Multi-Layer AI Architecture for Real-Time Detection of Workflow and Data Integrity Failures in Distributed Retail Platforms

Sai Sruthi Puchakayala

***Abstract*—Distributed retail systems run on heterogeneous data systems whose joint state represents business invariants that cannot be verified in the life-cycle of any one of them. Silent inconsistencies arise when, downstream, purchase orders go missing, quantities do not match, suppliers are blocked, or data drifts across systems. Rule-based monitoring systems and single-system observability do not detect these failures by definition. Text-to-SQL and retrieval-augmented generation technologies answer these questions without executing the multi-step investigation an incident demands. We present a two-layer AI architecture that separates deterministic data access and non-deterministic reasoning. The semantic data access layer, implemented as an MCP server, defines eight domain tools as well as five templates with natural-language descriptions. The stateful reasoning layer parses incidents, reasons about them, selects tools, takes investigative steps and derives remediation following a seven-step workflow captured as a knowledge-based process in a Camunda BPMN process engine. Production evidence from a large international retailer shows that average resolution effort per incident fell from 9 hours to 2.25 hours, that Level-1 incident volume fell from 33 to about 8 over comparable 2-month windows, and that engineering escalations fell from 67 to about 17 over comparable 3-month windows, a reduction close to 75% in both effort and volume. Tool description debt is a new class of technical debt that is specific to LLM-based observability systems. A four-element taxonomy is proposed.**

***Keywords*—anomaly detection, cross-system invariants, large language models, model context protocol.**

I. Introduction

Modern retail applications run as constellations of microservices coordinating ordering, planning, suppliers, payments, fulfillment and analytics. The joint state of the application carries the semantic invariants of the business. An order in the order-management system corresponds to a purchase order in the procurement system, just as a planned shipment corresponds to an approved supplier quote. This means that, when the BPMN engine is executing a given workflow instance, it must eventually reach a terminal activity within the deadline. Invariants are not known to any single component, as it can only access a local projection of a joint state. An operator knows that a failure has happened when something downstream cannot continue because, for instance, a vendor cannot ship goods, an invoice cannot post, a store cannot be restocked, or an on-call engineer must cross-reference systems with different schemas and vocabularies.

Industry surveys of distributed-system observability and AIOps find a general gap between the abundance of pattern detection on individual signals and the multi-step investigations real cross-system incidents demand [1]. The few attempts to fill this gap through intra-service anomaly detection, distributed tracing, or natural language interfaces to single databases fall short of the multi-source reasoning an operational backbone would require.

These LLMs show sufficient reasoning, planning, and tool-use capabilities that a multi-step diagnosis is possible. This led to a tool invocation protocol for LLMs, called the Model Context Protocol (MCP), to expose tools and external resources to the LLM in a standardized format that was first proposed in late 2024 [2]. Initial work on the MCP server has identified security and maintainability issues [3]. Another side effect of common use of MCP servers is tool descriptions carrying over technical debt as the underlying domain changes. This affects the quality of reasoning for any LLM-based observability system built on top of MCP.

The contributions are as follows. First, a two-layer architecture, where a semantic data access layer with an MCP server containing eight tools, five analysis templates, and natural-language documentation is separated from a stateful reasoning layer containing a seven-step reasoning workflow. Second, the mapping of the reasoning loop of the agent to the Belief-Desire-Intention (BDI) agent model [4] and the exposure of the inner process state through the integration of the Camunda BPMN engine [5]. Third, using production validation results from a large international retailer, the authors show five cross-system invariants representing anomaly types. They also show that average resolution effort per incident fell from 9 hours to 2.25 hours while incident and escalation volumes declined by close to 75%. Fourth, the paper identifies tool description debt as a new technical-debt category that extends the categories of Cunningham [6] and Kruchten et al. [7].

The paper is structured according to the IMRAD format: Section II describes its methods, including related work and theory, the formal problem statement, and design of the proposed two-layer architecture. Section III describes results: production measurements from the deployment, their mapping onto the anomaly classes, and a representative investigation. Section IV describes lessons learned, threats to validity, tool description debt, and generalizability. Section V

Manuscript received May 5, 2026.

S.S. Puchakayala is a Software Development Engineer III at Walmart from Centerton, Arkansas (e-mail: saisruthi.puchakayal@walmart.com).

concludes.

II. METHODS

A. Related work and theoretical foundations

There are three lines of research that are related to the problem: anomaly and failure detection in IT systems. A deep-learning anomaly detection survey by Pang et al. [8] categorized reconstruction and prediction deep-learning approaches for logs, metrics and traces. A wider AIOps failure management survey by Notaro et al. [1] considered 100 papers, within 5 high-level properties, for each of detection, prediction, root cause analysis and remediation. However, industrial experience with observability tools shows that distributed tracing alone leaves most diagnostics to engineers [9], and none of these approaches reason over cross-system semantic invariants.

The second line of work studies language-model agents and tools. An example is chain-of-thought prompting [10], or prompting the language model to display its intermediate reasoning steps. ReAct [11] interleaves reasoning and action selection. Toolformer [12] shows how to teach LMs when to call external APIs in a self-supervised way. LLM-based autonomous agents have been surveyed [13]. Multi-agent systems have also been studied [14]. LLMs in software engineering have been surveyed [15]. For example, RCACopilot [16] applies LLMs to root-cause analysis for cloud incidents, achieving up to 0.766 accuracy on real cloud incident datasets. Section IV.A will discuss the differences between RCACopilot and this work. A survey of MCP's security threats and maintainability in [2, 3] is the first academic survey in this ecosystem.

The third line focuses on natural-language interfaces. Text-to-SQL surveys [17, 18] discuss the progression from semantic parsing to natural language generation with LLMs as well as state-of-the-art performance on text-to-SQL benchmarks such as Spider and BIRD. Retrieval-augmented generation [19] seeks to address hallucination [20, 21] by conditioning LLM outputs on retrieved evidence. Text-to-SQL or RAG have a common limitation in this case: to answer a question that can only be consulted a single data source. On the contrary, to classify an impossible anomaly, it is needed to check many systems. The results can also be used to update a state in a process engine [5].

The architecture is based on classical theoretical foundations, such as information hiding [22] that motivates separating deterministic data access from non-deterministic domain reasoning, and hexagonal architecture [23] that matches this model well by exposing domain capabilities via ports whose adapters vary independently of the domain core. The separation of mechanism from policy [24] can be mapped to tool set, mechanism, and agent's reasoning and policy. In BDI theory [4], the agent's parser is a belief-updating operator, the router is a desire-forming operator, and the synthesizer is an intention-executing operator. The ReAct model [11] implements this theoretical loop in the language model context. Literature about industrial AI in supply chains [25, 26] or digital twins advocates resilient operation under the influence of AI, but focuses on demand forecasting, planning, and disruption simulation rather than the integrity of interdependent systems in operation. Research on technical debt [6, 7, 27] addresses code, design, architecture, and documentation debt. In Section IV.B, it is argued that descriptions of tools built with LLMs form a new kind of debt.

B. Problem formalization

The platform is modelled as a finite set of systems $S = \{S_1, S_2, \ldots, S_n\}$. Each system S_i has its own state space Σ_i and exposes a projection $\pi_i : \Sigma \to \Sigma_i$ of the joint state $\Sigma = \Sigma_1 \times \cdots \times \Sigma_n$. A business invariant I is a predicate on Σ whose truth depends on values that lie in two or more component projections. Such a predicate is called a k-ary cross-system invariant if at least k component projections are needed to evaluate it.

Observation 1 (single-system observability). Let O_i denote an observer whose access is limited to π_i. For any k-ary cross-system invariant I with $k \geq 2$, no single-system observer O_i can determine I in the general case: two joint states σ, σ' may agree on π_i yet differ on the value of I, since I depends on at least one projection π_j with $j \neq i$. The observation is folkloric in the distributed-systems literature. It was recorded here to motivate the architectural decision to expose every relevant system through a uniform tool interface in Layer 1, so that any agent above Layer 1 can in principle, assemble the projections needed to evaluate cross-system invariants.

The framework is instantiated on five anomaly classes that arise in retail operations. Each class violates a cross-system invariant, whose arity is recorded in Table 1.

Table 1. Five Anomaly Classes as Cross-System Invariants

Class	Invariant	Arity k
A1. Quantity reconciliation	∀ order o, converted quantity in OMS = converted quantity in Planning = quantity shipped	3
A2. Missing purchase orders	∀ order o requiring procurement, ∃ PO p in Procurement with link(o)=p	2
A3. Stalled workflow states	∀ process instance π, time−in−activity(π, a) ≤ SLA(a)	2
A4. Quote/approval blockages	∀ supplier order, status(quote) ∈ {approved, active} at order creation	2
A5. Cross-system data drift	∀ shared field f tracked across systems S, T: π_S(f) = π_T(f)	2

C. System architecture

The architecture has two layers with a semantic contract, with Layer 1 being the semantic data access layer running as an MCP server. There are eight domain tools and five analysis templates for wrapping heterogeneous back-end systems, order management, planning, supplier management, the

Camunda BPMN engine, payments, etc. Each tool is defined by a natural-language description that serves as the semantic contract of the tool. It specifies what a tool does, when it is applied, and the shape of the result. The implementation of the contract must be deterministic; tooling should provide explicit input and output schemas, declare idempotency, and annotate side effects.

Layer 2, the stateful reasoning layer, is also an AI agent. It takes incident tickets in natural language and outputs remediation reports. Like Layer 1, it has no access to the backend systems, and all actions are performed by the tool described in Layer 1, separating mechanism and policy [24] between deterministic (Layer 1) and non-deterministic (Layer 2) concerns. New tools can be registered with the MCP server without retraining the agent, and the agent can modify its reasoning policy at runtime without modifying the code running in the backend, as shown in Figure 1.

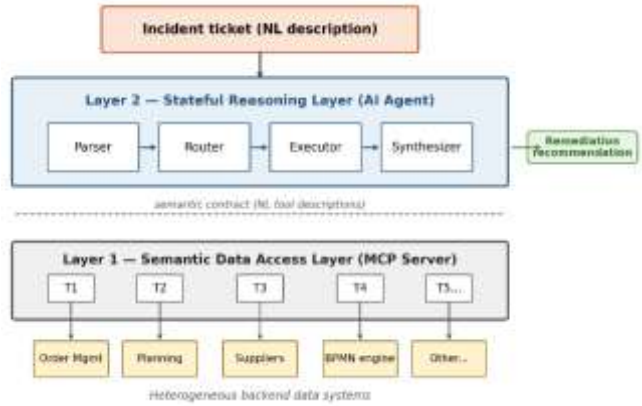

Fig. 1. Two-layer architecture

All Layer 1 tools have five components: tool name, natural-language description of the tool, input schema, output schema, applicability declaration, idempotency declaration, and side effects. The natural-language description encodes most of the discriminative power. It is used by the agent to decide whether to call the tool at the current step in the investigation. Five analysis templates supplement the bare tool catalog with patterns of repeated composite queries, such as "verify quantity consistency for an order across OMS and Planning" and "scan the BPMN engine for instances stuck in activity X for more than D days". They encapsulate tools into reusable investigation primitives, which reduce the agent's reasoning burden.

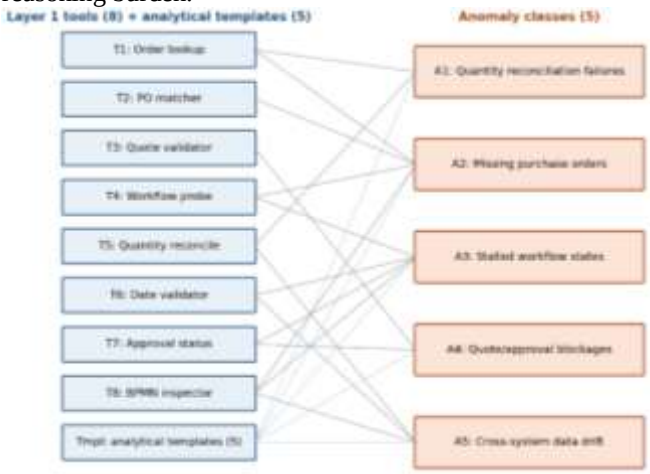

Fig. 2. Bipartite mapping from Layer 1 tools and analytical templates to the five anomaly classes

Fig. 2 depicts the bipartite mapping between tools/templates and the five anomaly classes.

This is a many-to-many mapping, so the BPMN inspector would join the task of identifying stalled workflows, missing POs, or drift in data across systems. Its advantages are the architectural claim, independence from a one-tool-per-anomaly mapping, and the reusability of its tool catalogue for new classes of anomalies through the introduction of additional investigation policies in Layer 2.

The agent subsequently performs language understanding (parsers carry out entity/system finding and explicit signal extraction), problem classification (routers map the result to one of five anomaly types, or unknown, triggering fallback behavior), tool selection (descriptors carry out filtering based on applicability conditions), investigation (tool calls over the MCP server and aggregation of the resulting evidence), analysis (reasoning over the aggregated result and the decision whether another tool call is needed), remediation synthesis (causal narrative and recommendations on remediation action), and status update (the results are written back to the BPMN engine via a specific tool, closing the loop with the process orchestrator).

It closely follows the widely used BDI agent model [4]. The parser fills the belief base with grounded facts. The router then specifies a desire that bounds the tool search space, and the synthesizer commits to an intention, that is, the recommendation. It is not only the terminology which is transformed: the belief base B is updated with new evidence E in lines 6-10 of Algorithm 1 (see Figure 3) before the next tool can be selected. This is an example of the belief revision loop mechanism at the core of the practical BDI implementations at the granularity of a single incident. It uses BDI vocabulary as a conceptual frame at the granularity of a single incident, while a fully fledged BDI agent maintains plan libraries and stable intentions across incidents, which is out of the scope of this work.

```
Input:  incident ticket t (natural language)
Output: remediation report r

1:  B ← Parse(t)                    ▷ belief update
2:  c ← Classify(B)                 ▷ anomaly class
3:  if c = unknown then return Fallback(B)
4:  T_set ← SelectTools(c, B)       ▷ by descriptors
5:  E ← ∅
6:  while not Sufficient(E) do
7:      τ ← NextTool(T_set, E)
8:      E ← E ∪ Invoke(τ, B)        ▷ via MCP server
9:      B ← Update(B, E)
10: end while
11: r ← Synthesize(B, E, c)         ▷ causal narrative
12: WriteBPMNStatus(r)              ▷ close the loop
13: return r
```

Fig. 3. Stateful reasoning workflow

Sufficient(E) holds if (i) the invariant arity knowledge is covered by the classified anomaly, and for a k-ary invariant, the joint of tool returns is sufficient to cover all k projections or (ii) the confidence on the synthesis by the agent is higher than the class threshold. Below the threshold, Sufficient(E) does not continue with the synthesis under the invariant, and the partial evidence and the chain of reasoning are given to

the human engineer. The agent thus does not operate over free-form associations, but only the evidence returned from tool calls. This was motivated by prior work on hallucination in foundation models (e.g., [20, 21]) and recommendations from an AIOps failure-management survey [1].

Business processes orchestrated by the Camunda BPMN engine [5] can be accessed by process instances and activity histories so that the BPMN engine becomes a first-class data source. An MCP tool collects process state, current activities, time in activities and completion percentages, and audit information. A related MCP tool updates process states, enabling the agent to transition the state of an incident with auditable transitions. The integration's result is to turn the BPMN engine from a passive runtime into an observable, addressable substrate carrying part of the architecture's cross-system invariants.

Each layer component, listed in Table 2, has its corresponding failure modes that appear when the respective layer is removed.

Table 2. Design-Level Assessment of Expected Anomaly-Class Coverage when Components are Removed (F-Full, D-Degraded, A-Absent)

Configuration	**A1**	**A2**	**A3**	**A4**	**A5**
Full architecture	F	F	F	F	F
No semantic descriptions	D	D	D	D	D
No stateful workflow	A	A	A	D	A
No BPMN integration	D	D	A	D	D
No classification step	D	D	D	D	D

The annotations represent the authors' design assessment of the deployment in Section III and carry no status of controlled measurement. Disabling components in the live incident-response loop was ruled out, since a deliberate degradation of a business-critical production process is operationally unacceptable, and Section IV.C outlines a replay ablation on synthetic incident traces outside production as future work. Without the semantic layer, an agent relies solely on raw APIs and brittle prompt engineering every time a new tool is introduced. The agent is a single-call generator, not a workflow. This works for single-step answers to questions, but multi-step problems are not handled correctly. Without BPMN integration, silent deadlock failures and workflows failing to complete are not detected. Without classification, the routing space flattens and fallback behaviors between each class are lost, leading to less reliability and more escalation.

III. RESULTS

A. *Setting and methodology*

The system was deployed at a large international retailer that operates an enterprise-scale supply chain platform. Observation covered a 2-month window for Level-1 operational incidents and a 3-month window for escalations to engineering, with matching windows before and after deployment. The primary metric is resolution effort per incident, defined as the operational effort that the incident tracking system records between ticket creation and the moment the ticket is marked resolved with a remediation recommendation. Both the pre-deployment and the post-deployment samples derive from the same logs, the same ticket structure, and the same routing rules. The tracker attributes each escalated incident to 1 of 4 operational categories at escalation time: purchase-order-related failures, item-creation failures, workflow-transition failures, and supplier-quotation or catalog failures. Per-incident timestamp records cannot leave the production perimeter, so the reported values are aggregate counts and averages taken from these logs.

The agent resides in Layer 2, which has a frontier instruction-tuned LLM with a long context window. Layer 1 has eight domain tools and five analytical templates, which encapsulate the order management, planning, supplier management, payments, and Camunda BPMN systems. The MCP server is delivered via a CI/CD pipeline with required static analysis and human review gates, and runs as a standalone always-on service consumed by the agent at runtime. Independent tool calls are parallelized where possible, resulting in low minutes for end-to-end agent latency on a typical investigation. The single largest cost is that of backend query time on large operational databases.

B. *Quantitative results*

During the 2-month pre-deployment window the platform recorded 33 Level-1 operational incidents with an average resolution effort of 9 hours per incident, which corresponds to 297 hours of operational effort. Over the matching post-deployment window the count fell to about 8 incidents and the average effort per incident fell to 2.25 hours. Escalations to engineering declined from 67 to about 17 across comparable 3-month windows, and Table 3 breaks these escalations down by operational category while Fig. 4 visualizes the volumes. Manual investigation of a complex multi-system case could previously consume up to 2 full working days. Per-incident effort and incident volume therefore both declined by close to 75%.

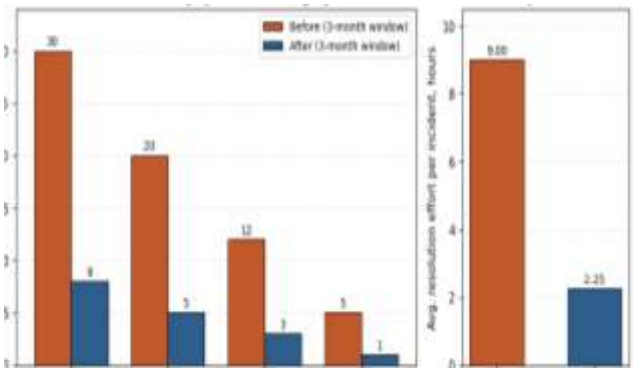

Fig. 4. Engineering escalations and resolution effort per incident before and after deployment, by operational category

Table 3. Engineering Escalations by Operational Category over Comparable 3-Month Windows

Category	Before	After	Invariant class
Purchase-order-related failures	30	≈8	A2
Item-creation failures	20	5	Outside taxonomy
Workflow-transition failures	12	3	A3
Supplier-quotation or catalog failures	5	≈1	A4

The operational categories of the tracker align with the invariant classes of Section II in an approximate way. Purchase-order-related failures correspond to class A2, workflow-transition failures correspond to class A3, and supplier-quotation failures correspond to class A4, while item-creation failures fall outside the 5-class taxonomy and classes A1 and A5 receive no dedicated escalation volume in the tracker data. The 5-class taxonomy therefore remains a conceptual instrument of the design, and the quantitative evidence follows the categories that the tracker records. Per-incident distributions, medians, interquartile ranges, and significance tests are absent from this evidence for 2 reasons. The production owner releases aggregate counts and averages alone, and the post-deployment samples of about 8, 5, 3, and 1 escalations per category sit below the threshold at which a rank-based test such as the Mann-Whitney U carries meaning. Reporting such a test on these samples would create an appearance of rigor that the data cannot support.

C. Qualitative effects and coverage

In addition to the headline effort reduction, 4 qualitative effects happened in production. Investigations run in parallel as the agent invokes independent tools and talks to the on-calls, who are no longer bottlenecks. Knowledge has democratized. The engineer who isn't privy to the tribal knowledge about one of the back-end systems still sees a coherent story because Layer 2 creates a causal explanation. This gave rise to nightly sweeps. The same tool used for on-demand troubleshooting was run in batch mode against the joint state to flush out inconsistencies that would have otherwise raised tickets. This not only sped up data engineers and operations iterations, but the BPMN engine also left a full audit trail of agent activity.

The five anomaly classes do not behave likewise and the unit of measure conversion tool is the most helpful for quantity reconciliation purposes. The missing purchase orders anomaly requires a join of the two systems as per the PO matcher tool T2. Stalled process states are finally visualized by tool T8, the BPMN inspector, while the consequences of quote and approval blockages depend on the supplier-status tool. The cross-system data drift runs the date and quantity validators in parallel (Fig. 2).

D. Representative investigation trace

The following trace is a record of a repeating case in the missing-purchase-order class. It is an illustration, and is not an instrumented run. The sequence is what the tool ran in production.

An on-call engineer receives a ticket: "PO 4421 missing, vendor cannot proceed". Fig 5 depicts the agent's response path. The identifier is extracted by the parser, and the systems are inferred to be order management and procurement.

Fig. 5. Investigation timeline for the missing-PO case, representative reconstruction

Marking the ticket as A2, the router invokes T1, an agent which tries to search for PO 4421 in an order-management system. Since it finds nothing, it invokes T2, a PO matcher across multiple systems. The matcher queries the planning system, which reports that order 9817 expects a purchase order that was never created and that the expected identifier matches PO 4421. The agent suspects it might be an orphan order and invokes its BPMN inspector T8 on order 9817's workflow instance. When the inspector points out a stuck instance at the PO-creation activity past SLA, the agent determines that the workflow timed out due to the absence of a procurement counterparty for the order at the time of PO creation. It suggests the following remedial actions: manually re-run the PO creation activity after data validation and increase its SLA to account for latency at earlier activities.

Three trace properties are noteworthy: the agent never directly calls a back-end tool, it logs which tool and rationale are used at each call, and the operators can replay the chain and inspect the evidence returned by each consecutive tool call along a reasoning trace. The whole investigation closes in under 40 minutes from ticket arrival on the underlying real incident, against a manual baseline that averages 9 hours of effort per incident.

IV. DISCUSSION

A. Why the architectural separation works, and how it differs from related LLM-AIOps work

At least since that idea that mechanism and policy be on opposite sides of an interface [24], the boundary between Layer 1 and Layer 2 should contain the mechanism: deterministic and auditable, explicit schemas and side-effect annotations. Layer 1 is the natural language descriptions, along with a semantic contract with which both layers can evolve. Layer 2 contains the policy, the choice of which tools to fire in what order against what evidence. These tools will

not retrain the agent; the agent uses prompt updates and routing rules to adjust its policy, leaving the underlying deterministic substrate untouched.

Microsoft's RCACopilot [16] is the closest to this work, but their system differs architecturally and operationally in that it performs root-cause analysis on cloud incidents using handler-classified retrieval and in-context examples. The substrate is monitor data and historical incident text, all in a cloud. The current implementation addresses cross-system integrity invariants on heterogeneous operational databases and a BPMN engine, structures the tool catalogue as an MCP server (for extensibility with new tools without retraining), and integrates the workflow engine as both a data source and write-back target. RCACopilot was reported to have an accuracy of 0.766 on the single-cloud incident benchmark, and these design choices solve separate pieces of the LLM-AIOps space.

B. Tool description debt: a new technical-debt category

Although the technical-debt metaphor devised by Cunningham [6] has been extended to other software artifacts such as design, architecture, and documentation [7, 27], experience with production led to a new risk category: tool description debt. The agent's tool selection step is weakened when the description of each tool in natural language diverges from the actual behavior of that tool. Four constituent forms were identified by the end of the deployment (Table 4).

Table 4. Taxonomy of Tool Description Debt

Type	Symptom	Remedy
D1. Vague	Description underspecifies applicability. Agent invokes tool in cases that return empty or noisy results.	Sharpen the description, add applicability conditions and counter-examples in metadata.
D2. Overlapping	Two or more tools claim similar semantics. Router’s decision boundary blurs.	Disambiguate scope, merge duplicates, or introduce hierarchical descriptors.
D3. Stale	Description survives a business-logic change. Tool now behaves differently from what the description claims.	Couple description reviews with code changes. Gate releases on description-test pairs.
D4. Over-specified	Description embeds operational details subject to change. Maintenance cost dominates value to the agent.	Move volatile details into runtime metadata. Keep the description focused on intent.

Vague or overlapping tool descriptions underspecify the conditions where the tool applies, making the agent call it even when it returns an empty response or adds noise. Overlapping tool descriptions have overlapping semantics, making the router's decision boundary less focused on specific tasks and slowing down the overall search. Stale descriptions can persist even when the underlying business logic has changed, leading the agent to invoke a tool with incorrect assumptions.

Over-specified descriptions include operational detail that is costly to maintain but that helps the agent reason. Such a debt can amass without warning because no rule fires when a description drifts. The only observable indicator is a degradation in the quality of investigations. It is distinct from documentation debt since it is read by a non-deterministic consumer that makes no audit of the reading process. Paying off this debt requires treating tool descriptions as first-class engineering artifacts, and accounting for reviews of selection behavior and tests of that behavior. The four-element taxonomy from Table 4 is a starting point for further empirical research on other deployments.

C. Production lessons, threats to validity, and generalizability

Taking into account the threat model described in Hou et al. [2] and the maintainability concerns discussed in Hasan et al. [3], it was explicitly asserted that the security of the MCP layer depends on the CI/CD pipeline's ability to enforce static analysis checks and a human approval gate for each tool before it reaches production. Side effects should be annotated on every tool descriptor to route every state mutation through the BPMN engine where it can be interrupted by the human-in-the-loop step. A containment policy on Layer 2: only allowed the agent to work on evidence returned by tool calls, not free-form associations (Section II.C). This reduces prompt-injection attack surface from incident-ticket text. Analyzing security properties with formal threat models and adversarial evaluation under a matched production load is future work.

Cross-system data latency was the dominant noise source. The agent observed, but did not affect, transient drift which resolved itself. A short synthesis cooldown time suppresses spurious alerts. Internationally dispersed supply chains required explicit unit-of-measure normalizations inside each analytical template. Operator access to audit trails builds trust. Tool owners update the descriptions whenever the underlying logic changes. The cost-effectiveness of a small review board for descriptors has been demonstrated within weeks. Internal validity remains limited by a single deployment without randomization. The quantitative evidence consists of aggregate counts and averages, and per-incident distributions with medians, interquartile ranges, or percentiles stay inside the production perimeter. Post-deployment samples of about 8, 5, 3, and 1 escalations per category are too small for a rank-based significance test, so no such test is reported. Construct validity rests on a single metric, resolution effort per incident as the tracker records it, and wall-clock elapsed time, investigation accuracy, and operator workload would give a fuller picture. The approximate mapping between tracker categories and invariant classes further limits how far the per-category counts speak to the taxonomy. The design assessments of Table 2 remain analytical judgments of the authors, and a replay ablation on anonymized or synthetic incident traces outside production, in which direct API calls bypass the MCP server, the stateful workflow collapses into single-step generation, and the BPMN integration is disabled, offers a feasible route to component-level attribution and is left as

future work. Reproducibility is limited because production data cannot be shared, and the full publication of the algorithm, the taxonomy, the deployment procedure, and the case study compensates in part. The external validity of the framework rests on whether the invariant structure can be applied elsewhere. Since financial transactions, healthcare claims processing and telecommunications operational support satisfy the criterion, the architectural skeleton could migrate if domain-specific tool catalogs are substituted for the retail tool catalogs, and empirical research is future work.

V. CONCLUSION

The paper proposes a two-layer AI architecture that tracks workflow and data integrity errors in real-time in distributed retail applications. A semantic data access layer exposes MCP domain tools with descriptions articulated in natural language. It has a stateful reasoning layer that runs a seven-step BDI process in the Camunda BPMN engine.

In a production deployment at a large international retailer, average resolution effort per incident fell from 9 hours to 2.25 hours, Level-1 incident volume fell from 33 to about 8 over comparable 2-month windows, and engineering escalations fell from 67 to about 17 over comparable 3-month windows, a reduction close to 75% across effort and volume. The architecture raises the challenge of tool description debt, a new type of technical debt, which can be represented to ease empirical research using a four element taxonomy.

Future work will pursue replay ablations on anonymized or synthetic incident traces outside production, extensions to further domains, and measures of tool description quality that can be computed before drift degrades investigation quality.

REFERENCES

[1] P. Notaro, J. Cardoso, and M. Gerndt, "A survey of AIOps methods for failure management," ACM Trans. Intell. Syst. Technol., vol. 12, no. 6, art. 81, 2021, doi: 10.1145/3483424.

[2] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model context protocol (MCP): landscape, security threats, and future research directions," arXiv:2503.23278, 2025, doi: 10.48550/arXiv.2503.23278.

[3] M. M. Hasan, H. Li, E. Fallahzadeh, G. K. Rajbahadur, B. Adams, and A. E. Hassan, "Model context protocol (MCP) at first glance: studying the security and maintainability of MCP servers," arXiv:2506.13538, 2025, doi: 10.48550/arXiv.2506.13538.

[4] A. S. Rao and M. P. Georgeff, "BDI agents: from theory to practice," in Proc. 1st Int. Conf. Multi-Agent Systems (ICMAS), 1995, pp. 312–319.

[5] J. Ortiz, V. Torres, and P. Valderas, "Microservice compositions based on the choreography of BPMN fragments: facing evolution issues," Computing, vol. 105, no. 1, pp. 375–416, 2023, doi: 10.1007/s00607-022-01128-8.

[6] W. Cunningham, "The WyCash portfolio management system," ACM SIGPLAN OOPS Messenger, vol. 4, no. 2, pp. 29–30, 1992, doi: 10.1145/157710.157715.

[7] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: from metaphor to theory and practice," IEEE Software, vol. 29, no. 6, pp. 18–21, 2012, doi: 10.1109/MS.2012.167.

[8] G. Pang, C. Shen, L. Cao, and A. van den Hengel, "Deep learning for anomaly detection: a review," ACM Comput. Surv., vol. 54, no. 2, art. 38, 2022, doi: 10.1145/3439950.

[9] B. Li, X. Peng, Q. Xiang, H. Wang, T. Xie, J. Sun, and X. Liu, "Enjoy your observability: an industrial survey of microservice tracing and analysis," Empirical Software Engineering, vol. 27, no. 1, art. 25, 2022, doi: 10.1007/s10664-021-10063-9.

[10] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in Proc. Adv. Neural Inf. Process. Syst. 35, 2022, pp. 24824–24837, doi: 10.48550/arXiv.2201.11903.

[11] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: synergizing reasoning and acting in language models," in Proc. 11th Int. Conf. Learn. Representations, 2023, doi: 10.48550/arXiv.2210.03629.

[12] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: language models can teach themselves to use tools," in Proc. Adv. Neural Inf. Process. Syst. 36, 2023, doi: 10.48550/arXiv.2302.04761.

[13] L. Wang, C. Ma, X. Feng, et al., "A survey on large language model based autonomous agents," Frontiers of Computer Science, vol. 18, no. 6, 186345, 2024, doi: 10.1007/s11704-024-40231-1.

[14] T. Guo, X. Chen, Y. Wang, et al., "Large language model based multi-agents: a survey of progress and challenges," in Proc. 33rd Int. Joint Conf. Artif. Intell. (IJCAI), 2024, pp. 8048–8057, doi: 10.24963/ijcai.2024/890.

[15] Q. Zhang, C. Fang, Y. Xie, et al., "A survey on large language models for software engineering," arXiv:2312.15223, 2024, doi: 10.48550/arXiv.2312.15223.

[16] Y. Chen, H. Xie, M. Ma, et al., "Automatic root cause analysis via large language models for cloud incidents," in Proc. 19th European Conf. Comput. Syst. (EuroSys), 2024, pp. 674–688, doi: 10.1145/3627703.3629553.

[17] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang, "Next-generation database interfaces: a survey of LLM-based text-to-SQL," arXiv:2406.08426, 2024, doi: 10.48550/arXiv.2406.08426.

[18] X. Liu, S. Shen, B. Li, et al., "A survey of text-to-SQL in the era of LLMs: where are we, and where are we going?" arXiv:2408.05109, 2024, doi: 10.48550/arXiv.2408.05109.

[19] Y. Gao, Y. Xiong, X. Gao, et al., "Retrieval-augmented generation for large language models: a survey," arXiv:2312.10997, 2024, doi: 10.48550/arXiv.2312.10997.

[20] L. Huang, W. Yu, W. Ma, et al., "A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions," ACM Trans. Inf. Syst., 2024, doi: 10.1145/3703155.

[21] P. Sahoo, P. Meharia, A. Ghosh, S. Saha, V. Jain, and A. Chadha, "A comprehensive survey of hallucination in large language, image, video and audio foundation models," in Findings of the Association for Computational Linguistics: EMNLP 2024, 2024, pp. 11709–11724, doi: 10.18653/v1/2024.findings-emnlp.685.

[22] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," Communications of the ACM, vol. 15, no. 12, pp. 1053–1058, 1972, doi: 10.1145/361598.361623.

[23] A. Cockburn, "Hexagonal architecture (ports and adapters)," 2005. [Online]. Available: https://alistair.cockburn.us/hexagonal-architecture/

[24] R. Levin, E. Cohen, W. Corwin, F. Pollack, and W. Wulf, "Policy/mechanism separation in Hydra," in Proc. 5th ACM Symp. Operating Systems Principles (SOSP), 1975, pp. 132–140, doi: 10.1145/800213.806531.

[25] M. Attaran, S. Attaran, and B. G. Celik, "The impact of digital twins on the evolution of intelligent manufacturing and Industry 4.0," Advances in Computational Intelligence, vol. 3, no. 11, 2023, doi: 10.1007/s43674-023-00058-y.

[26] A. Samuels, "Examining the integration of artificial intelligence in supply chain management from Industry 4.0 to 6.0: a systematic literature review," Frontiers in Artificial Intelligence, vol. 7, art. 1477044, 2025, doi: 10.3389/frai.2024.1477044.

[27] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (Dagstuhl seminar 16162)," Dagstuhl Reports, vol. 6, no. 4, pp. 110–138, 2016, doi: 10.4230/DagRep.6.4.110.