The Temporal Coherence Problem: Synthetic Point-in-Time Environments for Evaluating LLM Agents with Dynamic Tool Dependencies

Danish N. Shaikh
Machine Learning Engineer, California, USA
MSc, California State University Chico

ABSTRACT
Large Language Model (LLM) agents increasingly orchestrate multiple external tools—including APIs, code functions, Model Context Protocol (MCP) servers, plugins, and sub-agents—to accomplish complex objectives. Evaluating these agents requires temporally coherent data across all tool dependencies, yet production environments feature independently versioned tools, data retention policies, and evolving sub-agent reasoning that make reproducible evaluation fundamentally difficult. Existing agent benchmarks do not face these issues, as they provide static, self-contained environments, leaving a critical gap between benchmark evaluation and production reliability. This paper makes three contributions. First, it introduces a dependency type spectrum classifying agent tool dependencies from stateless APIs to LLM-based sub-agents by their drift characteristics and snapshot fidelity, formalizing the qualitative difference between data drift and reasoning drift. Second, it presents a taxonomy of four temporal challenges—tool drift, temporal incoherence, forward-looking data gaps, and privacy-constrained reproducibility—with a formal analysis of why standard inference-time logging is insufficient for agent evaluation. Third, it proposes design patterns for synthetic point-in-time snapshot generation and validates them experimentally using a simulated incident root-cause analysis agent, demonstrating that temporal incoherence reduces diagnostic accuracy from 100% to 40% and that synthetic snapshot restoration recovers it to 80%.

General Terms
Algorithms, Performance, Reliability, Experimentation, Verification.

Keywords
Evaluation, LLM Agents, Point-in-Time Data, Sub-Agent Reasoning, Synthetic Data, Temporal Coherence, Tool Dependencies.

1. INTRODUCTION
Large Language Model (LLM) agents have evolved beyond single-turn question answering into autonomous systems that orchestrate multiple external tools to accomplish complex, multi-step objectives. Modern agent frameworks such as LangChain, CrewAI, and AutoGen enable agents to invoke APIs, query databases, execute code, and even delegate subtasks to other LLM-based sub-agents. Industry adoption has accelerated rapidly, with surveys indicating that over 50% of AI professionals now deploy agents in production settings [1].

Evaluating these agents presents a fundamentally different challenge than evaluating standalone language models. Where model evaluation requires only a fixed input-output mapping, agent evaluation requires an entire environment of tool responses that must be internally consistent. A growing body of agent benchmarks has emerged to address this need. However, these benchmarks uniformly rely on static, self-contained environments where tool responses are pre-scripted or run against frozen datasets.

In production, agents operate across a heterogeneous ecosystem of dependencies that are independently versioned, owned by different teams, and subject to data retention and privacy policies. The tools an agent calls today may return different schemas, different data, or fundamentally different reasoning tomorrow. When practitioners attempt to build offline evaluation datasets, they encounter a problem that the benchmark literature does not address: temporal coherence across independently evolving data sources.

A natural response to this challenge is to apply the standard machine learning approach: log all inputs and outputs at inference time and use the logs for offline evaluation. This paper shows that while such logging supports regression testing, it is fundamentally insufficient for evaluating agent changes, owing to the path-dependent nature of agent execution.

This paper makes three contributions: (1) a dependency type spectrum classifying agent dependencies from stateless APIs to LLM-based sub-agents by their drift characteristics and snapshot fidelity; (2) a problem taxonomy of four temporal challenges with a formal analysis of why inference-time logging is insufficient; and (3) design patterns for synthetic point-in-time snapshot generation, validated experimentally on a simulated incident root-cause analysis (RCA) agent.

2. LITERATURE REVIEW
2.1 Agent Evaluation Benchmarks
The period 2023–2026 has seen rapid proliferation of LLM agent benchmarks. A recent survey by Mohammadi et al. [2] catalogs more than two dozen distinct evaluation frameworks targeting different aspects of agent capability. Among the most influential, AgentBench [3] established the first multi-environment evaluation paradigm with eight interactive settings testing 29 LLMs, finding that poor long-term reasoning and decision-making are the primary failure modes. WebArena [4] introduced a self-hosted web environment with 812 tasks where GPT-4 agents achieved only 14.41% success versus 78.24% for humans. τ-bench [5] introduced the pass^k reliability metric, measuring whether agents succeed across k independent trials, revealing that even GPT-4o achieves less than 25% pass^8 in retail domains. SWE-bench [6] evaluates real GitHub issue resolution with repository state pinned to specific commits, and AgentBoard [7] introduced progress rate alongside success rate for more granular evaluation.

A critical observation unifies these benchmarks: every one controls the evaluation environment by freezing it. Tool responses are pre-scripted, web environments are self-hosted

snapshots, and code repositories are pinned to specific commits. This freezing is precisely what production teams cannot do when their agents depend on multiple independently evolving external services. This benchmark-versus-production gap has begun to draw attention from industry practitioners. Engineering teams at Anthropic [8] and Amazon Web Services [9] have published guidance acknowledging that production agent evaluation requires methodologies different from benchmark-driven approaches, particularly when agents depend on services that change between evaluation runs.

2.2 Reliability and Failure Analysis

ReliabilityBench [10] applies chaos engineering principles to agent evaluation, testing behavior under network failures and API errors, but does not address temporal data consistency. Cemri et al. [11] provide the most comprehensive empirical failure taxonomy for multi-agent systems, identifying 14 distinct failure modes in three categories, including inter-agent misalignment. Complementing this academic taxonomy, the Microsoft AI Red Team [12] catalogs failure modes observed in production agentic systems, with categories spanning unintended tool invocation, prompt injection across agent boundaries, and capability degradation under load. Neither work connects these failures to temporal inconsistency across tools, the focus of the present paper. Kapoor et al. [13] further demonstrate that flawed benchmarking practices lead to incorrect conclusions about agent capabilities, highlighting the need for more realistic evaluation.

2.3 The Offline-Online Gap

The disconnect between offline evaluation and online performance is well-characterized in recommendation systems. Castells et al. [14] established that the correlation between offline and online evaluation outcomes is often weak, attributing this to data delivery bias and feedback loop absence. This parallel is directly relevant to LLM agent evaluation, where static benchmarks cannot capture the dynamic, multi-source nature of production tool dependencies. Synthetic data generation for ML evaluation is a growing field, but existing work focuses on tabular data privacy [15] rather than generating temporally coherent multi-source environments for agent workflows.

Gap Statement: No existing work addresses the specific challenge of generating synthetic, temporally coherent, multi-dependency evaluation environments for LLM agents operating across independently evolving data sources with heterogeneous drift characteristics.

3. PROBLEM TAXONOMY

3.1 Formal Problem Setup

The evaluation scenario is defined as follows. An Agent A receives a task T and can invoke a set of dependencies $\{D_1, D_2, ..., D_n\}$. Each dependency D_i has both a data component $d_i(t)$ and a logic component $l_i(t)$, both functions of time. A correct evaluation requires all dependency responses to reflect the same logical timestamp t_0. The agent's output is evaluated against a ground truth $G(t_0)$ that is also time-dependent.

3.2 The Dependency Type Spectrum

Not all agent dependencies are equal. The nature of what can change and how capturable that change is varies fundamentally across dependency types. This paper proposes a four-level spectrum that, to the author's knowledge, has not been previously formalized.

Level 1: Stateless APIs. Examples include metrics query APIs, database lookups, and REST endpoints. What drifts is data values and schema format. Snapshot fidelity is high because what is captured is exactly what the agent saw. The limitation is that a snapshot only captures one query path.

Level 2: MCP Servers and Stateful Services. Examples include Model Context Protocol [16] servers, search indices, and feature stores. What drifts is data, schema, and available capabilities. A new tool function may appear or be removed, changing the agent's decision space. Snapshot fidelity is high if the tool manifest is also recorded.

Level 3: Plugins and Skills. Examples include code execution plugins, retrieval-augmented skills, and rule-based processing modules. What drifts is internal logic, rule sets, and retrieval corpora. The same input may produce different output after a logic update with no visible schema change. Snapshot approach is version-pinning, yielding medium fidelity.

Level 4: Sub-Agents (LLM-based). Examples include a log analysis agent with its own LLM and Retrieval Augmented Generation (RAG) pipeline, or a knowledge agent with its own retrieval corpus. What drifts is the model weights, prompts, system instructions, and knowledge base, essentially the sub-agent's reasoning itself. Snapshot fidelity is low because the output can be logged but not the reasoning process. If the sub-agent's model or knowledge has been updated, replaying its old output means testing the orchestrator against a frozen reasoning artifact that no longer represents the current sub-agent.

Table 1 summarizes the spectrum. The key insight is that as one moves from Level 1 to Level 4, the nature of drift shifts from data drift to reasoning drift. Data drift is observable and reproducible. Reasoning drift is opaque, as two different model versions may produce outputs that are both individually valid but differently reasoned, making it impossible to determine whether an evaluation failure is due to the orchestrator or the sub-agent's evolved reasoning.

Table 1. The Dependency Type Spectrum

Lvl	Type	What Drifts	Snapshot Fidelity	Replay Validity
1	Stateless API	Data, schema	High	Same query path
2	MCP Server	Data, schema, capabilities	High	If tool manifest unchanged
3	Plugin / Skill	Logic, rules, retrieval	Medium	Only at pinned version
4	Sub-Agent (LLM)	Model, prompts, knowledge, reasoning	Low	Tests orchestrator only

3.3 Why Inference-Time Logging Is Insufficient

A natural response to the temporal coherence problem is to apply the standard ML approach: log all inputs and outputs at inference time and use the logs for offline evaluation. In classic ML, the feature vector is fixed regardless of what the model predicts, enabling perfect offline replay. For agents, this approach has three fundamental limitations.

Path Dependence. Agent tool calls are conditional on previous tool responses. The agent observes metrics indicating a latency spike and decides to query deployment history next. A modified agent might observe the same metrics and decide to query logs first instead. The logged trajectory from the original agent is irrelevant because the modified agent would traverse a different path through the tool space.

Evaluating Agent Changes. Evaluation is needed most when changing the agent, such as updating the model, prompt, or tool set. But this is exactly when logged trajectories become stale, because the changed agent would call different tools in a different order with different parameters. Re-execution against live tools is required, which reintroduces the temporal coherence problem.

Sub-Agent Reasoning Opacity. If a sub-agent's output is logged, the log captures what it concluded but not why. If the sub-agent's model or knowledge has since been updated, replaying its old output tests the orchestrator against reasoning that no longer represents the current system.

When logging is valid: Inference-time logging works well for regression testing on the same agent version and for post-hoc analysis of individual incidents. The limitation is specifically about evaluating agent modifications or maintaining evaluation datasets as the dependency ecosystem evolves.

Table 2. Inference-Time Logging vs. Synthetic Snapshots

Evaluation Scenario	Inference-Time Logging	Synthetic Snapshots
Same agent, regression testing	Valid	Valid
Modified agent evaluation	Invalid (path dependence)	Valid (re-execution)
New tool added	No data exists	Counterfactual backfill
Sub-agent reasoning changes	Captures output, not reasoning	Partial (can regenerate)
Data retention compliance	Logged data subject to deletion	Synthetic data is retention-exempt

3.4 Challenge 1: Tool Drift

Dependencies evolve on their own release cycles without coordination with the agent team. The nature of this drift varies across the dependency spectrum: for APIs it is data and schema drift; for sub-agents it is reasoning drift. For example, in an incident RCA agent, the alerting system may update its severity taxonomy from {low, medium, high} to {P1, P2, P3, P4, P5}, while a log analysis sub-agent may receive a model update that changes how it categorizes error patterns. Schema changes cause format errors that are detectable; sub-agent reasoning changes produce subtly different diagnoses that are not.

3.5 Challenge 2: Temporal Incoherence

Even when individual dependency responses are available, they may reflect different points in time, creating a logically inconsistent view of the world. Consider an RCA agent diagnosing a latency spike at 14:00. The metrics service returns data from 14:00 (showing the spike). The deployment history, queried against a stale cache, shows the state at 15:00 (including a post-incident rollback and a follow-up hotfix). The log service returns logs from 15:00 showing errors caused by the hotfix, not the original incident. The agent now sees a world where the spike exists, a hotfix has been deployed, and the errors relate to the hotfix rather than the root cause—an impossible combination that leads to a misdiagnosis.

3.6 Challenge 3: Forward-Looking Data Gaps

When a new dependency is added to the agent's workflow, there is no historical data for it at the evaluation timestamp. For example, adding a canary metrics tool to an RCA agent means no canary data exists for past incidents. Teams face a choice: evaluate only on new incidents going forward (slow, small sample), exclude the new tool from historical evaluations (incomplete), or synthetically generate plausible data that is diagnostically consistent with the known ground truth.

3.7 Challenge 4: Privacy-Constrained Reproducibility

Data retention policies and privacy regulations (GDPR, CCPA) delete or anonymize data after a retention window. User-facing logs containing request parameters may be deleted after 30 days. An incident from 45 days ago can no longer be fully reproduced, as the log query tool would return empty results even though metrics and alerts are still available. Evaluation datasets therefore have an expiration date, and long-term regression testing becomes impossible without synthetic replacements that preserve diagnostic signals without sensitive content.

4. PROPOSED APPROACH

This paper proposes a Snapshot Environment layer that sits between the agent and real dependencies during evaluation. For each evaluation scenario, the snapshot environment provides a temporally coherent, synthetic view of all dependency responses at a target timestamp t_0, derived from a single scenario specification that defines the ground truth timeline of events. Figure 1 illustrates the architecture.

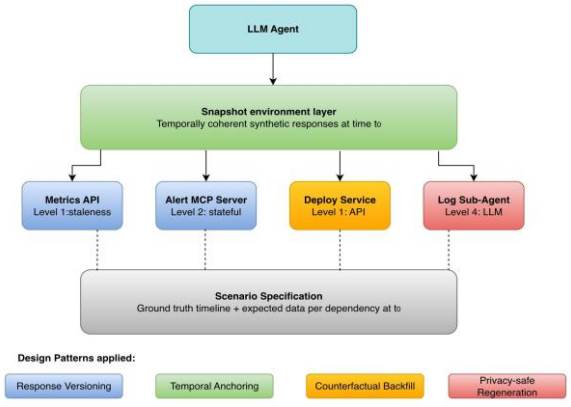

Fig 1: Snapshot environment architecture. The agent interacts with a snapshot layer that provides temporally coherent synthetic responses derived from a single scenario specification.

4.1 Design Pattern 1: Response Versioning

For Level 1–2 dependencies, record API schemas and response formats alongside evaluation datasets. Each scenario includes a dependency manifest specifying the expected schema version. For Level 3 dependencies, version-pin plugins and include the version identifier. For Level 4 dependencies (sub-agents), two options are documented with explicit tradeoffs: output replay (freeze the sub-agent's prior response, high consistency but

does not test current interaction) and controlled re-execution (run the current sub-agent against synthetic data, tests real interaction but introduces variance). Output replay is recommended as the default for regression testing and controlled re-execution for integration testing of new sub-agent versions.

4.2 Design Pattern 2: Temporal Anchoring

Generate all synthetic dependency responses from a single scenario specification that defines the ground truth timeline. For an RCA scenario: at t_0−10min a deployment is pushed, at t_0 latency spikes, at t_0+5min an alert fires, at t_0+15min a rollback is initiated. All tool responses are derived from this timeline, guaranteeing logical consistency regardless of when the evaluation is run.

4.3 Design Pattern 3: Counterfactual Backfill

For forward-looking data gaps, generate plausible synthetic data for new dependencies based on the scenario specification. The key insight is that evaluation requires data that is diagnostically consistent with the ground truth, not historically accurate. For canary metrics that did not exist during a past incident, generate synthetic canary-versus-baseline data that is consistent with the known root cause.

4.4 Design Pattern 4: Privacy-Safe Regeneration

For privacy-constrained reproducibility, generate synthetic log entries that preserve diagnostic signals (error codes, latency values, endpoint paths) without sensitive content (user IDs, request payloads). Template-based generation using the scenario specification as seed ensures that the diagnostic signal is maintained while compliance requirements are satisfied.

5. EXPERIMENTAL VALIDATION

5.1 Experimental Setup

A simulated Incident Root-Cause Analysis (RCA) agent is constructed that diagnoses production latency spikes using four tool dependencies: a Metrics Service (Level 1 API), an Alert Service (Level 2 MCP server), a Deployment History Service (Level 1 API), and a Log Analysis Sub-Agent (Level 4 LLM-based). The agent receives structured data from all four tools and outputs a root cause diagnosis in JSON format.

The agent is powered by Claude Sonnet (Anthropic) with temperature set to 0 for reproducibility. Five incident scenarios are designed with known root causes: (1) bad configuration deployment, (2) database connection pool exhaustion, (3) upstream third-party dependency timeout, (4) memory leak from code regression, and (5) DNS resolution failure. Each scenario includes coherent tool data at the incident time, plus drifted variants for each experimental condition.

The agent was evaluated under six conditions, each repeated three times per scenario (90 total LLM calls):

C1: Coherent Snapshot (baseline). All tools return data from the same point in time, consistent with the ground truth.

C2: Temporal Incoherence. Tools return data from different time windows. Metrics show the incident, but alerts reflect post-resolution state with new unrelated alerts, deployments show post-incident hotfixes that appear causal, and logs show errors from post-incident activity.

C3: Schema Drift. The alert service returns responses in a new schema with abbreviated service names, different field names (priority instead of severity), and removed human-readable message fields.

C4: Sub-Agent Reasoning Drift. The log analysis sub-agent categorizes errors using a different analytical framework (e.g., classifying DNS resolution failures as network connectivity issues), leading to different diagnostic framing.

C5: Data Gap. A canary metrics tool is added but data exists for only 2 of 5 scenarios.

C6: Restored Snapshot. Synthetic point-in-time snapshot applied using the proposed design patterns, restoring temporal coherence.

Three metrics are measured: Diagnostic Accuracy (correct root cause identified, binary per scenario), Average Evidence Quality (overlap between cited evidence and ground truth evidence, scored 0–1), and Consistency (same diagnosis across repeated runs).

5.2 Results

Table 3 presents the aggregate results across all six conditions, and Figure 2 visualizes diagnostic accuracy. Table 4 then provides the per-scenario breakdown that exposes which conditions broke which scenarios.

Table 3. Diagnostic Accuracy Across Experimental Conditions

Condition	Per-Run Accuracy	Scenarios	Avg Evid. Quality
C1: Coherent Snapshot	15/15 (100%)	5/5	0.60
C2: Temporal Incoherence	6/15 (40%)	2/5	0.45
C3: Schema Drift	12/15 (80%)	4/5	0.75
C4: Sub-Agent Reasoning Drift	12/15 (80%)	4/5	0.45
C5: Data Gap	12/15 (80%)	4/5	0.57
C6: Restored Snapshot	12/15 (80%)	4/5	0.60

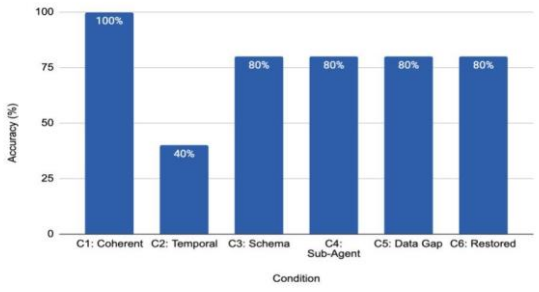

Fig 2: Diagnostic accuracy across experimental conditions. Temporal incoherence (C2) causes a 60 percentage-point drop from baseline; snapshot restoration (C6) recovers 40 percentage points

Table 4. Per-Scenario Diagnostic Outcomes Across Conditions

Scenario	C1	C2	C3	C4	C5	C6
Bad Deploy	✓	✓	cfg_chg	✓	✓	✓
DB Pool	✓	bad_dep	✓	✓	✓	✓
Upstream	✓	bad_dep	✓	✓	✓	✓
Memory Leak	✓	✓	✓	✓	bad_dep	bad_dep
DNS Failure	✓	cfg_chg	✓	net_iss	✓	✓

A ✓ indicates the correct root cause was identified in all three runs (3/3). Abbreviated labels indicate the (consistent) incorrect prediction: bad_dep = bad_deployment; cfg_chg = config_change; net_iss = network_issue. Full identifiers are shown in Figure 3.

5.3 Discussion

The results demonstrate three principal findings. First, temporal incoherence produces the most severe degradation, reducing diagnostic accuracy from 100% to 40%. A closer look at the C2 failures reveals a specific mechanism. The database connection pool exhaustion and upstream dependency timeout scenarios both collapsed to the same wrong diagnosis—bad_deployment—despite having different root causes. In both cases, the temporally drifted data presented post-incident hotfix deploys as the most recent visible change. The agent's diagnostic reasoning latched onto the post-incident deploy as the causal event, treating its proximity to the incident as evidence. Two genuinely distinct failure modes therefore produced the same misdiagnosis through the same mechanism: the agent was not confused by missing signal but misled by plausible-looking signal that happened to post-date the incident. This failure mode is more dangerous than random error because the wrong answer arrives with high confidence (0.90 in both cases) and a coherent supporting narrative.

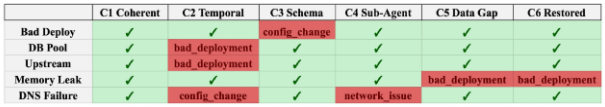

Fig 3: Per-scenario diagnostic outcomes across conditions. Green cells indicate correct diagnosis; red cells show the incorrect predicted cause

Second, schema drift (C3) and sub-agent reasoning drift (C4) each reduced accuracy by 20 percentage points, but through qualitatively different mechanisms. The DNS resolution failure scenario illustrates the distinction directly: it is misdiagnosed under both C2 and C4, yet for different reasons and with different wrong answers. Under C2 (temporal incoherence), the scenario is diagnosed as config_change, with the agent attributing the latency to a TLS certificate rotation that appeared in the drifted data window. Under C4 (sub-agent reasoning drift), the same scenario is diagnosed as network_issue, with the log analysis sub-agent categorizing DNS resolution errors as generic network connectivity failures and the orchestrator adopting that framing. The two failures are qualitatively different. The C2 error is factually wrong: TLS certificate rotation did not cause the latency. The C4 error is defensible but imprecise: network_issue is a superset of dns_failure, yet it is the wrong answer for an operator who needs to know whether to restart name servers or investigate routing. This pair maps directly onto the Level 1–2 (data drift) versus Level 4 (reasoning drift) distinction introduced in Section 3.2 and illustrates why a single accuracy metric cannot describe what has gone wrong in an agent evaluation.

Third, synthetic snapshot restoration recovered accuracy from 40% to 80%—a 40 percentage-point improvement—validating the proposed design patterns. The remaining 20-point gap between C6 (80%) and C1 (100%) is concentrated in a single scenario, the memory leak from code regression, which fails under both C5 (data gap) and C6 (restored snapshot) with the same incorrect prediction of bad_deployment. This is a scenario-level limitation rather than a framework failure. When canary metrics are added to the agent's toolset and synthetically backfilled, the canary signal correlates with the deployment that immediately preceded the leak's onset. Two root causes—memory leak and bad deployment—both remain plausible given the evidence, and the agent selects the more common explanation. The snapshot framework can restore temporal coherence across dependencies but cannot resolve inherent scenario ambiguity when multiple root causes are consistent with the data. This points to a practical boundary for the approach: evaluation scenarios should be designed to discriminate between candidate root causes, and counterfactual backfill for new dependencies should preserve that discrimination.

Fourth, diagnostic accuracy and evidence quality do not track each other. As shown in Figure 4, schema drift (C3) yields the highest average evidence quality at 0.75 despite an 80% accuracy, while sub-agent reasoning drift (C4) produces one of the lowest at 0.45 at the same 80% accuracy. The gap reflects what each type of drift does to the agent's reasoning substrate. Schema drift leaves the underlying evidence record intact; the agent cites the right signals but misreads a single field. Sub-agent reasoning drift changes the evidence the orchestrator sees in the first place, because the sub-agent has surfaced a different set of observations. The implication for evaluation is that accuracy alone is insufficient: a secondary metric tracking evidence overlap is needed to distinguish surface-level errors from reasoning-level ones, which require fundamentally different remediation.

Fig 4: Accuracy versus evidence quality across conditions. Evidence quality is scaled by 100 for visual comparability. Conditions C3 and C4 share 80% accuracy but diverge sharply in evidence quality, reflecting the data-drift versus reasoning-drift distinction

All conditions exhibited perfect consistency (1.00) across repeated runs, confirming that the observed accuracy differences are systematic effects of temporal incoherence rather than stochastic variation.

6. CONCLUSION

This paper formalized the temporal coherence problem in LLM agent evaluation, an increasingly critical challenge as agents are deployed in production environments with multiple independently evolving tool dependencies. It introduced a

dependency type spectrum that classifies agent dependencies from stateless APIs to LLM-based sub-agents, formalizing the qualitative shift from data drift to reasoning drift. It also demonstrated that the standard ML approach of inference-time logging is insufficient for agent evaluation due to path dependence, and introduced four design patterns for synthetic point-in-time snapshot generation.

Experimental validation on a simulated incident RCA agent demonstrated that temporal incoherence alone can reduce diagnostic accuracy by 60 percentage points, while synthetic snapshot restoration recovers 40 percentage points. Sub-agent reasoning drift presents a distinct challenge where different analytical framing leads to defensible but incorrect diagnoses, validating the need for dependency-type-aware evaluation strategies.

Future work includes scaling to larger dependency ecosystems, automated scenario generation using a Scenario Generation LLM Agent, integration with existing benchmarks such as AgentBench and τ-bench as a temporal consistency layer, and formal metrics for measuring the temporal fidelity of synthetic evaluation environments.

7. CONFLICTS OF INTEREST

The author declares no conflict of interest regarding the publication of this paper.

8. AI DISCLOSURE STATEMENT

AI-assisted tools (Claude, Anthropic) were used to assist with literature search, Claude Code for code implementation of the experimental prototype, and language editing during manuscript preparation. All research design, analysis, interpretation, and conclusions are the author's own. The experimental agent used in Section 5 was powered by Claude Sonnet for the RCA diagnosis task.

9. REFERENCES

[1] LangChain, "State of AI Agents," LangChain Survey Report, 2024. [Online]. Available: https://www.langchain.com/stateofaiagents

[2] S. Mohammadi et al., "Evaluation and Benchmarking of LLM Agents: A Survey," arXiv preprint arXiv:2507.21504, KDD 2025 Tutorial, 2025.

[3] X. Liu et al., "AgentBench: Evaluating LLMs as Agents," International Conference on Learning Representations (ICLR), 2024.

[4] S. Zhou et al., "WebArena: A Realistic Web Environment for Building Autonomous Agents," International Conference on Learning Representations (ICLR), 2024.

[5] S. Yao et al., "τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains," arXiv preprint arXiv:2406.12045, 2024.

[6] C. E. Jimenez et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" International Conference on Learning Representations (ICLR), 2024.

[7] C. Ma et al., "AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents," Advances in Neural Information Processing Systems (NeurIPS), 2024.

[8] Anthropic, "Demystifying Evals for AI Agents," Anthropic Engineering Blog, 2025. [Online]. Available: https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents

[9] Amazon Web Services, "Evaluating AI Agents: Real-World Lessons from Building Agentic Systems at Amazon," AWS Machine Learning Blog, 2025. [Online]. Available: https://aws.amazon.com/blogs/machine-learning/

[10] ReliabilityBench, "Evaluating LLM Agent Reliability Under Production-Like Stress Conditions," arXiv preprint arXiv:2601.06112, 2026.

[11] M. Cemri et al., "Why Do Multi-Agent LLM Systems Fail?" arXiv preprint arXiv:2503.13657, 2025.

[12] Microsoft AI Red Team, "Taxonomy of Failure Modes in Agentic AI Systems," Microsoft Whitepaper, 2025. [Online]. Available: https://www.microsoft.com

[13] S. Kapoor et al., "AI Agents That Matter," Transactions on Machine Learning Research (TMLR), arXiv preprint arXiv:2407.01502, 2024.

[14] P. Castells et al., "Offline Recommender System Evaluation: Challenges and New Directions," AI Magazine, vol. 43, no. 1, 2022.

[15] N. Patki, R. Wedge, and K. Veeramachaneni, "The Synthetic Data Vault," IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 399–410, 2016.

[16] Anthropic, "Model Context Protocol Specification," 2024. [Online]. Available: https://modelcontextprotocol.io