ATOMIX: TIMELY, TRANSACTIONAL TOOL USE FOR RELIABLE AGENTIC WORKFLOWS

Bardia Mohammadi[1], **Nearchos Potamitis**[2], **Lars Klein**[3], **Akhil Arora**[2], **Laurent Bindschaedler**[1]
[1]Max Planck Institute for Software Systems [2]Aarhus University [3]EPFL
bmohammadi@mpi-sws.org, bindsch@mpi-sws.org

ABSTRACT

LLM agents increasingly act on external systems, yet tool effects are immediate. Under failures, speculation, or contention, losing branches can leak unintended side effects with no safe rollback. We introduce Atomix, a runtime that provides *progress-aware transactional semantics* for agent tool calls. Atomix tags each call with an epoch, tracks per-resource frontiers, and commits only when progress predicates indicate safety; bufferable effects can be delayed, while externalized effects are tracked and compensated on abort. Across real workloads with fault injection, transactional retry improves task success, while frontier-gated commit strengthens isolation under speculation and contention.

1 INTRODUCTION

Large Language Model (LLM) agents are increasingly deployed to perform multi-step tasks that modify external state: editing code repositories, updating databases, filling web forms, sending emails, and calling third-party APIs (LangChain, Inc., 2025; CrewAI, 2025; Wu et al., 2023; Anthropic, 2025; OpenAI, 2025; Model Context Protocol a Series of LF Projects, LLC, 2025). As these systems mature, they are moving beyond single-agent sequential execution toward more sophisticated patterns: multiple agents collaborating on shared resources, single agents pursuing multiple plans in parallel to improve success probability, and speculative execution where an agent begins downstream work before upstream results are confirmed. These patterns promise higher throughput and reliability, but they introduce a fundamental problem: *how do you control when tool effects become permanent?*

Current agent frameworks provide orchestration (checkpointing, retries, timeouts) and rely on idempotency keys plus Saga-style compensations for reliability (LangChain, Inc., 2025; Temporal Technologies Inc., 2025; Amazon Web Services, 2025; Garcia-Molina & Salem, 1987). The underlying assumption is that tool effects are *immediate*: they externalize the moment the tool completes, and reliability comes from retrying or compensating after the fact.

These mechanisms work for sequential tasks, but they break once agents speculate or contend on shared state. Three failure modes emerge:

1. *Speculative branches*: parallel plans execute real tool calls; losing branches leave side effects.
2. *Contention*: concurrent agents touch the same resource; updates interleave and corrupt state.
3. *Crash recovery*: after restart, the agent cannot distinguish partial effects from committed work.

We introduce **Atomix**, a runtime that treats tool calls as transactional effects and controls when they become permanent. The approach draws on database transactions (atomicity, isolation) and Saga-style compensation, but the core challenge is different: in speculative, multi-agent workflows, *when* is it safe to commit?

Atomix tags each tool call with an *epoch* (a logical timestamp), tracks per-resource *frontiers* (progress markers indicating the minimum completed work), and commits only when frontiers confirm no earlier work can still arrive. Bufferable effects (changes that can be delayed, such as local file writes) are held until commit; externalized effects (changes that execute immediately, such as API calls) are tracked and compensated on abort via cleanup handlers, yielding progress-aware commit without modifying tools. Figure 1 illustrates this commit gating under speculation.

We evaluate on WebArena, OSWorld, and τ-bench with fault injection, plus microbenchmarks for speculation and contention. We measure task success, contamination, and irreversible-effect leakage,

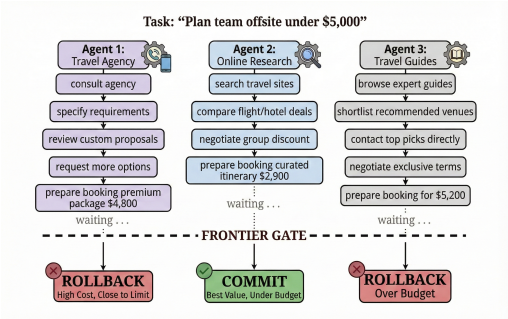

Figure 1: Speculative parallel execution with frontier gating. Three agents pursue different strategies; bufferable effects execute in isolation, and only the winning branch's effects persist.

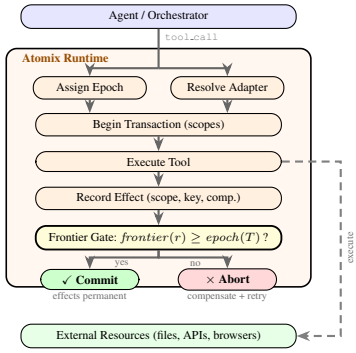

Figure 2: Tool call lifecycle. The runtime interposes between workflow and resources; commits are gated by per-resource frontiers.

and compare Atomix to ablation variants that emulate checkpoint rollback, remove frontier gating, or remove transactions entirely.

Across real workloads with 30% per-call fault injection, Tx-Full (full transactional wrapping with frontier-gated commit and retry) reaches 37—57% task success versus 0—7% for immediate-effect baselines, No-Frontier (compensation without ordering enforcement, no retry) and No-Tx (no Atomix wrapping), while closely matching snapshot rollback (CR, checkpoint-based recovery). The microbenchmarks show where frontiers matter: Atomix eliminates transient contamination on bufferable state, serializes contention, and gates irreversible effects.

Contributions.

1. **A transactional model for tool effects** (§3): epochs, per-resource frontiers, and a commit predicate for safe effect visibility.
2. **A runtime** (§4–5): a shim that interposes on tools and enforces commit/abort with compensation.
3. **Empirical validation** (§6): task success under faults and isolation under speculation/contention.

While the technical machinery draws on classical transaction processing, it is motivated by and designed for the unique characteristics of LLM agent workloads: non-deterministic tool use, speculative parallel reasoning, and external side effects that traditional replay-based systems assume away. The result enables agent capabilities (safe parallel speculation, fault-transparent tool use, multi-agent coordination) that current ML agent frameworks cannot provide.

2 Background and Motivation

State of the art. Agent frameworks and workflow engines provide orchestration: durable graphs, retries, checkpoints, and timeouts (LangChain, Inc., 2025; Temporal Technologies Inc., 2025; Amazon Web Services, 2025). Reliability for side effects typically relies on tool-level idempotency keys (when available) and Saga-style compensations triggered after failures (Garcia-Molina & Salem, 1987). File-system workflows sometimes add git-style isolation (e.g., AgentGit) so changes merge only on success (Li et al., 2025). These mechanisms assume a single agent or linear workflow; effects still externalize immediately, and there is no principled signal that "no earlier work remains." This gap becomes acute once workflows go parallel.

Why this matters. As deployments move to parallel plans, speculation, and multiple agents over shared resources, concurrent agents interleave updates, and crash recovery cannot reliably distinguish partial from committed effects. Without a notion of progress across agents and resources, "retry and compensate" fails to prevent residual writes or transient contamination.

Running example. Consider Best-of-K speculation (Figure 1): three branches book different itineraries in parallel, but only one should persist. One branch may find a cheaper rate while another is over budget; the fastest branch does not necessarily win. Calls must execute to assess prices, yet cancellations are not atomic—losing branches can leave residual effects. The workflow needs a safe commit point.

Design requirements. We need a transactional notion of tool effects that spans multiple calls and agents, plus a progress predicate that tells us when it is safe to commit. The mechanism should buffer effects when possible, compensate when not, and define clear commit/abort boundaries without modifying tools.

3 TRANSACTIONAL ABSTRACTIONS AND SEMANTICS

Atomix instantiates the requirements from §2: a transaction boundary across tool calls and a progress predicate that signals when it is safe to commit. The goal: delay permanence until we know no earlier work can still arrive, while still supporting external effects that must execute immediately. In the Best-of-K travel example (Figure 1), each branch explores a different itinerary in parallel; effects are grouped into a transaction and only the winning branch commits, while losing branches compensate.

3.1 CORE ABSTRACTIONS

Atomix uses four minimal abstractions:

- **Artifacts** carry (epoch, trace_id), where `trace_id` identifies the workflow instance.
- **Effects** describe a tool call's side-effect: its resource scope, an idempotency key (from the tool or adapter), and a compensation handler.
- **Frontiers** encode progress per resource; the orchestrator advances a resource's frontier once it knows no earlier work for that resource remains.
- **Transactions** group effects so they commit or abort together.

Epochs are totally ordered, but visibility is controlled per-resource by frontiers, yielding the progress predicate needed for safe commit.

3.2 COMMIT AND ABORT RULES

The **commit rule** enforces the progress predicate: a transaction commits only when every resource it touches has advanced its frontier to *at least* the transaction's epoch. This provides the precise "no earlier work remains" signal missing in current stacks.

The **abort rule** executes compensations in reverse dependency order. When effects are reversible, compensation undoes them; when they are not, Atomix records the effect for audit and manual intervention. Atomix logs artifacts, effects, commits, and compensations to support replay-based debugging and audit (Dunlap et al., 2002). For bookings, abort can issue a cancellation for reversible holds; when cancellation is only partially effective, it triggers concrete mitigation (e.g., an email or phone cancellation request).

3.3 SAFETY INVARIANTS

Atomix's model targets three safety guarantees aligned with the gap in §2:

Atomicity (by effect class). Bufferable effects are all-or-nothing: they either become visible together at commit or are discarded. Externalized reversible effects are compensated on abort but can leave residue if compensation fails; irreversible effects are gated rather than undone.

Exactly-once effects (model). With idempotency keys and deduplication state, retries do not duplicate effects. When tools lack keys, adapters derive stable keys (e.g., file path + hash). In our prototype, this holds within a process lifetime; crash safety requires durable key storage (future work).

Progress-safe commit. A commit happens only when every touched resource's frontier has reached the transaction's epoch, serializing conflicting agents per resource without global locking. If Agent A (epoch 5) and Agent B (epoch 6) touch the same resource, B cannot commit until A finishes; agents on disjoint resources proceed in parallel. Aborted effects leave no persistent state after compensation; irreversible effects are gated. Effects from losing speculative branches do not remain in shared state after compensation.

These guarantees describe the intended semantics; the current implementation approximates them as described in §5. Atomix prioritizes safety over liveness: partitions or crashes result in stalls rather than inconsistency.

3.4 EFFECT TAXONOMY

Effects vary in reversibility and bufferability. *Reversible effects* (file edits, database writes) are undone via compensation; *reversible-with-cost effects* (refundable purchases with fees, API calls

consuming rate limits) can be mitigated but incur penalties; *irreversible effects* (emails, financial transfers) require gating (Gray, 1978). Orthogonally, *bufferable* effects (local writes) can be held until commit, while *externalized* effects (remote APIs) execute immediately. In the travel example (Figure 1), creating a reservation is reversible, while a confirmation email is irreversible and should be gated. Classification is adapter-defined; missing compensations are logged for review.

4 System Design

Atomix is a runtime shim between orchestrators and tools. The design goal is to control *when* effects become permanent without modifying tools, by exposing a single progress predicate and a transactional boundary that spans multiple calls. This drives the architecture around three collaborating components: a transaction manager, a progress tracker, and tool adapters.

4.1 Architecture

Figure 2 shows the control flow: the transaction manager wraps each tool call (`begin`, `record_effect`, `commit`/`abort`), the progress tracker answers whether a commit is safe, and adapters translate tool I/O into effects with scopes and compensation. This separation lets Atomix interpose on existing workflows while keeping tools unchanged.

4.2 Progress Tracking

The transaction manager queries a single predicate (`can_commit`) to determine when a transaction may safely finalize: a transaction commits only when, for every resource in its scope, the frontier has reached the transaction's epoch. Frontiers are initialized to 0; epochs start at 1. A transaction at epoch e can commit when $\mathit{frontier}(r) \geq e$ for all resources in scope, meaning all tool calls with epoch $< e$ have completed.

Atomix maintains per-resource frontiers, relying on integrations to call `advance_frontier` only after all earlier work on a resource completes: sequential runs advance after each call, speculation waits for all branches, directed acyclic graphs (DAGs) wait for predecessors, and distributed settings use the minimum completed epoch across agents. Premature advancement violates safety; partitions stall frontiers (and thus delay commits) to preserve correctness. Appendix Algorithm 1 provides the full pseudocode for progress tracking. For Best-of-K travel, each provider resource advances only once every branch either commits or aborts its reservation attempt for that provider.

4.3 Tool Adapters

Adapters wrap around tool invocations as effects; tools are invoked unmodified. Each adapter translates tool inputs/outputs to effects, extracts resource scopes, and provides compensation handlers. Resources are represented as hierarchical scopes with wildcards (e.g., `fs:/repo/src/**`, `api:stripe:customer:cust_123`). Atomix includes adapters for filesystem changes, browser automation (WebArena), GUI/OS automation (OSWorld), and structured API tool environments (τ-bench); additional adapters can be implemented following the same pattern.

Integration hooks. We provide thin integration hooks for Claude Code and LangGraph: pre/post tool hooks open transactions and record effects; `ToolNode` wrappers add scopes and commit gating. These integrations illustrate substrate-agnostic interposition; our evaluation uses workload-specific harnesses for controlled fault injection.

Distributed deployment. Atomix currently runs as a single-process library; distributed deployment is not supported. Supporting it would require a shared, linearizable frontier store and a durable, replicated effect log plus a global epoch allocator, which we leave to future work.

5 Implementation

Atomix is implemented as a single-process Python library ($\sim$2k LOC) comprising the transaction manager, frontier tracker, effect model, and replay engine. Effects are logged to JSON Lines or SQLite for replay and metrics.

Buffering. Buffered effects remain invisible to other transactions until commit, preventing cascade aborts: if a transaction later aborts, no other transaction has observed its buffered writes. Externalized effects can be transiently visible, but compensation restores shared persistent state after abort.

Compensation. On abort, compensation handlers execute in reverse dependency order. Failed compensations retry with exponential backoff (up to three attempts); unresolved effects are logged for operator follow-up.

Limitations. The prototype runs in library mode with in-memory (non-crash-safe) deduplication and tool-specific compensation handlers instead of global snapshots. A distributed version would need a shared frontier store and durable effect log, which we leave to future work. Atomix prioritizes safety over liveness: if an agent hangs, the frontier stalls and later transactions on the same resource block. The orchestrator must enforce timeouts and advance frontiers for crashed branches to ensure progress.

6 EVALUATION

We evaluate Atomix's progress-aware commit semantics on reliability and correctness around three research questions:

RQ1: Fault recovery. Does retry recover task success under faults on real workloads? (§6.2)

RQ2: Speculation. Does frontier-gated commit prevent speculative contamination? (§6.3)

RQ3: Correctness. Do transactional boundaries preserve state correctness, and does the effect taxonomy gate irreversible effects? (§6.4)

6.1 METHODOLOGY

6.1.1 MODES UNDER TEST

We aim to compare against existing systems, but none expose the fault-injection hooks or effect-level semantics required by our benchmarks; direct ports would be both costly and potentially biased toward Atomix's design goals. Instead, we implement four modes that represent the guarantees and mechanisms those systems provide, evaluated under the same harness.

All experiments compare four configurations, representing progressively stronger guarantees:

- **Tx-Full**: Full Atomix with transactional wrapping, retries, and frontier-gated commits.
- **CR (Checkpoint-Rollback)**: Snapshot rollback with retry; effects externalize immediately (captures AgentGit's recovery *mechanism* but not its branching/merge features) (Li et al., 2025).
- **No-Frontier**: Transactions and compensation without ordering enforcement; no retry (compensation-only workflows).
- **No-Tx**: Baseline with no Atomix wrapping; tools execute directly.

This design isolates retry and transactional boundaries in RQ1 (Tx-Full/CR vs No-Frontier/No-Tx), and isolates progress-aware commit gating in RQ2/RQ3 (Tx-Full vs CR). We also report a mock ablation with **Tx-NoFrontier+Retry** (transactions + compensation + retry, no frontier waiting) in Appendix Table 7 to isolate retry from frontiers in a controlled setting.

6.1.2 WORKLOADS

We evaluate two categories. *Controlled microbenchmarks* isolate specific properties (speculation isolation, fault recovery, irreversible gating, contention). *Real-world benchmarks* (WebArena, OSWorld, τ-bench) are run under fault injection on real infrastructure. Existing benchmarks do not support fault injection, so we instrument tool calls with a per-call fault profile; details (including the lost-response model) are in Appendix A.2. We report real-infrastructure results in the main text; mock/scaled runs used for sanity checks are in Appendix A.

Controlled microbenchmarks. **Speculative execution**: $K \in \{2, 4, 8, 16\}$ parallel branches touch a shared in-memory store; only one branch should persist. We measure transient and end-state contamination. **Synthetic fault recovery**: a 10-step state-update task with per-step fault probability $fp \in \{0.0, 0.1, 0.3, 0.5\}$ tests whether retry restores correctness. **Irreversible effect gate**: mixed reversible/irreversible actions test whether irreversible effects are correctly gated under faults. **Multi-agent contention**: concurrent agents access shared resources under disjoint and conflicting patterns to measure correctness.

Real-world benchmarks. **WebArena** is a suite of multi-step browser tasks over realistic web apps (Zhou et al., 2023; 2024). We run 30 tasks on real infrastructure at $fp = 0.3$ with GPT-4o (max 30 steps), using the benchmark-recommended model to ensure nontrivial success without faults; success uses the benchmark's eval=1.0 criterion.

OSWorld is a GUI automation benchmark over desktop tasks (Xie et al., 2024b;a). We run 30 tasks at $fp = 0.3$ with Claude 4.0 Sonnet on real VMs, using a recommended model configuration with reasonable failure-free success.

τ-bench is a tool-using retail customer-service benchmark (Yao et al., 2024). We run 30 tasks at $fp = 0.3$ with a GPT-4o agent and simulated user (max 100 steps), following the benchmark's standard setup for stable baseline performance.

We use fp=0.3 as a stress test to ensure sufficient fault events per run ($\sim$9 faults over 30 steps); real-world fault rates are typically 1–10%, but lower rates require larger sample sizes to observe recovery behavior. Appendix Table 11 reports results at $fp \in \{0.05, 0.1, 0.3\}$, showing that Atomix retains a meaningful advantage at lower fault rates. Each configuration is run multiple times; we report 95% confidence intervals from bootstrap resampling.

Important caveat: mock vs. real. Without faults, the LLM completes 100% of mock tasks. The gap between mock (78–100%) and real (37–57%) success reflects LLM-side limitations (stochasticity, state drift), not transaction overhead; mock experiments isolate mechanisms from LLM variability.

6.1.3 METRICS

We measure:

- **Task success rate**: fraction of tasks completing all steps without unrecoverable error.
- **Correct state %**: fraction of runs whose final state matches the expected ground truth.
- **Transient contamination**: fraction of runs where any non-committed branch effect becomes visible at any time.
- **Completion rate**: fraction of expected steps present in the final state.
- **Corruption rate**: fraction of steps with incorrect values in the final state.
- **Tool errors propagated**: number of fault-induced errors surfaced to the agent.

6.2 RQ1: FAULT RECOVERY ON REAL INFRASTRUCTURE

RQ1 asks whether retry plus transactional wrapping improves task success under realistic faults on real agentic workloads. We run 30 tasks each on WebArena (GPT-4o, max 30 steps), OSWorld (Claude 4.0 Sonnet, $\sim$30 steps), and τ-bench (GPT-4o agent+user, max 100 steps) with per-call fault injection at $fp = 0.3$ on real infrastructure. We compare Tx-Full, CR, No-Frontier, and No-Tx; No-Frontier models compensation-only workflows without progress gating, and No-Tx reflects orchestrators (e.g., LangGraph/Temporal) where effects externalize immediately. CR snapshots differ by workload: WebArena captures browser state (Document Object Model, navigation history, form inputs) but not backend server state; OSWorld uses full VM snapshots; τ-bench snapshots the Python environment (globals/modules). Mock and scaled runs are reported in Appendix A.

Mode	WebArena	OSWorld
Tx-Full	**57.2 $\pm$ 6.7**	**37.0 $\pm$ 8.8**
CR	53.2 $\pm$ 4.3	37.1 $\pm$ 5.1
No-Frontier	7.8 $\pm$ 2.0	0.5 $\pm$ 0.5
No-Tx	3.3 $\pm$ 2.4	0.0 $\pm$ 0.0

Table 1: Task success (%) on WebArena/OSWorld, 30% faults. n=30, 95% CI.

Mode	Success (%)	Faults	Retries	Recov.
Tx-Full	**53.5 $\pm$ 7.7**	195	186	110
CR	41.0 $\pm$ 6.9	214	202	125
No-Frontier	7.7 $\pm$ 2.5	68	0	0
No-Tx	0.0 $\pm$ 0.0	63	0	0

Table 2: τ-bench success and fault stats, 30% faults. n=30, 95% CI.

Results. Tables 1–2 show Tx-Full and CR achieve 37–57% success, while immediate-effect baselines collapse to $\leq$8%. Tx-Full improves success 7$\times$ over No-Tx on WebArena (57% vs 8%) and from 0% to 37% on OSWorld; these differences are statistically significant (non-overlapping 95% CIs). Tx-Full and CR show overlapping CIs on WebArena/OSWorld, but on τ-bench Tx-Full outperforms CR (53.5% vs 41%, 12 pp gap with barely-overlapping CIs), suggesting frontiers help

on longer multi-turn tasks. Tx-Full recovers 59% of retried faults (110/186 on τ-bench). A mock ablation (Appendix Table 7) confirms retry+compensation drive fault recovery.

Takeaway. Retry plus compensation drives fault recovery (7× improvement over baselines). Frontiers show marginal benefit on short sequential tasks but help on longer tasks (τ-bench). Immediate-effect baselines fail under faults.

6.3 RQ2: SPECULATION

RQ2 asks whether speculation can run without contaminating shared state from losing branches. We test a bufferable in-memory store with $K \in \{2, 4, 8, 16\}$ parallel branches (200 runs), a speculative WebArena variant (mock, not real infrastructure) forking k=3 branches per step over five shared DOM elements, and a file-write benchmark measuring leaked files from aborted branches.

Results. Tx-Full eliminates transient contamination on the bufferable-store benchmark (0% vs. 100% for baselines). In the speculative WebArena variant (Table 3), success rates overlap (77.3% vs. 77.7%), but Tx-Full halves conflicts (148 vs. 296). The k=1 column shows retry/compensation benefit (+90 pp). For externalized effects, Tx-Full leaks zero files vs. 40 for baselines under k=3 speculation.

Takeaway. Frontier gating does not improve success rates under speculation (CIs overlap), but provides two safety benefits: 50% conflict reduction and **zero** external contamination from aborted branches (0 leaked files vs 40 for baselines).

Mode	Success, k=1 (%)	Success, k=3 (%)	Conflicts
Tx-Full	**92.0 ± 4.5**	77.3 ± 4.8	**148 ± 6**
No-Frontier	2.0 ± 1.4	77.7 ± 5.7	296 ± 11
CR	4.7 ± 2.6	73.7 ± 4.3	305 ± 7

Table 3: Speculative WebArena (mock) with k=3 parallel branches per step. Frontier gating halves conflicts (148 vs. 296–305) without significantly changing success (CIs overlap). k=1 shows retry benefit under faults. $n = 300$ trials, 95% CI.

6.4 RQ3: CORRECTNESS

6.4.1 CONTENTION

RQ3 asks whether transactional boundaries preserve correctness under multi-agent contention. We evaluate four contention scenarios with 2–8 concurrent agents, each performing 50 operations (100 runs per configuration): disjoint keys, shared counters, read-modify-write, and mixed read/write. To isolate coordination overhead, we measure throughput on no-op operations (no actual tool calls) at 16 concurrent transactions. Detailed breakdowns are in Appendix Table 9.

Results. Tx-Full achieves 100% correctness on all scenarios (200/200 correct final states across configurations), while No-Frontier and No-Tx produce 0% correct outcomes due to lost updates and write-write conflicts. Coordination throughput varies: No-Tx reaches 1,182,535 tx/s (no coordination), No-Frontier 13,575 tx/s, and Tx-Full 14,209 tx/s. The 83× gap reflects serialization cost, but No-Tx produces incorrect results. With real tool latencies (100 ms per call), maximum end-to-end throughput is ∼160 tx/s regardless of coordination layer, so the 14k tx/s coordination capacity is never a bottleneck.

Takeaway. Frontiers serialize conflicting agents and prevent shared-state corruption; coordination overhead is negligible relative to tool latency.

6.4.2 IRREVERSIBLE EFFECTS

Irreversible actions must be gated to prevent permanent damage under faults. A mixed task includes reversible file writes and two irreversible email sends per run. We run 600 runs per mode (100 runs × 3 fault probabilities × 2 confirm settings) and measure leaked, gated, and confirmed emails; Appendix Table 10 reports results.

Results. Tx-Full leaks **zero** irreversible effects (0 of 1,200 emails), while No-Tx leaks all 1,200 and CR leaks 1,351 due to retry-induced duplicates (Appendix Table 10).

Takeaway. Transactional gating is required for irreversible effects; retry without buffering worsens leakage. Tx-Full: zero leakage across all benchmarks.

6.5 SUMMARY

Retry plus compensation gives a 7× gain over immediate-effect baselines on real workloads (RQ1). On short sequential tasks, Tx-Full and CR have similar success (overlapping CIs), but Tx-Full beats CR on longer tasks (τ-bench: 53.5% vs 41%). With speculation, frontier gating halves conflicts and prevents external contamination (RQ2). For irreversible effects, Tx-Full leaks no emails versus 1,200+ for baselines (RQ3). Coordination overhead is negligible ($< 0.01\%$ of tool latency).

Evaluation limitations.

- **Prototype scope**: deduplication is in-memory (not crash-safe); distributed deployment would require a shared frontier store that can stall under partitions.
- **Benchmark coverage**: real workloads are largely sequential; speculation results use mock/variant benchmarks; tools without compensation handlers may still leak.
- **Baselines**: we do not include dedicated per-resource locking or retry-without-frontiers ablations.

7 POSITIONING AND RELATED WORK

Transaction foundations. Atomix builds on classical transactions (ACID, write-ahead logging, ARIES) and Sagas (Gray, 1978; Mohan et al., 1992; Garcia-Molina & Salem, 1987), but commits based on progress predicates rather than lock release—critical under LLM-driven non-determinism.

Orchestration stacks. Agent runtimes (LangGraph, CrewAI, AutoGen) and workflow engines (Temporal, Cadence, Step Functions, Airflow, Durable Functions) (LangChain, Inc., 2025; CrewAI, 2025; Wu et al., 2023; Temporal Technologies Inc., 2025; Cadence a Series of LF Projects, LLC, 2025; Amazon Web Services, 2025; The Apache Software Foundation, 2025; Burckhardt et al., 2021) offer retries and compensations, but effects are immediate with no policy to delay visibility based on progress; Atomix supplies that policy as a shim. We compare *mechanisms* via ablations rather than head-to-head benchmarks (§6.1).

Systems-level contrasts. *AgentGit* (Li et al., 2025) versions workflow state with git-style branching and rollback, but provides no commit gate for external effects—rollback is reactive and speculative effects can leak (CR mode captures this). *AIOS* (Mei et al., 2024) focuses on internal snapshots, not external-effect semantics. *Speculative Actions* (Ye et al., 2025) predicts likely actions for latency; Atomix targets correctness via frontier-gated commit. See Appendix Table 13.

Progress signals in dataflow. Streaming systems use watermarks/frontiers and checkpoint barriers to determine when earlier work can no longer arrive (Murray et al., 2013; Akidau et al., 2021; Apache Flink, 2026; Chandy & Lamport, 1985). Atomix adapts these progress predicates to heterogeneous tool effects with adapter-defined scopes and compensations.

Complementary work. Sherlock (Ro et al., 2025) combines speculative execution with selective verification; AsyncLM (Gim et al., 2024) targets asynchronous function calls for parallelism. Transactional LLM planning and record–replay systems (Chang & Geng, 2025; Feng et al., 2025) target planning or reuse; serving stacks (Kwon et al., 2023; Zheng et al., 2023) target throughput. Atomix is complementary, focusing on when effects become visible via progress predicates.

8 CONCLUSION

We presented Atomix, a transactional runtime that interposes on agent tool calls and commits effects only when progress predicates signal safety. The key contribution is frontier-gated commit with compensation, which provides transactional semantics for external effects without modifying existing tools or orchestrators. Across real workloads at fp=0.3, Atomix improves completion from 0–7% for immediate-effect baselines to 37–57% (matching snapshot rollback); transactional retry enables fault recovery, and progress-aware commit enables safe speculation, contention, and irreversible-effect gating as shown in our microbenchmarks.

REFERENCES

Tyler Akidau, Edmon Begoli, Slava Chernyak, Fabian Hueske, Kathryn Knight, Kenneth Knowles, Daniel Mills, and Dan Sotolongo. Watermarks in stream processing systems: Semantics and comparative analysis of apache flink and google cloud dataflow. *Proceedings of the VLDB Endowment*, 14(12):3135–3147, 2021. doi: 10.14778/3476311.3476389. URL https://www.vldb.org/pvldb/vol14/p3135-begoli.pdf.

Amazon Web Services. What is step functions? - aws step functions. https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html, 2025. Accessed 2025-11-30.

Anthropic. Claude code overview - claude code docs. https://code.claude.com/docs/en/overview, 2025. Accessed 2025-11-30.

Apache Flink. Stateful stream processing. https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/concepts/stateful-stream-processing/, 2026. Accessed 2026-01-28.

Sebastian Burckhardt, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, and Christopher S. Meiklejohn. Serverless workflows with durable functions and netherite. *arXiv preprint arXiv:2103.00033*, 2021. doi: 10.48550/arXiv.2103.00033. URL https://arxiv.org/abs/2103.00033.

Cadence a Series of LF Projects, LLC. Cadence workflow. https://cadenceworkflow.io/, 2025. Accessed 2025-11-30.

K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985. doi: 10.1145/214451.214456. URL https://doi.org/10.1145/214451.214456.

Edward Y. Chang and Longling Geng. SagaLLM: Context management, validation, and transaction guarantees for multi-agent LLM planning. *arXiv preprint arXiv:2503.11951*, 2025. doi: 10.48550/arXiv.2503.11951. URL https://arxiv.org/abs/2503.11951.

CrewAI. Crewai documentation. https://docs.crewai.com/, 2025. Accessed 2025-11-30.

George W Dunlap, Samuel T King, Sukru Cinar, Murtaza A Basrai, and Peter M Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating Systems Review*, 36(SI):211–224, 2002.

Erhu Feng, Wenbo Zhou, Zibin Liu, Le Chen, Yunpeng Dong, Cheng Zhang, Yisheng Zhao, Dong Du, Zhichao Hua, Yubin Xia, and Haibo Chen. Get experience from practice: LLM agents with record & replay. *arXiv preprint arXiv:2505.17716*, 2025. doi: 10.48550/arXiv.2505.17716. URL https://arxiv.org/abs/2505.17716.

Hector Garcia-Molina and Kenneth Salem. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data (SIGMOD '87)*, pp. 249–259. Association for Computing Machinery, 1987. doi: 10.1145/38713.38742. URL https://dl.acm.org/doi/10.1145/38713.38742.

In Gim, Seung-seob Lee, and Lin Zhong. AsyncLM: Asynchronous LLM function calling. *arXiv preprint arXiv:2412.07017*, 2024. doi: 10.48550/arXiv.2412.07017. URL https://arxiv.org/abs/2412.07017.

Jim Gray. Notes on data base operating systems. In Michael J. Flynn, Jim Gray, Anita K. Jones, Klaus Lagally, Holger Opderbeck, Gerald J. Popek, Brian Randell, Jerome H. Saltzer, and Hans-Rüdiger Wiehle (eds.), *Operating Systems, An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, pp. 393–481. Springer, 1978. ISBN 3-540-08755-9. doi: 10.1007/3-540-08755-9_9. URL https://doi.org/10.1007/3-540-08755-9_9.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Yu, Joey Gonzalez, Hao Zhang, and Ion Stoica. vLLM: Easy, fast, and cheap LLM serving with PagedAttention. https://blog.vllm.ai/2023/06/20/vllm.html, 2023. Published 2023-06-20; Kwon and Li indicated as equal contribution on the post; Accessed 2025-11-30.

LangChain, Inc. LangGraph overview. https://docs.langchain.com/oss/python/langgraph/overview, 2025. Accessed 2025-11-30.

Yang Li, Siqi Ping, Xiyu Chen, Xiaojian Qi, Zigan Wang, Ye Luo, and Xiaowei Zhang. AgentGit: A version control framework for reliable and scalable LLM-powered multi-agent systems. *arXiv preprint arXiv:2511.00628*, 2025. doi: 10.48550/arXiv.2511.00628. URL https://arxiv.org/abs/2511.00628.

Kai Mei, Xi Zhu, Wujiang Xu, Wenyue Hua, Mingyu Jin, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. AIOS: LLM agent operating system. *arXiv preprint arXiv:2403.16971*, 2024. doi: 10.48550/arXiv.2403.16971. URL https://arxiv.org/abs/2403.16971.

Model Context Protocol a Series of LF Projects, LLC. Specification - model context protocol. https://modelcontextprotocol.io/specification/2025-03-26, 2025. Version 2025-03-26; Accessed 2025-11-30.

C. Mohan, Don Haderle, Bruce G. Lindsay, Hamid Pirahesh, and Peter M. Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1):94–162, 1992. doi: 10.1145/128765.128770. URL https://dl.acm.org/doi/10.1145/128765.128770.

Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: A timely dataflow system. In Michael Kaminsky and Mike Dahlin (eds.), *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP 2013, Farmington, Pennsylvania, USA, November 3-6, 2013*, pp. 439–455. ACM, 2013. doi: 10.1145/2517349.2522738. URL https://doi.org/10.1145/2517349.2522738.

OpenAI. Function calling — openai api. https://platform.openai.com/docs/guides/function-calling, 2025. Accessed 2025-11-30.

Yeonju Ro, Haoran Qiu, Íñigo Goiri, Rodrigo Fonseca, Ricardo Bianchini, Aditya Akella, Zhangyang Wang, Mattan Erez, and Esha Choukse. Sherlock: Reliable and efficient agentic workflow execution. *arXiv preprint arXiv:2511.00330*, 2025. doi: 10.48550/arXiv.2511.00330. URL https://arxiv.org/abs/2511.00330.

Temporal Technologies Inc. Temporal docs — temporal platform documentation. https://docs.temporal.io/, 2025. Accessed 2025-11-30.

The Apache Software Foundation. Documentation — apache airflow. https://airflow.apache.org/docs/, 2025. Accessed 2025-11-30.

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. *arXiv preprint arXiv:2308.08155*, 2023. doi: 10.48550/arXiv.2308.08155. URL https://arxiv.org/abs/2308.08155.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024a. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html. Project website: https://os-world.github.io/. Accessed 2025-11-30.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024b. doi: 10.48550/arXiv.2404.07972. URL https://arxiv.org/abs/2404.07972.

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. doi: 10.48550/arXiv.2406.12045. URL https://arxiv.org/abs/2406.12045.

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative actions: A lossless framework for faster agentic systems. *arXiv preprint arXiv:2510.04371*, 2025. doi: 10.48550/arXiv.2510.04371. URL https://arxiv.org/abs/2510.04371. Submitted on 5 Oct 2025.

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. *arXiv preprint arXiv:2312.07104*, 2023. doi: 10.48550/arXiv.2312.07104. URL https://arxiv.org/abs/2312.07104.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. doi: 10.48550/arXiv.2307.13854. URL https://arxiv.org/abs/2307.13854.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, Vienna, Austria, May 2024. URL https://arxiv.org/abs/2307.13854. Project website: https://webarena.dev/. Accessed 2025-11-30.

APPENDIX

A ADDITIONAL EVALUATION DETAILS

A.1 PROGRESS TRACKING ALGORITHM

Algorithm 1 Progress tracking and commit predicate

```
State: frontier[r] ← 0 for each resource r

function CanCommit(tx):
for all r ∈ tx.scope do
    if frontier[r] < tx.epoch then
        return false {earlier work incomplete}
    end if
end for
return true

procedure AdvanceFrontier(r, e):
    frontier[r] ← max(frontier[r], e)
```

A.2 FAULT INJECTION DETAILS

Faults are injected per tool call using a Bernoulli profile with probability fp. An injected fault raises an exception at the tool boundary. In Tx-Full and CR, the runtime aborts the transaction and retries with a fresh epoch. In No-Frontier, the exception propagates after abort (no retry). In No-Tx, we model a *lost response*: the action may have already been applied, but the caller sees an error, reflecting realistic timeouts or dropped acknowledgements. This makes baseline comparisons fairer by allowing side effects to occur even when the agent cannot confirm them.

A.3 MOCK ENVIRONMENTS

Table 4 summarizes task success rates on mock environments across fault probabilities.

Results match theoretical expectations. At $fp = 0.3$ with 10-step tasks, Tx-Full's per-step failure probability is $0.3^4 = 0.81\%$; over 10 steps, expected task failure is $\sim 7.8\%$. Observed: 8% (WebArena). No-Frontier/No-Tx match $(1-fp)^{10} = 2.8\%$.

	$fp = 0.1$				$fp = 0.3$				$fp = 0.5$			
Workload	Tx	CR	No-Fr	No-Tx	Tx	CR	No-Fr	No-Tx	Tx	CR	No-Fr	No-Tx
WebArena	**100**	36	26	26	**92**	2	2	6	**58**	0	0	0
OSWorld	**100**	100	23	30	**100**	100	1	2	**100**	100	0	0

Table 4: Mock environment success rates (%). Tx = Tx-Full; CR = Checkpoint-Rollback; No-Fr = No-Frontier.

A.4 SCALED MOCK EXPERIMENTS

Table 5 shows results with 10 tasks per workload at $fp = 0.3$, providing cleaner separation.

Workload	Steps	n	Tx-Full	No-Frontier	No-Tx
τ-bench	~ 50	10	**100**	0	0
OSWorld	14	9	**78**	0	0
WebArena	12	10	**100**	0	0

Table 5: Scaled mock experiments, success rates (%) at $fp = 0.3$

At scale, separation is absolute: Tx-Full achieves 78–100% success while both baselines achieve 0%. With 12–50 steps at $fp = 0.3$, the probability of completing without any fault is $(0.7)^{12} = 1.4\%$ to $(0.7)^{50} \approx 0\%$.

A.5 Speculation Commit Latency

Table 6 reports wall-clock latency from branch creation to winner commit. Commit latency scales linearly with K: aborting $K-1$ losers dominates at $\sim$60 μs per branch.

Branches (K)	Setup (μs)	Commit (μs)	Total / p99 (μs)
2	13	85	97 / 295
4	23	142	165 / 273
8	44	321	365 / 1339
16	86	939	1025 / 23198

Table 6: Speculation commit latency ($n = 200$ runs per K)

At $K = 16$, total frontier-gating overhead is $\sim$1 ms. Real speculative branches execute tool calls taking 100 ms–10 s, so the commit decision adds $<$1% overhead. The p99 spike at $K = 16$ reflects garbage-collection pauses in CPython, not algorithmic cost.

A.6 Synthetic Fault Recovery

10-step sequential writes, 100 runs per (fp, mode). Tx-Full: **100% correct state at all fault rates** (0.0–0.5). CR: also 100% correct; checkpoint-rollback recovers sequential faults through whole-state restore. No-Frontier: 46%→5%→0% as fp increases (no corruption, but lost steps). No-Tx: 39%→3%→0% with 30% corruption at $fp = 0.3$. On sequential tasks, CR and Tx-Full are equivalent for fault recovery; the distinction emerges under speculation and irreversible effects (below).

A.7 Frontier vs Retry Ablation (Mock)

Table 7 isolates the contribution of retry from frontier gating. Tx-NoFrontier+Retry adds compensation and retry but omits frontier waiting. On this 10-step mock workload, retry alone recovers most faults; Tx-NoFrontier+Retry matches Tx-Full, confirming that frontier gating provides isolation (RQ2/RQ3) rather than fault recovery.

Mode	Success (%)	Total Faults
Tx-Full	90	121
Tx-NoFrontier+Retry	**93**	108
No-Frontier	0	30
No-Tx	3	29
CR	0	30

Table 7: Frontier vs. retry ablation (mock, $n = 30$ tasks, 10 steps, $fp = 0.3$)

A.8 No-Fault WebArena Sanity Check

Table 8 verifies that all modes succeed under zero faults, confirming that performance differences stem from fault handling rather than baseline capability. At $fp = 0.3$, only Tx-Full maintains 100% success on this 6-task subset.

A.9 Additional Real-Workload Tables

A.9.1 Multi-Agent Contention

Table 9 breaks down correctness by contention scenario. Tx-Full achieves 100% correct final states across all shared-resource scenarios, while No-Frontier and No-Tx fail on any scenario involving concurrent writes to shared state. Coordination throughput shows the 83$\times$ gap between No-Tx

Mode	$fp = 0$ (%)	$fp = 0.3$ (%)
Tx-Full	100	100
No-Frontier	100	67
No-Tx	100	50

Table 8: WebArena no-fault sanity check, success rates ($n = 6$ tasks)

(no serialization) and frontier-based modes; with real tool latencies ($\sim$100 ms), this overhead is negligible.

Scenario	Agents	Tx-Full	No-Fr	No-Tx
Correct final states (%):				
Disjoint writes	2/4/8	**100**	100	100
Counter (shared)	2	**100**	0	0
Counter (shared)	4	**100**	0	0
Counter (shared)	8	**100**	0	0
Read-write conflict	2	**100**	0	0
Write-write conflict	2	**100**	0	0
Coordination throughput (tx/s):				
No-op transactions	16	14,209	13,575	1,182,535

Table 9: Multi-agent contention ($n = 100$ runs, 50 ops/agent). No-Fr = No-Frontier.

A.9.2 Irreversible Effect Gate

Table 10 reports email counts for tasks mixing reversible file writes with irreversible email sends. Tx-Full leaks zero emails from aborted transactions; all 600 gated emails are confirmed only after commit. CR exceeds the 1,200-email baseline because retry without buffering re-sends emails, producing 151 duplicates.

Mode	Leaked	Gated	Confirmed	Duplicates
Tx-Full	**0**	600	600	0
No-Frontier	0	519	517	0
No-Tx	1,200	0	0	151
CR	1,351	0	0	151

Table 10: Irreversible effect leakage, email counts ($n = 600$ runs, 1,200 expected)

A.9.3 Multi-Rate Experiments

Table 11 sweeps fault probability across real infrastructure. Tx-Full retains a meaningful advantage even at lower fault rates ($fp = 0.05$), confirming that the benefit is not an artifact of the $fp = 0.3$ stress test used in the main evaluation.

A.9.4 Retry Budget Sensitivity

Table 12 sweeps max_retries on a mock workload. The largest gain comes from $0 \rightarrow 1$ retry (+90 pp for Tx-Full); diminishing returns set in after 3 retries. This suggests max_retries=3 is a reasonable default.

A.10 Robustness and Overhead

Multi-rate experiments. Sweeping $fp \in \{0.05, 0.1, 0.3\}$ (Table 11), protection remains necessary even at low fp; the need for progress-aware commit is not an artifact of high fault rates.

Retry budget. Sweeping max_retries $\in \{0, 1, 2, 3, 5+\}$ (Table 12), the largest gain comes from $0 \rightarrow 1$ retry with diminishing returns after 3; max_retries=3 is a reasonable default.

fp	OSWorld ($n = 7$)			WebArena ($n = 10$)		
	Tx	No-Fr	No-Tx	Tx	No-Fr	No-Tx
0.05	**71**	14	0	**60**	10	0
0.10	**71**	0	0	**50**	0	10
0.30	14	0	0	**60**	0	0

Table 11: Multi-rate results, success rates (%) on real infrastructure. No-Fr = No-Frontier.

Max Retries	Tx-Full (%)	No-Frontier (%)
0	5	5
1	95	80
2	97	82
3	98	83
5+	99	83

Table 12: Retry budget sensitivity, success rates (mock, $n = 30$ tasks, $fp = 0.3$)

Runtime overhead. Tx-Full adds 7.7 μs per step versus 0.8 μs for No-Tx (1000 runs, fault-free). Given tool latencies of 100 ms–10 s, overhead is $< 0.01\%$ of wall-clock time.

A.11 Key Findings (Supplementary)

- **Fault recovery at scale.** On real workloads at $fp = 0.3$, Tx-Full improves task success from 0–7% for immediate-effect baselines to 37–57%, matching CR on success while preserving stronger safety (Tables 1–2).
- **Speculation isolation.** Tx-Full eliminates transient contamination in the bufferable-store benchmark and reduces conflicts in a speculative WebArena variant (Table 3).
- **Contention correctness.** Per-resource frontiers serialize conflicting agents, yielding 100% correctness on shared-resource scenarios with similar throughput to No-Frontier (Table 9).
- **Irreversible gating.** Tx-Full leaks zero irreversible effects; retry without buffered commit increases leakage (Table 10).
- **Negligible overhead.** Transactional bookkeeping adds microseconds per step, $< 0.01\%$ of tool latency (Table 6).

B Additional Related-Work Details

Table 13 summarizes transactional effect capabilities based on published documentation (not empirically verified on shared benchmarks). • denotes full support, ○ partial support, and – absent. *Partial* means: AIOS compensation uses internal snapshots rather than external-effect undo; AIOS durable execution is snapshot-based without write-ahead logging; LangGraph compensation is checkpoint rollback rather than per-effect handlers.

System	Epochs	Progress Gate	Buffered Commit	Compensation	Spec. Isolation	Irrev. Gate	Durable Exec.
Atomix	●	●	●	●	●	●	–
AgentGit	–	–	–	●	–	–	–
AIOS	–	–	–	○	–	–	○
LangGraph	–	–	–	○	–	–	●
Temporal	–	–	–	●	–	–	●

Table 13: Transactional effect capabilities (documentation-based). ● = full, ○ = partial, – = absent.

C ATOMIX API REFERENCE

This appendix provides detailed specifications for the Atomix API, including message schemas, error handling, and security considerations.

C.1 MESSAGE SCHEMAS

Effect log entry. Effect entries stored in JSONL or SQLite follow the structure below:

```
{
  "tx_id": "uuid",
  "trace_id": "string",
  "branch_id": "string",
  "epoch": 12,
  "status": "committed|aborted",
  "scopes": ["fs:/path/to/file"],
  "payload": { /* adapter-specific */ },
  "idempotency_key": "string",
  "description": "write:/path@12"
}
```

Compensation handler. Compensations are Python callables supplied by adapters and invoked in reverse order when a transaction aborts.

Resource scope. Scopes follow a hierarchical naming convention:

```
<type>:<path>[:<identifier>]

Examples:
  fs:/home/user/project/src/** # filesystem subtree
  api:stripe:customers:cust_123 # specific API resource
  db:postgres:mydb:users:* # all rows in table
  email:outbox:user@example.com # user's outgoing mail
```

C.2 ERROR TAXONOMY

The transaction aborts if a tool raises an exception or if an injected fault causes it to fail, which in turn triggers compensations if needed.

C.3 RETRY SEMANTICS

Effect application uses an in-memory idempotency set. If an effect's idempotency key has already been applied, the apply step is skipped.

C.4 SECURITY AND AUTHENTICATION

Security and authentication are not implemented. Scope enforcement and credential management are left as future work, treated as deployment concerns.