	1.0
Context-added	FP50	0.0/128	1763.3	119.0/128	6.0	1.0

Prompt matches counts inputs identical to the paired Reference input. Outcome transitions compare each condition with Reference at the fixed snapshot. Precision is 100%/75%/50% for Exact/FP75/FP50, with 100% recall.

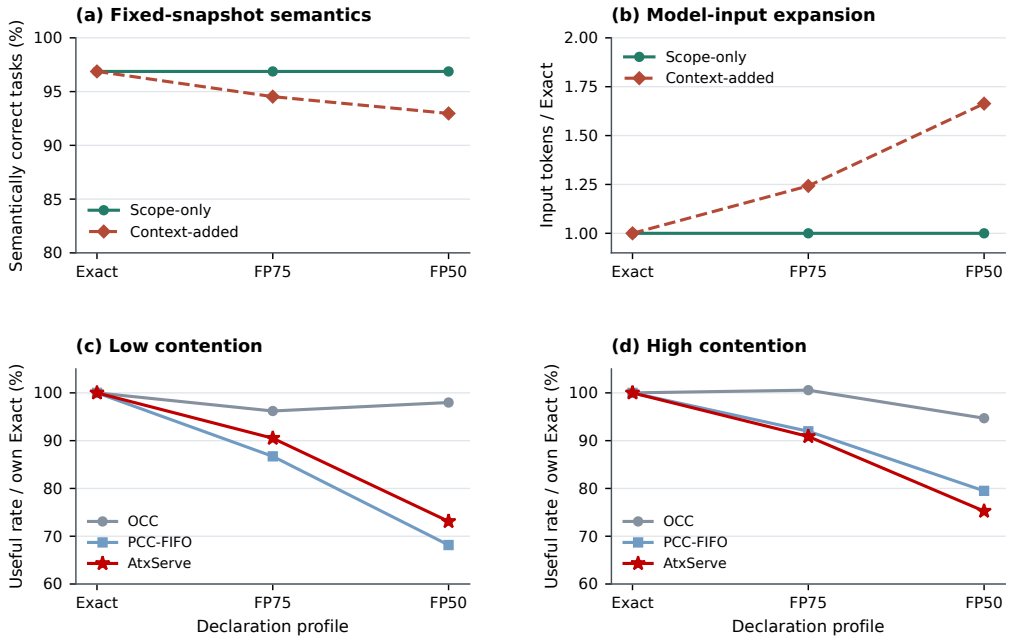


Figure 10: Declaration sensitivity. (a) Semantic correctness at a fixed snapshot. (b) Input tokens relative to Exact. (c,d) Useful rates at Low and High relative to each policy's own Exact rate. Percentages and normalized rates use means over three runs.

Scope-only preserves inputs and all paired outcomes, consistent with Proposition 8. Context-added instead lowers accuracy as inputs expand (Table 20). In concurrent tests, expanded declarations lower active grant occupancy (Table 21). At FP50, useful rates fall by 26.9%/24.8% at Low/High, while all 128 tasks commit. The High FP50 rate is 1.5% above OCC. All tested policies and profiles record zero stale commits and lost updates. These diagnostics isolate declaration expansion at full recall, with input equality evaluated at a fixed snapshot.

Table 21: Concurrent declaration sensitivity. Rates divide mean useful counts by mean elapsed time (ops/hour); all means use three runs.

		Useful rate (ops/hour)			AtxServe diagnostics			
Setting	Profile	OCC	PCC-FIFO	AtxServe	Useful / committed	Slots (%)	Wait P95 (s)	Stale / lost
Low	Exact	6009.7	5637.4	5844.1	123.7/128	63.4	1.2	0/0
Low	FP75	5781.7	4888.5	5289.4	124.3/128	56.6	2.8	0/0
Low	FP50	5888.3	3842.3	4272.4	122.0/128	43.6	5.4	0/0
High	Exact	2175.3	2434.6	2779.6	124.0/128	28.4	4.9	0/0
High	FP75	2187.7	2238.9	2525.9	123.7/128	24.5	8.8	0/0
High	FP50	2059.9	1935.3	2091.6	123.0/128	18.5	12.4	0/0

Slots measures active grant occupancy averaged over time. Wait P95 averages trial percentiles. Stale/lost counts stale committed updates and lost updates; rejected attempts and retries are separate.

F SCALABILITY EVALUATION

F.1 SUSTAINED CONTENTION

Measured entries in Table 22 report means over seeds 7, 17, and 27. Its latency P95 excludes OCC aborts, whereas both graph policies complete all operations. Reservation changes mean throughput by -0.1% in this sustained-load comparison. A separate fixed-arrival test reports waiter progress and overtakes in Appendix F.3.

Table 22: Progress timings (512 operations; three seeds).

Policy	Commits	Abort	Time (s)	TP	Waste (%)	P95 (ms)
OCC	365.0	147.0	719.8	1430.5	73.3	**65394.1**
OCC+Saga	365.0	147.0	736.7	1397.7	73.3	66929.9
CoAgent	511.0	1.0	662.5	2169.7	54.2	152492.2
PCC-FIFO	**512.0**	**0.0**	1176.3	1221.1	**14.7**	79042.6
w/o Reservation	**512.0**	**0.0**	**490.2**	**2915.7**	18.1	110531.3
ATXSERVE	**512.0**	**0.0**	494.2	2913.6	17.6	113759.3

F.2 CONCURRENCY SCALING

The scaling study retains the transaction protocol for 1,024 operations from 8, 16, or 32 agents. Each policy comparison shares a GPU.

Table 23: Concurrency scaling at Medium contention (1,024 operations). n counts measured trials.

Agents	Policy	n	Commits ↑	TP ↑	S/R/A ↓	Waste (%) ↓	Time (s) ↓
8	OCC	3	**1024.0**	5002.9	39.0/48.0/**0.0**	12.0	**583.6**
	OCC Backoff	3	**1024.0**	4954.5	36.0/45.0/**0.0**	11.5	591.5
	PCC-FIFO	3	**1024.0**	4914.8	**0.0/0.0/0.0**	**9.5**	597.7
	ATXSERVE	3	**1024.0**	**5016.5**	**0.0/0.0/0.0**	**9.5**	586.3
	CoAgent	3	1024.0	4734.2	20.3/16.7/0.0	16.5	621.2
	OCC+Saga	3	1024.0	4957.2	39.0/48.0/0.0	12.0	589.0
16	OCC	3	1013.0	4882.4	96.0/114.0/11.0	16.9	600.2
	OCC Backoff	3	1014.0	4840.0	112.0/130.0/10.0	17.5	604.7
	PCC-FIFO	3	**1024.0**	4723.6	**0.0/0.0/0.0**	**9.6**	628.0
	ATXSERVE	3	**1024.0**	**5165.7**	**0.0/0.0/0.0**	**9.6**	**572.9**
	CoAgent	3	1023.3	4743.5	44.0/29.7/0.7	20.8	623.5
	OCC+Saga	3	1013.0	4820.1	96.0/114.0/11.0	16.9	608.0
32	OCC	3	948.0	3748.4	342.0/383.0/76.0	35.1	760.7
	OCC Backoff	3	934.0	3752.9	328.0/369.0/90.0	35.5	754.0
	PCC-FIFO	3	**1024.0**	4454.3	**0.0/0.0/0.0**	**8.0**	685.4
	ATXSERVE	3	**1024.0**	**5272.4**	**0.0/0.0/0.0**	8.2	**579.7**
	CoAgent	3	1023.3	4585.6	165.3/122.7/0.7	28.4	666.0
	OCC+Saga	3	948.0	3681.6	342.0/383.0/76.0	35.1	774.4

Table 24: Concurrency scaling at High contention (1,024 operations). n counts measured trials.

Agents	Policy	n	Commits ↑	TP ↑	S/R/A ↓	Waste (%) ↓	Time (s) ↓
8	OCC	3	1020.0	5233.7	40.0/46.0/4.0	9.9	594.3
	OCC Backoff	3	1022.0	**5276.3**	39.0/44.0/2.0	9.4	592.2
	PCC-FIFO	3	**1024.0**	5058.0	**0.0/0.0/0.0**	**6.2**	619.9
	ATXSERVE	3	**1024.0**	5270.1	**0.0/0.0/0.0**	7.5	**591.6**
	CoAgent	3	1023.7	4945.6	20.7/12.0/0.3	15.0	630.3
	OCC+Saga	3	1020.0	5172.5	40.0/46.0/4.0	9.9	601.3
16	OCC	3	1004.0	5085.5	113.0/130.0/20.0	14.5	631.4
	OCC Backoff	3	1009.0	5074.2	113.0/133.0/15.0	14.2	631.4
	PCC-FIFO	3	**1024.0**	4453.1	**0.0/0.0/0.0**	**5.5**	726.0
	ATXSERVE	3	**1024.0**	**5465.8**	**0.0/0.0/0.0**	5.6	**590.8**
	CoAgent	3	1023.7	5019.4	59.0/43.0/0.3	17.7	643.2
	OCC+Saga	3	1004.0	5010.3	113.0/130.0/20.0	14.5	640.9
32	OCC	3	837.0	2647.5	575.0/620.0/187.0	46.8	996.7
	OCC Backoff	3	806.0	2545.2	552.0/582.0/218.0	47.4	992.9
	PCC-FIFO	3	**1024.0**	3901.3	**0.0/0.0/0.0**	6.6	817.6
	ATXSERVE	3	**1024.0**	**4988.9**	**0.0/0.0/0.0**	**6.5**	**633.6**
	CoAgent	3	1022.7	4104.1	305.3/205.3/1.3	33.1	769.2
	OCC+Saga	3	837.0	2592.8	575.0/620.0/187.0	46.8	1017.7

F.3 ADDITIONAL CONTROLS

Table 25: Mean USEFULTP for backoff validation, with caps in ms.

Base (ms)	Cap 400	Cap 1,600	Cap 6,400
50	2410.3	2406.4	2385.7
100	2372.6	2347.9	2394.9
400	**2515.2**	2383.6	2361.2

Backoff selection maximizes mean validation USEFULTP over Medium/High and seeds 37/47/57 (Table 25). The selected base and cap of 400 ms remain fixed for testing on disjoint seeds.

The reservation stress test offers 180 GPU stages at 0.25 second intervals and observes each policy for 60 seconds. These stages use budgets of 64, 96, or 128 tokens, with conflicting arrivals competing against older waiters while independent branches remain available. Table 26 counts completed stages per second, rather than semantically scored updates. Its wait tails condition on admission by the deadline and average trial statistics; queued and running stages count as unfinished. At $B = 8$, maximum overtakes fall from 97 to 8 at a 10.6% throughput cost, with more unfinished stages and higher wait P95. These overtakes use client arrival and admission order; $B = 4$ also records eight, consistent with the conditional guarantee in Proposition 5. In a descriptive pooled comparison of 45 prespecified stages with concentrated conflicts, $B = 8$ completes 35 versus 23 without reservation.

Table 26: GPU stage stress under fixed arrivals. Throughput counts completed stages/s; overtakes reports the maximum across trials.

Policy	B	Stages/s ↑	Wait P95 (s) ↓	Wait max (s) ↓	Overtakes max ↓	Unfinished ↓
PCC-FIFO	off	1.67	32.6	34.4	**0**	80.0
ATXSERVE	4	2.48	26.2	32.7	8	31.0
ATXSERVE	8	2.53	23.5	34.7	8	28.0
ATXSERVE	16	2.53	25.0	**30.5**	16	28.3
w/o Reservation	off	**2.83**	**12.2**	46.0	97	**10.0**

Table 27 measures grants per nonempty decision, average active grants relative to capacity, and decision CPU cost. These measurements share workload, capacity, collection window, and grant lifetime across policies.

Table 27: Admission instrumentation at Controlled High.

Policy	Admission width ↑	Slot occupancy (%) ↑	Decision CPU P95 (ms) ↓
PCC-FIFO	**1.675**	24.039	0.078
w/o Reservation	1.058	29.781	**0.075**
ATXSERVE	1.064	**29.847**	0.078

G DYNAMIC ACCESS

G.1 PLANNING VALIDATION

We first use a deterministic planner, a CPU executor, and barriers to check the event order in Algorithm 2. Stable execution and replanning share the same client. Each trial resets the workspace and allows one change. The routing trial changes the catalog from A to B after planning, then scores against its state at successful binding. The unrelated change instead lies outside all recorded dependencies, including shared directory versions.

After binding, managed writers wait for conflicting grants; external versioned writers bypass admission and trigger rejection at final validation. Rejected attempts may restart after the writer stops. Regression checks also cover repeated reads, deletion and recreation, lease expiry, cancellation, and exception cleanup. The control omits only planning validation at binding and retains grant coverage, inputs, and final read checks. These diagnostics distinguish stale inference from incorrect commits. Separately, Table 28 measures stable-input overhead with five pairs of 32 tasks, identical inputs, and randomized path order. Capture and binding times are per attempt; elapsed time covers the workload and serving variation.

Table 28: Stable path accounting. Values before/after each slash denote planning validation only at commit/planning validation at binding.

Pair	Planning capture (μs/attempt)	Binding (μs/attempt)	Model calls	Elapsed (s)
1	186.9	209.2/458.0	64/64	294.6/299.7
2	199.4	202.3/416.9	64/64	308.7/307.0
3	169.9	252.5/351.4	64/64	298.1/300.5
4	176.0	195.6/440.4	64/64	316.4/324.0
5	176.0	231.1/369.8	64/64	296.1/299.5

Each run has one planning call and one main call per task, with no dependency changes. Capture records observations as they are obtained; binding validates those records and reads the selected snapshot. Both timers measure protocol work, excluding model inference, on this workload and model.

G.2 SOURCE EDITING

The case comprises four groups of four source-editing tasks in one pinned repository, separate from development tasks. These groups combine repairs, source search, and compatible work under reproducible tests. Search records scanned blocks and directory metadata in H_i during planning. During execution, tools check scope before reading content and require write permission for patches; arbitrary shell access and untracked external inputs are excluded. Denied access releases the grant and discards the attempt, retaining only the resource identifier and required permission. Replanning obtains fresh versions and expands the declaration; earlier committed stages remain intact.

All configurations share Qwen3 4B, tools, validation, and budgets that include planning, retries, and recovery. Serial execution runs one task at a time; OCC Backoff uses delays from 0 to 400 ms without grants. Protected policies share API placement, K, a one millisecond collection window, and $B = 8$. After reservation, PCC-FIFO skips blocked requests in arrival order, whereas ATXSERVE selects by minimum degree. Each group uses seeds 7/17/27, randomized policy order, and independent workspace resets. Thus each policy has 12 episodes and 48 evaluations of 16 distinct tasks, with 48 episodes overall. Pairing fixes initial conditions but permits different declarations and tool traces. Throughput counts task evaluations passing task tests and publication checks per GPU hour, including

planning, failed attempts, recovery, and scoring. Recovery reports tasks with repeated planning and calls beyond the first per stage, normalized per 100 evaluations.

G.3 Compensation Baselines

The CoAgent/MTPO comparison (Lyu et al., 2026) shares the source-editing model, GPU, tasks, tools, budgets, and seeds 7/17/27. The comparisons also cover controlled contention, Commit Focus, the 512-operation sustained-load study, and 1,024-operation concurrency scaling. CoAgent/MTPO preserves ordered reads, conflict notifications, agent-guided repair, and write undo/reordering. OCC+Saga combines optimistic validation with logged inverse actions for already published steps (Garcia-Molina & Salem, 1987), followed by failed-stage replay rather than LLM-guided repair. With one atomic publication, rejected validation leaves nothing to compensate; a multi-step case is needed to exercise compensation itself. Compensated but abandoned operations still count as aborts; only successful recovery reduces terminal aborts, with replay charged to the same budgets. Throughput accounting includes repair costs and uses the same correctness, serial-consistency, and freshness checks. The episode count above covers Serial, OCC Backoff, PCC-FIFO, and ATXSERVE; it excludes the two compensation baselines.