IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, November 2026
Impact Factor: 8.2

A Multi-Tenant Unified Observability Platform for Data Pipelines and Trading Systems

Manish Narkhede[1], Karan Nigal[2], Niraj Shevade[3], Dr. R. Y. Totare[4]
Students, Department of Information Technology[1-3]
Professor and Project Guide, Department of Information Technology[4]
AISSMS Institute of Information Technology, Pune, India

Abstract : *Observability tools in workflows, distributed batch processing, microservices and stream processing are typically instrumented separately, increasing debugging complexity and tool overhead. This paper describes Capsule; an open source observability platform which aggregates metrics, logs and traces from Apache Airflow, PySpark, multi-language microservices, and Kafka based trading workloads. It is designed as a multi-tenanted system; multiple teams can create projects and be issued scoped keys, with instrumentation provided via auto-generated config or simple language-specific SDKs built on top of open telemetry. All data across the different teams is isolated using a tenant aware API gateway. Query results are cached and presented in a unified web interface. An optional support layer is built atop the telemetry to detect anomalies within data sources, explain incidents using only collected data, and perform low risk remediation under a hierarchical and auditable policy. This paper describes the rationale behind Capsule, it's architecture, onboarding process, support features, deployment strategy, target users, and future roadmap.*

Keywords: Observability, Multi-Tenancy, OpenTelemetry, Data Engineering Pipelines, Apache Kafka, Incident Diagnosis, Large Language Models, Cloud-Native Deployment.

I. INTRODUCTION

Observability in modern data and financial systems often requires multiple tools. These may include workflow management tools (Airflow), distributed computation tools (PySpark), microservices and an event-based system (Kafka) to transport market/transaction data. Metrics, logs and increasingly traces can be obtained from all of these systems. However, each tool has its own way of presenting observability data.

In practice, this can lead to fragmented observability. When a daily pipeline is late, an engineer may have to check the scheduler, spark, log search and broker dashboards in order to find out why. Commercial solutions exist to overcome this problem, however they are generally costly, with costs dependent on number of hosts, data ingested, or data storing time. This makes such solutions unfeasible for many smaller teams, labs and learning environments. Open source solutions reduce this cost; Prometheus, Elasticsearch and Grafana can be used together to provide observability data. However this results in a number of loosely connected dashboards, and the correlation between dashboards depends on the care taken by the person setting up the observability infrastructure [1]-[3].

Typically, within self-hosted observability environments, observability systems will be implemented once per team. This causes redundancy in operations, and makes it difficult for larger departments or incubators (or colleges) to provide collective observability. In order for multiple independent groups to share an observability platform, while connecting their own systems using only a few steps, and without having visibility of any other group's data, a solution is required.

Capsule is an open source observability solution designed to address the above problems. It collects metrics, logs and traces from Airflow, PySpark, microservices and Kafka-based trading systems. These are stored in well known open source engines. The data can then be queried with a caching API and displayed in a single UI. Capsule provides multi-

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 4, November 2026 Impact Factor: 8.2

tenancy, with automated on-boarding via API keys and configuration files. An optional support layer is provided, which identifies anomalous behaviour, infers what is happening from the evidence gathered, and undertakes low risk fixing actions following an explicit safety policy. This paper provides an overview of Capsule, including high level architecture and features. Performance metrics are not covered.

Section II reviews related work. Section III includes design objectives. Section IV outlines architecture. Section V describes multi-tenancy on-boarding. Support services are outlined in section VI, deployment in section VII, and target users in section VIII. Finally, limiting factors and future work are discussed in section IX and the paper is concluded in section X.

II. RELATED WORK

A. Research on Observability and Autonomous Operations

There has been recent work in observability and autonomous systems. Satapathy et al. present a dynamic observability system for microservices sandboxes [1], while Chakraborty et al. investigate intent driven data-flows, integrating multiple processing engines across distributed geo-clouds [3]. Further, [5] presents a unified telemetry system across domains within a computing continuum. Automation and language models have also been applied to observability. Thakar et al. coordinate AI agents to autonomously manage and heal data pipelines [2], and LEMON applies language models to monitor orchestrating microservices [4]. Overall, there is active research into systems providing dynamic telemetry and model driven autonomy. However, each investigation considers only a single type of workload or deployment.

B. Background Concepts in Capsule

Capsule leverages many well-known concepts. On-call processes are defined in Site Reliability Engineering [6], and observability tools aim to reduce the burden of these. Distributed tracing has been implemented at scale by Dapper [7]. OpenTelemetry provides a vendor agnostic tool for producing metrics, logs and traces [8]. Apache Kafka provides log stream infrastructure used by many transactional and event based systems [9]. Kubernetes provides orchestration and self-healing to containerised services [10].

Within the assistive layer, there are known risks. Language models may generate coherent but incorrect information (hallucinations) [11], however this can be reduced by retrieving grounded information [12]. Tool calling models can introduce undesired side-effects into real systems [13]. Finally, if instructions are embedded in data (such as inserting a malicious log entry), the model may be tricked [14]. If multiple organisations share a single platform, multi-tenancy introduces challenges in terms of maintenance and separation [15].

C. Place in the Landscape

Capsule is positioned alongside two categories of tool (Table I). This is a qualitative comparison based on design intent.

TABLE I: Qualitative Comparison of Monitoring Approaches

Aspect	Commercial SaaS suites	Self-assembled open-source stacks	Capsule (design intent)
Unified view of pipelines and trading	Available through integrations	Separate consoles per tool	One dashboard over metrics, logs and traces
Cost model	Usage-based licence	Infrastructure and set-up effort	Infrastructure only; open source
Data location	Vendor-hosted	Self-hosted	Self-hosted
Multi-tenant sharing	Managed by the vendor	Manual configuration	Built in; enforced at the gateways

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, November 2026
Impact Factor: 8.2

Aspect	Commercial SaaS suites	Self-assembled open-source stacks	Capsule (design intent)
Assisted diagnosis and action	Vendor-specific features	Not included	Evidence-bound, tiered and auditable

III. DESIGN GOALS

Unified support: Support for orchestration pipelines, batch processing, microservices, and streaming services in one system. [1, 2] Cost-effective and open: All core components are open source and self-hosted. Costs therefore depend entirely on infrastructure costs. [3] Isolated multi-tenancy: Multiple tenants can exist within one deployment. However, telemetry data from one tenant cannot be read by another. [4] Standardised: Data ingests using the OpenTelemetry standard and common exporters, so instrumentation can easily be adapted from existing systems. [5] Fast querying: Repeated querying from dashboards can be quickly resolved with caching. [6] Controlled guidance: Guidance and correction is done using evidence, a limited set of actions, and requiring user approval for higher risk actions. [7] Reproducible: Can be run locally with just one command, and deployed in the cloud from code.

IV. SYSTEM ARCHITECTURE

A. Layered Organisation

Capsule consists of 7 distinct layers (Fig. 1). Telemetry data flows from workloads, through the collector and tenant aware gateway, before being stored. Querying occurs in the opposite direction. Queries from the dashboard are sent to the access layer. They are then routed to both the cache and the stores. The guidance layer works alongside the access layer. Queries are sent to it in the same way as to the user, and corrective actions are returned to the workloads. Following this, the guidance layer sends queries to collect telemetry data to verify the effect of the corrective actions.

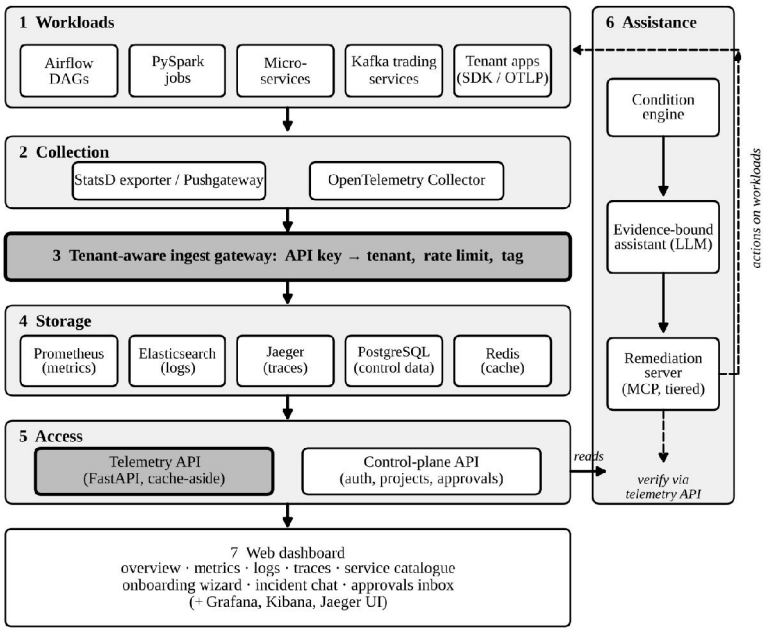


Fig. 1. Layered organisation of Capsule. Solid arrows show telemetry and query flow; dashed arrows show the corrective loop of the assistance layer.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, November 2026
Impact Factor: 8.2

TABLE II: Layers and Principal Components

Layer	Principal components	Purpose
Workloads	Airflow, PySpark, Python / Java / Node.js microservices, Kafka producers and consumers	Generate telemetry
Collection	OpenTelemetry Collector, StatsD exporter, Prometheus Pushgateway	Receive and normalise signals
Ingest gateway	API-key validation, rate limiting, tenant tagging	Admit data per tenant
Storage	Prometheus, Elasticsearch, Jaeger, PostgreSQL, Redis	Metrics, logs, traces, control data, cache
Access	FastAPI telemetry API, control-plane API	Tenant-scoped queries and administration
Assistance	Condition engine, LLM assistant, remediation server	Detect, explain and act
Presentation	Next.js dashboard, Grafana, Kibana, Jaeger UI	Visualise and investigate
Infrastructure	Docker, Docker Compose, Kubernetes on Amazon EKS, Terraform	Packaging and provisioning

B. Collection and Ingestion

Airflow sends data to StatsD. This data includes timing information, execution statuses and heartbeat information from the scheduler. Exporters convert these stats into Prometheus metrics. PySpark applications have a short lifetime, so on completion, information about the status, duration and number of processed records are sent to a Pushgateway. Microservices leverage OpenTelemetry instrumentation to send metrics, structured log entries and distributed traces. Services using Kafka send metrics regarding producer throughput, consumer lag by partition, and number of errors. All data is sent to an OpenTelemetry Collector. From here, metrics, logs and traces are passed to the corresponding storage. As discussed in section V, the tenant aware gateway means data can only be sent from sources with a valid key.

C. Storage and Query Access

Metrics, logs and traces are respectively stored in Prometheus, Elasticsearch and Jaeger. Retention periods depend on their respective volume. Control plane data, including organisations, projects, keys, approvals and audit information, is stored in PostgreSQL. Data is only retrieved from the telemetry API (written using FastAPI). To reduce latency, a cache aside pattern is used. If the desired data is not found in Redis, it is fetched from the data store and cached for a short duration. Endpoints exist for retrieving metrics (filtered by component and time period), consumer lag from Kafka, searches in logs and traces, health checks and a summary payload to allow quick population of the dashboard. Metrics are published from the API allowing self observation.

D. Dashboard

Using Next.js, this dashboard provides an overview of service and alert status. It provides metrics filtered by component and time period, log searching with filters on log severity, and navigation to traces from log lines. It also provides trace exploration via waterfall graphs. Discovery of services is provided based on ingested telemetry data. There is also onboarding, an incident chat, approvals inbox and a time line of automated actions. For deeper analysis, Grafana, Kibana and the Jaeger UI can be accessed from the dashboard.

V. MULTI-TENANT ONBOARDING AND INTEGRATION

A. Tenancy Model

There are four tenancy related objects in Capsule. ‘Organisation’ relates to companies/teams. Projects represent an isolated unit of tenancy, meaning that each project equates to one tenant. Users can be assigned to projects with

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, November 2026
Impact Factor: 8.2

different roles (owner, admin, viewer), meaning that users can have varied roles between different projects. Telemetry sources are authenticated using API keys. Keys are associated with one project, have predictable prefixes, are displayed once, stored as hashes, and can be revoked/rolled.

B. Isolation Principles

Isolation follows the following principles: Tenancy information is obtained server side from either the API key (ingestion) or authenticated session (queries). It should never come from the client. All metrics, logs and traces contain tenant information as tags. All queries from the telemetry API include filtering by the tenant of the calling user prior to reaching the data store. Cache keys include the tenant as a prefix. Rate limiting and quotas are implemented per tenant, ensuring that one high-volume source does not negatively affect other tenants. As separation is enforced by gateways rather than data stores, simplicity was prioritised and existing data stores were used, rather than native multi-tenancy data stores. These are considered as future work.

C. Onboarding Workflow

Onboarding should be quick, taking a few minutes (see Fig 2). Users create an organisation, create a project and receive an API key. If control over who submits data is required from an operator, projects can be created in a pending state. Once approved by an administrator, keys provided to projects in pending state will become active. The user selects which stack they wish to integrate. Configuration is generated for the user to copy, with endpoint, key and tenant information pre-populated. In real-time, once the first telemetry with the given project is received, the check turns green. A default dashboard for the selected stack is created, meaning the user can view populated data upon navigating to the dashboard.

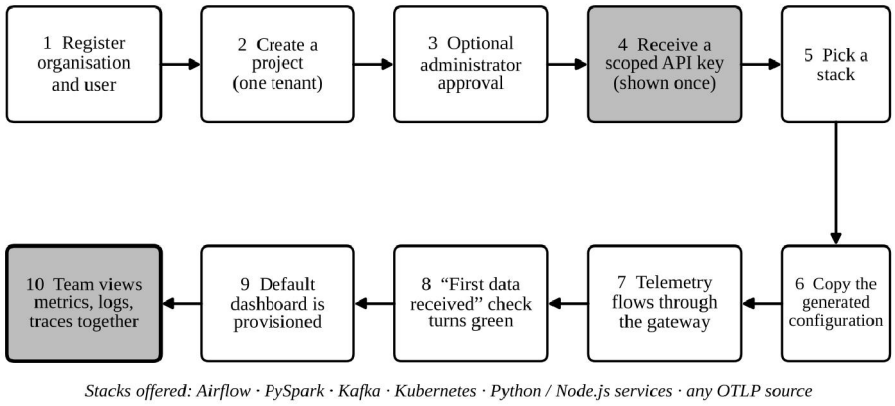


Fig. 2. Self-service onboarding of a tenant, from registration to the first populated dashboard.

D. Integration Catalogue and SDKs

Table III lists the integration paths offered by the onboarding wizard. For application code, thin SDKs wrap the official OpenTelemetry libraries so that a service is connected with a single initialisation call that sets the endpoint, API key, service name and environment. The first SDKs target Python and Node.js, with Java and Go planned. Because the SDKs add no proprietary protocol, teams that already use OpenTelemetry can connect by changing only an endpoint and a header.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, November 2026
Impact Factor: 8.2

TABLE III: Integration Paths Offered to Tenants

Source	Integration mechanism	Signals captured
Apache Airflow	Native StatsD emission via the StatsD exporter	Task durations, DAG run states, scheduler heartbeat
PySpark	Job-level push to the Pushgateway	Job status, runtime, records processed
Microservices	OpenTelemetry SDK or Capsule SDK wrapper	Request metrics, structured logs, traces with context propagation
Kafka services	Exporter or application-level metrics	Throughput, partition-level consumer lag, error counts
Kubernetes workloads	Collector configuration generated per project	Pod and namespace metrics, container logs
Any OTLP source	Standard OpenTelemetry environment variables	Metrics, logs and traces

VI. ASSISTANCE SERVICES

Layer [optional] to reduce time to resolution and prevent uncontrolled access to production systems.

A. Condition Engine

Rules are evaluated based on recent telemetry data. For example, if error rate is above a threshold, consumer lag exceeds a threshold, or there is a spike in error logs. All evaluations (triggered or not) are logged meaning rule behaviour is observable. If a rule triggers, an investigation begins.

B. Evidence-Based Assistant

This investigation is done using an assistant using Gemini API with function calling. The assistant is instructed to only make statements about the current system status after querying for data through telemetry tools. Each factual statement is accompanied by the reference within the data used. If the query returns no relevant data, the assistant should state that it has no data. The tools are specific to each tenant and make the same API calls as in the dashboard. This means that log lines, trace attributes and alerts retrieved from telemetry tools are all wrapped in analysis material and cannot be interpreted as instructions. This prevents unintended impact from injected text [14]. A small vector database containing runbooks and previous incident notes helps to answer questions that cannot be answered by real-time telemetry. For example, what is the normal process for high consumer lag? [12]. Questions can also be asked via the incident chat in the dashboard.

C. Ranked Resolution

Suggestions are not applied by the model, but passed to a resolution server. This is implemented as a Model Context Protocol server. It provides a fixed set of tools, and a ranking is set upon definition of the tool (see Table IV). The ranking is inherent to the tool, and therefore cannot be increased or decreased by prompts [13].

TABLE IV: Remediation Tiers

Tier	Typical actions	Handling
1	Restart a service, clear a cache, return replicas to baseline	Executed automatically; reversible and limited to one tenant
2	Roll back a deployment, change a configuration value	Recorded as a pending request with reasoning and references; executed only after a person approves
3	Destructive or irreversible operations	Not offered as tools at all

D. Verification, Limits and Audit

After an automatic action the engine reads telemetry again to check that the condition has cleared. If it has not, the action may be repeated only up to a configured limit, after which a circuit breaker stops further automatic attempts and

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, November 2026
Impact Factor: 8.2

escalates the incident to a person. Every proposal, approval, execution and verification outcome is written to an audit trail and shown as a timeline in the dashboard, so that reviewers can see what was done, why, and on the basis of which evidence.

VII. DEPLOYMENT MODEL

All components are containerised. A simplified version of the application is provided to support development/education, with log search components removed to allow deployment on memory constrained machines. In shared circumstances, deployment is done on Kubernetes (Amazon EKS) providing scalability and robustness [10]. Infrastructure components including networking, clusters, load balancing, and managed caching, are defined using Terraform. This allows easy review, creation, and destruction of environments. Thus ensuring costs in the cloud are predictable. Deployment is currently limited to 1 region within AWS. Secrets can be passed in via configuration, or via a secret management system. They are not stored in source control.

Sample workloads have been included to allow demonstration of this platform without access to live systems. These include Airflow DAGs with realistic variation, PySpark processes on synthetic transactions, order/inventory microservices, and a variable speed trading simulation using Kafka. Workloads also act as teaching resources and can produce reliable sources of issues for the assistant layer.

VIII. INTENDED USERS AND USE CASES

Data engineers can monitor DAGs and processes. If a process has failed, information relating to that process can be found in one place. Site Reliability Engineers and DevOps Engineers can correlate errors, traces, and lag in brokers, and query the assistant for a data-informed explanation of an issue. Engineers working on a trading platform can monitor the speed of events, lag in consumers, and count of errors across streaming services. Students and Educators can learn observability in a reliable environment. Multiple projects can be run from a single deployment, allowing multiple student teams to work on separate projects. Small organisations can monitor systems in a unified way without paying per host.

Four example use cases include: resolving an erroneous pipeline run, identifying the cause of increasing consumer lag, adding a new service quickly with automatically generated configuration, and running a deployment serving multiple teams, with each team in a separate project.

IX. SCOPE, STATUS AND FUTURE WORK

In this platform, metrics, logs and traces are handled for the four workload types. A control plane is present for managing tenants within the platform, as well as the assistant layer. The trading workload is simply a simulation of load patterns in event driven trading applications, rather than a trading location. Only one region of cloud deployment is supported in the initial release, with isolation occurring at the gateway level. Performance metrics such as latency, data ingestion correctness, cost, and effectiveness of automatic diagnostic are outside the scope of this summary.

Future features include anomaly detection, more alerting mechanisms, additional alert delivery methods, storage backends with built in tenancy, additional SDKs for more languages, multi-region deployments, a greater variety of faulty scenarios to test diagnostic effectiveness, and per-tenant customisation of retry limits.

X. CONCLUSION

When pipelines, services and streaming applications are run in conjunction, separate tools must often be used for monitoring. The capsule platform provides a single open source tool for viewing metrics, logs, and traces from Airflow, PySpark, microservices, and Kafka-based workloads. Multiple autonomous teams can connect their systems using automatically generated configurations and OpenTelemetry based SDKs. Data from each system remains isolated between gateways. An optional assistant layer demonstrates how a language model can be used for diagnosis and standard recovery, with constraints including retrieving supporting data, a set list of actions to perform, and manual

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, November 2026
Impact Factor: 8.2

consent to riskier steps. This platform is aimed at data engineers, reliability teams, students, and small organisations who want a unified monitoring solution without paying for monitoring in addition to infrastruture.

ACKNOWLEDGMENT

The authors thank the Department of Information Technology, AISSMS Institute of Information Technology, Pune, for the facilities and guidance provided during this work.

REFERENCES

1. U. Satapathy *et al.*, "XPLOG: A dynamic observability framework for distributed sandboxed microservices," *IEEE Trans. Services Comput.*, vol. 19, no. 1, pp. 794–807, Jan.–Feb. 2026.
2. D. Thakar, V. Bhatt, and M. Vidyanand, "Next-generation data pipelines: Coordinated AI agents for autonomous management, optimization and self-healing," in *Proc. 2026 Int. Conf. AI x Data and Knowl. Eng. (AIxDKE)*, 2026, doi: 10.1109/AIxDKE67294.2026.00025.
3. A Chakraborty *et al.*, "Intent-driven multi-engine observability dataflows for heterogeneous geo-distributed clouds," in *Proc. 2024 IEEE 17th Int. Conf. Cloud Comput. (CLOUD)*, 2024, doi: 10.1109/CLOUD62652.2024.00014.
4. M. Götz and H. Baghban, "LEMON: LLM-enabled monitoring for microservices orchestration," *IEEE Trans. Services Comput.*, 2026.
5. "Cross-domain telemetry architecture: Dynamic observability in the computing continuum," *Computing*, Springer, 2026.
6. B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly Media, 2016.
7. B. H. Sigelman *et al.*, "Dapper, a large-scale distributed systems tracing infrastructure," Google Inc., Mountain View, CA, USA, Tech. Rep. dapper-2010-1, Apr. 2010.
8. OpenTelemetry Authors, "OpenTelemetry specification," Cloud Native Computing Foundation. [Online]. Available: https:*opentelemetry.io*
9. J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proc. NetDB Workshop*, 2011.
10. B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70–93, 2016.
11. Z. Ji *et al.*, "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–38, 2023.
12. P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020.
13. S. Yao *et al.*, "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.
14. K. Greshake *et al.*, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *Proc. 16th ACM Workshop Artif. Intell. Secur. (AISec)*, 2023, pp. 79–90.
15. C.-P. Bezemer and A. Zaidman, "Multi-tenant SaaS applications: Maintenance dream or nightmare?" in *Proc. Joint ERCIM Workshop Softw. Evol. and Int. Workshop Principles Softw. Evol.*, 2010, pp. 88–92