ASSETOPSBENCH: BENCHMARKING AI AGENTS FOR TASK AUTOMATION IN INDUSTRIAL ASSET OPERATIONS AND MAINTENANCE

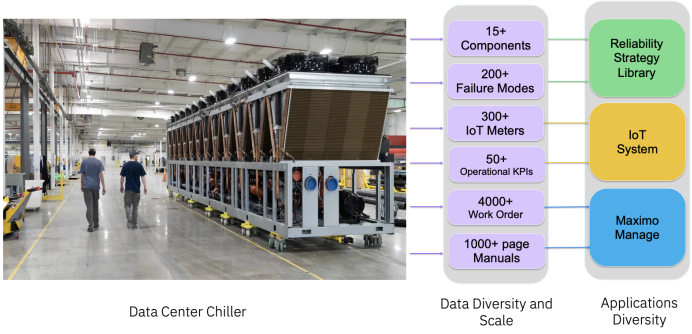
Anonymous authors
Paper under double-blind review

ABSTRACT

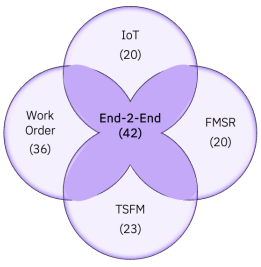
AI for Industrial Asset Lifecycle Management aims to automate complex operational workflows, including condition monitoring, maintenance planning, and intervention scheduling, thereby reducing human workload and minimizing system downtime. Traditional AI/ML approaches have primarily tackled these problems in isolation, solving narrow tasks within the broader operational pipeline. In contrast, the emergence of AI agents and large language models (LLMs) introduces a next-generation opportunity: enabling end-to-end automation across the entire asset lifecycle. This paper envisions a future where AI agents autonomously manage tasks that previously required distinct expertise and manual coordination. To this end, we introduce AssetOpsBench, a unified framework and environment designed to guide the development, orchestration, and evaluation of domain-specific agents tailored for Industry 4.0 applications. We outline the key requirements for such holistic systems and provide actionable insights into building agents that integrate perception, reasoning, and control for real-world industrial operations.

1 INTRODUCTION

Industrial assets, such as data center chillers (Naug et al. (2024)) and wind farms (Monroc et al. (2024)), are complex, multi-component systems that generate vast amounts of multimodal data, including time-series sensor readings, textual inspection and workorder records, operational logs, and images, throughout their lifecycle. The ability to monitor and interpret heterogeneous data from diverse sources, such as IoT SCADA (Supervisory Control and Data Acquisition) sensors, operational KPIs, failure mode libraries, maintenance work orders, and technical manuals, is key to effective Asset Lifecycle Management (ALM). However, subject matter experts such as maintenance engineers, site operators, and plant managers face considerable challenges in synthesizing insights from these disparate data streams to support timely and condition-aware decisions. As highlighted in Figure 12, the scale, semantic diversity of assets, and application-specific contexts often render traditional monitoring and management systems inadequate.



(a) Complex Industrial Asset – Data Centers managing Chiller and Air Handling Units (AHUs)



(b) Distribution of open-sourced scenarios for benchmarking agents in a simulated environment.

To address these challenges, the research and industrial communities are increasingly turning to AI agents: autonomous and goal-driven systems capable of integrating data across silos, reasoning over complex conditions, and triggering acti mkons. AI agents are particularly promising in the context of Industry 4.0, where the confluence of real-time IoT telemetry (e.g., Oracle IoT Oracle (2025), enterprise asset management (EAM) systems, and IBM Maximo IBM) and reliability engineering frameworks necessitates scalable and intelligent automation. These agents promise to support a wide range of industrial workflows, from anomaly detection to maintenance scheduling, by bridging the gap between raw sensor data, maintaiance report, work-order and business-level insights.

Despite recent advances in agent-based systems, such as ReAct Yao et al. (2023), HuggingGPT Shen et al. (2023), Chameleon Lu et al. (2023), and Generalist Agents Fourney et al. (2024); Marreed et al. (2025), a gap remains in adapting these innovations for real-world industrial settings. Most recent domain and application specific benchmarks (e.g., ITBench Jha et al. (2025), SWE-bench Chan et al. (2025), $\tau-$bench Yao et al. (2024) and its extension Fu-Hinthorn (2025), Customer Support Benchmarks Team (2025)) are tailored toward machine learning, IT or customer-service domains and do not address the unique challenges of industrial applications, such as data modality diversity(time series and text), business object complexity(e.g., failure mode, work orders, asset hierarchies), and task collaboration across multiple operational personas (reliability modeling by expert and time series modeling based on data scientist).

This paper introduces **AssetOpsBench**, the first dataset and benchmarking system designed to evaluate AI agents for real-world industrial asset management tasks. By leveraging experts in development, we have carefully built real multi-source datasets, intent-aware scenarios, and domain-specific agents to develop, evaluate, and compare multi-agent systems. Our system includes:

- A catalog of **domain-specific AI agents**, including an IoT agent, a failure mode to sensor mapping (FMSR) agent, a foundation model-driven time series analyst (TSFM) agent, and a work order (WO) agent. Each agent has tools and targets different modalities and tasks.
- A curated to be open-source **intent-driven** 141 scenario of **human-authored natural language queries**, grounded in real industrial data center operations (Figure 1b), covering tasks such as sensor-query mapping, anomaly detection, failure diagnosis, and work-order modeling.
- A **simulated industrial environment** based on a CouchDB-backed IoT telemetry system and **real** multi-source dataset, enabling end-to-end benchmarking of multi-agent workflows and open source contributions without the constraints associated with production systems
- A comparative analysis of multi-agent architectural paradigms: **Agent-As-Tool** vs. **Plan-Executor**, highlighting tradeoffs between interleaved decomposition or decomposition-first
- A three-pronged evaluation consisting of (i) an LLM-based rubric, (ii) reference-based scoring of task decomposition and execution, and (iii) manual expert verification for certain scenarios.
- A systematic procedure for the **automated discovery** of emerging failure modes in multi-agent systems, extending beyond fixed taxonomies and its benefits.

A key insight from our study is that domain complexity in industrial settings necessitates a **multi-agent approach**, where specialized agents are developed for isolated tasks, then orchestrated to solve composite problems. For example, sensor data may be handled by an IoT agent, while fault history is managed by an FMSR agent. These agents must collaborate intelligently to answer user queries, such as "Why is the chiller efficiency dropping?", which blend physical reasoning, historical correlation, and operational semantics. Furthermore, the design of agent workflows must respect the **natural language and intent patterns** used by industrial end users. Unlike IT users, operators and engineers often refer to assets in physical or operational terms (e.g., "chiller performance","oil temperature spike") rather than referring to database fields or ontologies. Crafting robust benchmarks requires capturing this domain-specific linguistic variance, ensuring agents not only retrieve correct answers but also follow reasoning patterns aligned with domain expectations.

Finally, we experimented with an additional closed-source **162 scenarios** to demonstrate generality, spanning 10 asset classes, 53 failure modes, and 20 sensors. These include 42 live-deployment scenarios (>90% correctness verified by a domain expert), 17 hydraulic system, 15 metro train, and 88 failure-mode scenarios encompassing diverse asset–failure–sensor relationships.

2 Related Work

Generalist Agents. The development of generalist agents capable of orchestrating multiple sub-agents to accomplish complex tasks has emerged as a prominent research direction. This paradigm is evident across various domains, including web systems such as Magentic Fourney et al. (2024) and CUGA Marreed et al. (2025), multimodal agents like GEA Szot et al. (2024), and software engineering platforms like HyperAgent Huy et al. (2025), ChatDev Qian et al. (2024), and MetaGPT Hong et al. (2024). These agents typically employ predefined sets of sub-agents, such as terminals, browsers, code editors, and file explorers, each assigned specific functional roles to facilitate task decomposition and planning. While this architecture enables targeted integration and task specialization, it often lacks flexibility. Most systems adopt hard-coded reasoning paradigms, such as plan-executor or ReAct, which limit their capacity to support new agents, adapt to novel task, or alternative coordination strategies, such as AOP Li et al. (2025) and Prospector Kim et al. (2024).

Domain-Specific Agents. Solving specialized tasks often requires domain-specific capabilities, prompting the development of tailored benchmarks such as MLEBench Chan et al. (2025) and MLAgentBench Huang et al. (2024) Arena. These frameworks evaluate agents on a diverse set of machine learning problems, such as classification and regression, across multiple modalities, including tabular and image data. They simulate end-to-end workflows, from resolving GitHub issues to automating model training and evaluation pipelines. The concept of the *AI Research Agent* has gained traction, referring to agents built for scientific discovery and iterative experimentation. For example, MLGym Nathani et al. (2025), a research agent in machine learning workflows. However, most current benchmarks lack support for temporal and text data modalities together, which are crucial in domains such as physical asset health monitoring.

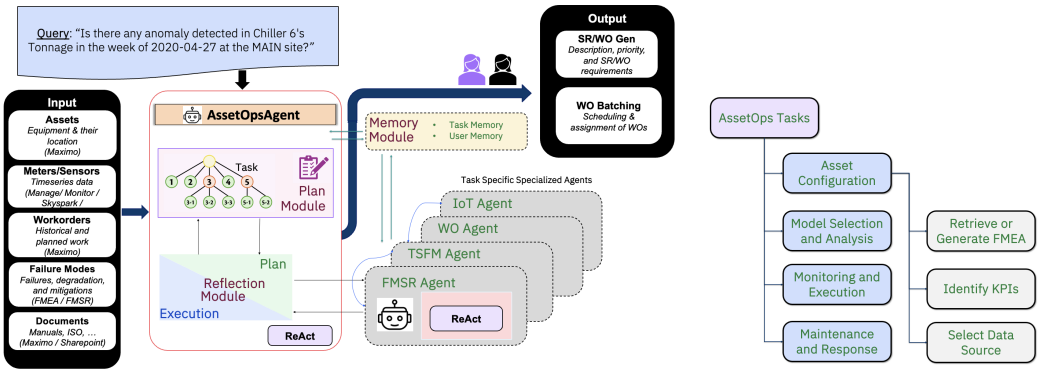
Application-Specific Agents. Agent-based automation is also advancing in operational settings, such as IT operations, customer support, and compliance monitoring. Frameworks developed under initiatives like ITBench Jha et al. (2025) and AIOpsLab Chen et al. (2025) aim to replicate real-world scenarios involving site reliability engineering, diagnostics, and system auditing. These systems reinforce the importance of application-specific benchmarks, tailored to specific personas, that not only evaluate agents across structured tasks but also expose capability gaps and drive innovation in reasoning and orchestration strategies. Current benchmarks in this space tend to be domain-specific in scope, lacking the generality and composability required to assess agent performance across diverse, multi-agent environments, especially those involving cross-modal reasoning or domain-specific tool usage.

Fine-Tuned and Compact Models. Complementary to architectural advances, recent work has focused on improving agent performance via fine-tuned language models. So-called *Large Action Models (LAMs)* are designed to execute structured actions within environments, often trained on large-scale datasets to support planning, sequencing, and low-level execution. Systems such as TaskBenchShen et al. (2024), xLAMZhang et al. (2025b), AgentGen Hu et al. (2025), AgentBank Song et al. (2024), AgentRM Xia et al. (2025), FireAct Chen et al. (2024), and ActionStudio Zhang et al. (2025a) exemplify this trend. These models are frequently trained in grounded settings, e.g., Windows-based environments Wang et al. (2025) and evaluated across diverse task categories including arithmetic, programming, and web-based interaction. While effective, these approaches are limited to textual or web environments and have not yet demonstrated broad applicability to more complex or industrial automation tasks involving hybrid agent compositions.

Open Challenges. Despite these advances, several gaps remain. First, there is a lack of comprehensive benchmark datasets targeting industrial asset domains, particularly those involving condition-based monitoring, predictive maintenance, automated diagnostics, and work order planning. To support this claim, we analyzed **a catalog of 135 public datasets** jonathanwvd (2025) and found that only one dataset includes any form of work-order or operational context, and even that lacks sensor history. Moreover, only 53 datasets mention failure modes, most of which contain just one or two modes, and none of the datasets support agentic applications. Second, time-series data, which plays a central role in industrial and infrastructure-related applications, remains underrepresented in existing agentic benchmarks. Finally, few systems support orchestration across heterogeneous agents, including those based on text, code, or simulations, nor do they offer modular reasoning strategies adaptable to complex, multi-agent workflows. Addressing these gaps is essential to advance general-purpose agent intelligence in high-stakes, real-world domains.

3 Problem Definition: Intelligent Agent-Based Asset Operations

AssetOpsBench aims to establish a generalist agent framework for managing the lifecycle of physical assets, integrating multiple domain-specific agents and a suite of application-specific tasks for systematic evaluation. It encompasses a comprehensive set of **operational** and **analytical** tasks that arise across the asset lifecycle. The benchmark focuses on scenarios commonly posed by domain experts, such as maintenance engineers, reliability specialists, and facility planners, who translate operational needs into data-driven actions. These scenarios cover key tasks including anomaly detection, root cause analysis, fault explanation, predictive maintenance planning, work order bundling, and service request initiation. For example, a user might request: "Help configure an anomaly detection model to monitor power consumption of `CUXP`. Trigger alerts when usage is projected to exceed 8 Watts above the maximum deviation observed over the past 30 days". Such task enables timely corrective actions such as "service request creation" to mitigate potential issues.



(a) Architecture of the Multi-Agent System: Time Series (TSFM) Agent, Failure Mode Sensor Relations (FMSR) Agent, Work Order (WO) Agent

(b) Exemplar AssetOps Task Hierarchy: Detailed hierarchy is given in Appendix B

Figure 2a illustrates the foundational components of our proposed framework. At the core is the **AssetOps** Agent, which functions as a global coordinator. It interprets high-level user queries expressed in natural language, decomposes them into structured subtasks, delegates these to specialized functional sub-agents (e.g., IoT, TSFM), and integrates their outputs into coherent responses. The architecture supports on-demand instantiation of agents, dynamic task planning, and reactive execution, capabilities essential for operating in complex, variable industrial environments. Formally, given a query or task $\tau \in \mathcal{T}$, the objective is to generate a valid plan $\pi \in \Pi$, leverage memory M to propagate relevant context, coordinate appropriate agents $A_i \in \mathcal{A}$ for task fulfillment, and produce an output $o \in \mathcal{O}$ that aligns with the intended goal and operational constraints. For brevity, Appendix A.1 discusses the mathematical formulation of agent-oriented planning, and Section A.2 provides details on all four agents.

In this paper, we leveraged ISO documents to build a structured task taxonomy aligned with the stages of the physical asset management ISO-2024 (2024); ISO (2016). Such a taxonomy provides a consistent and scalable approach to scenario generation for benchmarking. We refer to this approach as **intent-driven** scenario generation, rather than **API-driven** scenario generation, as popularized in Yao et al. (2024); Shen et al. (2024). As illustrated in Figure 2b, the taxonomy begins with **Asset Configuration**, encompassing activities such as retrieving Failure Mode and Effects Analysis (FMEA) documentation and selecting performance KPIs, typically carried out by reliability engineers. It progresses to **Model Selection and Analysis**, where data scientists apply anomaly detection models (e.g., Time Series Foundation Model, ML Models) and use LLM-powered retrieval to surface relevant historical failures. In the **Monitoring and Execution** phase, operations teams manage live telemetry, refine detection models, and enforce safety guardrails. Finally, the **Maintenance and Response** phase focuses on actionable outputs: generating work orders, summarizing system health, and prioritizing interventions—tasks typically handled by maintenance engineers. Defining **tasks** and **APIs** based on a standard is key, as it generalizes various different application software Oracle (2025); IBM.

4 ASSETOPSBENCH

AssetOpsBench comprises a real multi-asset, multi-source dataset (Section 4.1) from a data center, 141 manually constructed task scenarios (Section 4.2), and a benchmarking environment that includes novel task-specific AI agents and an evaluation framework (Section 4.3). The scenarios are developed in collaboration with SMEs and reflect the essential day-to-day capabilities that agents operating in realistic industrial settings are expected to possess. We also compare scenario counts with prior benchmarks (see Appendix Table C.3), where human-authored scenarios typically contain about 100 scenarios, while LLM-generated benchmarks display greater variability in scale.

4.1 MULTI-SOURCE DATASET

A key distinguishing feature of **AssetOpsBench** is its integration of richly structured, expert-curated multi-source data that reflects the complexity of real-world industrial asset operations. Unlike a simple data-gathering effort, constructing this benchmark required extensive data cleaning, the development of a novel failure taxonomy, and careful alignment across heterogeneous sources. As shown in Table 1, the benchmark includes over 2.3 million sensor data points across 6 assets (4 *Chillers* and 2 *AHUs*), capturing time-series signals such as *chiller return temperature*, *load percentage*, and *condenser water flow*. The structured failure models, derived from Failure Mode Effects Analysis (FMEA) records, encompass 53 failure entries across three equipment assets. FMEA provides provide detailed insights into the physical locations of failures, degradation mechanisms (such as *wear* and *erosion*), and the influencing factors (including *runtime*, *fluid conditions*, and *shock loading*) that contribute to each failure. Work order histories span 4.2K records across 10+ assets and incorporate ISO-standard failure codes, event timestamps, and linkages to alerts and detected anomalies.

Table 1: Key data modalities with 3 Example Fields used for open source scenario construction

Data Source	Field	Description
Sensor Data*	Chiller Return Temp.	Measures temperature of water returning to chiller
# Industrial Assets: 6	Chiller % Loaded	Indicates current load as a fraction of the maximum
Quantity: 2.3M points	Condenser Water Flow	Indicates the current flow rate through the condenser
FMEA	Failure Location / Comp.	Subsystem/part where failure occurs (e.g., bearings,)
# Industrial Assets: 3	Degradation Mechanism	Physical process driving failure (e.g., wear, erosion)
Quantity: 53 records	Degradation Influences	Stressors like runtime, fluid quality, or shock loading
Work Orders	ISO Failure Code	Standardized classification of the failure category.
# Ind. Assets: 10+	Event Log Timestamp	Time-marked entry recording an operational event
Quantity: 4.2K records	Linked Anomaly / Alert	References to alerts or anomalies tied to work order

Additionally, the operational system generates a temporal sequence of alarm logs and also leverages domain-specific technical rules obtained from experts, enabling contextual grounding of operational anomalies. This diverse data foundation, comprising 9 modalities, facilitates a comprehensive evaluation of decision-making, tool usage, and multi-hop reasoning in industrial environments. Full dataset description is provided in Appendix D.

4.2 SCENARIO DESIGN AND COVERAGE

Each scenario in **AssetOpsBench** represents a structured operational query grounded in the lifecycle-aligned task taxonomy (Figure 2b) and asset-specific datasets (Table 1). Each scenario is formalized as:

$$P = \langle id, type, text, category, form \rangle$$

where *id* is a unique identifier; *type* specifies the task type (e.g., knowledge retrieval, analytical); *text* is the natural language query; *category* denotes the operational domain (e.g., IoT, FMSR, TSFM, WO or End-2-End (i.e., more than one agent)); and characteristic *form* defines the expected output (e.g., explanation, API call, action plan). Scenarios are categorized into two types: (1) single-agent utterances, which only require probing a single specific agent (e.g., IoT, TSFM, FMSR, WO), and (2) multi-agent tasks, which span multiple agents and require coordinated reasoning and data exchange. As shown in Figure 1b, the to be open-sourced version comprises a total of 141 scenarios, consisting of 99 single-agent and 42 multi-agent tasks.

First, we introduce two representative agents to highlight the complexity at the tool level before narrowing our focus to a specific scenario. Figure 3(a) depicts the TSFM agent, which leverages a pretrained time series foundation model from Hugging Face, and the FMSR agent, which employs an LLM to generate mappings between failure modes and sensors (get_mapping). We have over 15 tools spread across these four agents. Figure 3(b) then presents **Utterance 507**, an instructive case where a user requests a prediction of future energy consumption. To address this query, the agent must first reason about which sensor variable to use, specifically the power input, and after retrieving the data recognize that most values are zero, indicating an insufficient data condition. This scenario highlights the importance of subject matter experts (SMEs) in designing tasks that assess the reasoning capabilities of LLMs, rather than merely testing tool functionality. In its characteristic form, we further emphasize key lexical markers that also enable a semantic-based analysis.

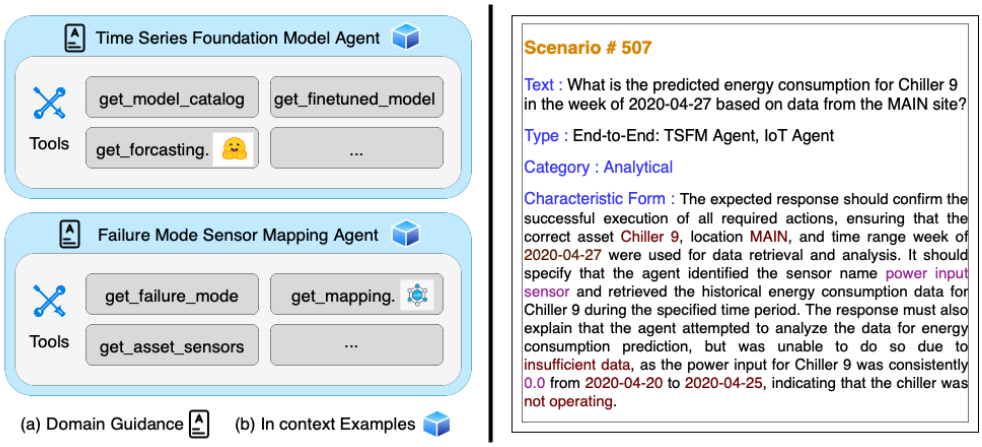


Figure 3: (a) Design of Agents with Domain Specific Guidance and Examples for In-context learning (b) Scenario 507 Example

Our dataset, particularly the work-order records, spans over **11 years** and includes rich fields such as problem codes, finish dates, and labor hours. This helps to design a scenario such as "Examine whether the year-over-year increase in corrective maintenance for CWC04009 warrants shifting resources from annual repairs toward multi-year replacement planning". Existing scenarios (IDs 407–413) support strategic work-order management tasks, including trend analysis, bundling, and probability forecasting. Overall, our scenario includes analytical reasoning (including coding, fine-tuning, and other approaches), context-aware decision-making, and language-based generalization.

4.3 Single and Multi-Agent Implementation

ReAct Yao et al. (2023) and CodeReAct Wang et al. (2024) are widely adopted baseline reasoning strategies for agent development. Three agents (TSFM, IoT, and FMSR) are built on ReAct, while the WO agent adopts CodeReAct; our framework also supports alternative strategies such as RAFA Liu et al. (2023) for extended testing. Given a mix of text- and code-based agents, it becomes necessary to introduce a global coordinator, the **AssetOps Agent**, which facilitates collaboration among these agents and can operate either under an **Agent-As-Tool** paradigm or within a **Plan-Execute** strategy.

In **Agent-As-Tool**, each individual agent is registered as a tool within a meta or supervisor agent, and the supervisor itself is instantiated using ReAct. This architecture emulates the layered decision-making found in real-world hierarchical organizations. On the other hand, **Plan-Execute**, as the name suggests, the process consists of two phases: a **Planner** and a **Reviewer**, which together generate a plan represented as a directed acyclic graph (DAG). This plan is then passed to the **Orchestrator/Executor**, which maintains a memory module that stores and transfers information between agents according to the configuration settings. To further enhance performance, we introduce an output augmentation mechanism that generates semantically enriched, self-descriptive outputs. This enables agents to trigger follow-up actions, ask clarifying questions, and make informed decisions. Read the full schema specifications in Appendix A.6.

5 EXPERIMENTS AND LEADERBOARD

To evaluate orchestration techniques across varying LLM sizes and agent-specific preferences, we adopt a **rubric-based** assessment LangChain (2025b); Wen et al. (2024); Wang et al. (2025); Andrews et al. (2025) complemented by a **reference-scoring** mechanism Yao et al. (2024); Wen et al. (2024); Cemri et al. (2025).

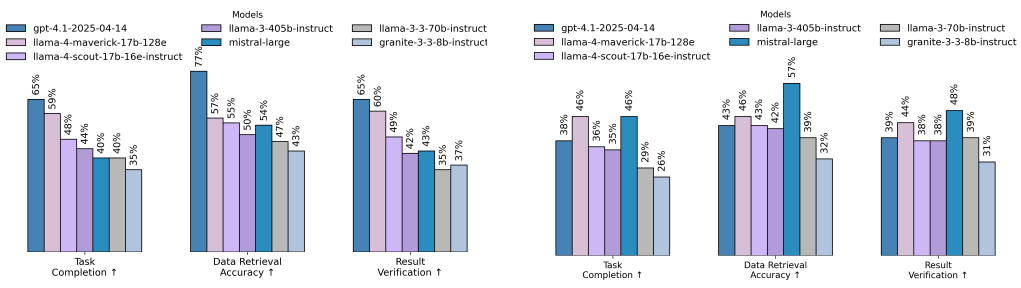
LLM-As-Judge. Each scenario is paired with a *characteristic form*, a structured specification defining both the expected final output and the intermediate reasoning or procedural steps required to achieve it. This form serves as the **soft ground truth** for evaluating agent behavior and supports rubric-based scoring with LLMs acting as judges. The evaluation rubric uses three qualitative metrics derived from experimental observations and common-sense principles. We define the **Evaluation Agent** as a scoring function that maps the original task query ($\mathcal{Q}$), the agent's trajectory output ($\mathcal{T}$, including intermediate reasoning and final output), and the characteristic form ($\mathcal{C}$, the ground-truth specification) to a set of scores (y_1, y_2, y_3). These scalar scores $(y_1, y_2, y_3) \in [0, 1]^3$ correspond to **Task Completeness** (y_1: are all required steps completed?), **Data Retrieval Accuracy** (y_2: was the correct data retrieved and used?), and **Result Verification** (y_3: is the final result logically and factually correct?). A detailed system prompt is provided in Appendix 20.

Reference-Based Scoring. For each scenario, we construct a structured ground truth (See Appendix E.1) inspired by Yao et al. (2024); Shen et al. (2024), where each entry captures the task workflow through `planning_steps` (high-level intended actions), `execution_steps` (concrete actions with corresponding inputs and outputs), and `execution_links` (dependencies between steps). This representation encodes both the logical structure and the expected outcomes. We assess an agent's **task decomposition** ability by comparing the `planning_steps` with either the thinking traces in the agent's trajectory (for Agent-as-Tool) or the DAG produced by Plan-Execute. Since agents communicate in natural language, a weighted score is employed to align action descriptions and their inputs, thereby quantifying **task execution** performance.

Experimental Setting. To quantify agent effectiveness in scenario evaluations, we adopt the Pass^k metric. Unlike the widely used Pass@k—which measures the probability that at least one of k independent attempts succeeds—Pass^k estimates the probability that an agent succeeds on *all* k attempts. This stricter criterion better captures the reliability demands of industrial applications, where retries may be impractical and consistent behavior is essential for production systems LangChain (2025a); Yao et al. (2024). In our benchmark, we report Pass^1 by default (one trial per task), as agents are executed only once per task instance. The evaluation agent, however, is executed five times to derive the performance metrics, with a sampling temperature of zero for all agents within AgentOps Agent and 0.3 for the evaluation agent. All reported results are obtained under this configuration.

5.1 ASSETOPSBENCH LEADERBOARD

Models. We conducted a series of benchmark experiments to evaluate a diverse set of language models, including closed-source models (e.g., `gpt-4.1`), frontier open-source models (e.g., `llama-4-maverick`, `llama-4-scout`, `mistral-large`, `llama-3-405b`), and medium-to-small open-source models (e.g., `llama-3-70b`, `granite-3-8b`). Currently, we have evaluated two different agentic strategies: *Agent-As-Tool* and *Plan-Execute*.



(a) **Agent-As-Tool** Approach (b) **Plan-Executor** Approach

Figure 4: Approach-wise Performance Evaluation. The order is based on the task completion rate.

Agent-As-Tool vs Plan-Execute Approach. Figure 4 shows the combined performance of both approaches using the rubric method. Overall, the Agent-As-Tool approach, as illustrated in Figure 4a, demonstrates that `gpt-4.1` leads across nearly all metrics. `llama-4-maverick` also performs competitively, particularly in result verification (60%) and clarity (78%). In contrast, smaller models such as `granite-3-8b` and `llama-3-3-70b` underperform across most dimensions. Although our benchmark did not yield promising results for small language models, as noted in a recent vision paper Belcak et al. (2025), we discuss later an interesting outcome that demonstrates the potential of combining model agency using both LLMs and SLMs for certain tasks.

A closer examination of the Plan-Execute approach (Figure 4b) shows that `mistral-large` leads overall, achieving the highest scores in task completion (46%) and data retrieval (57%). `llama-4-maverick` demonstrates balanced performance, particularly in task completion (46%) and result verification (44%), with a average data retrieval rate (46%). Interestingly, `gpt-4.1` exhibits consistent mid-range performance across all axes, suggesting potential issues with planning.

Upon deep examination, we observed that larger models generate shorter plans for the Plan-Execute approach (typically 2–3 steps) compared to the Agent-As-Tool strategy, which generally requires 5–6 steps (Appendix Section E.2). Moreover, the execution time for smaller models is nearly double, suggesting that each sub-agent takes longer to complete its assigned steps. Due to space constraints, a detailed analysis of cost and runtime is provided in the Appendix E.2 and E.13, where results are presented by partitioning the 141 scenarios into single-agent and multi-agent tasks. To further emphasize the potential of small language models (SLMs), we conducted a more in–depth task–category–level analysis. As shown in Figure 5, SLMs perform particularly well on tasks associated with the IoT Agent, while tasks related to WO and End-to-End agents still require broader improvements across all model scales.

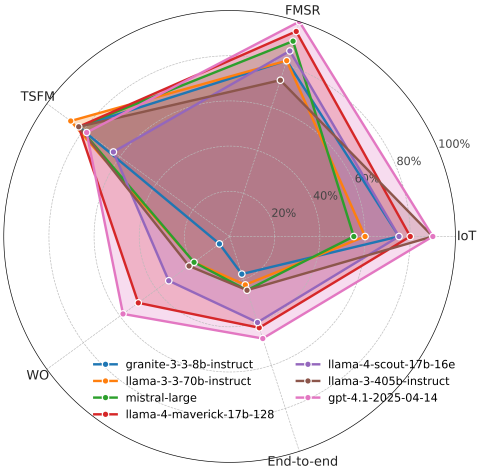


Figure 5: Agent Level Task Accomplishment with respect to Agent-As-Tool Approach

Human Validation. To assess the reliability of using LLMs as automatic evaluators for benchmarking tasks, we compare model-generated judgments against human annotations on a sample of 40 tasks. Each task is evaluated along three dimensions by four domain experts, all operating under the same information constraints as the LLMs. We use `llama-4-maverick` as the default model for conducting this study. These inter-rater reliability scores indicate substantial agreement across key evaluation dimensions, with *Data Retrieval Accuracy* showing the strongest consistency (Cohen's $\kappa = 0.79$, 90.48% accuracy). *Task Completion* ($\kappa = 0.62$) and *Generalized Result Verification* ($\kappa = 0.71$) also reflect a high level of alignment among evaluators.

Reference-Based Scoring. Focusing on the **Plan-Execute** paradigm, we observe that `mistral-large` achieves the highest performance for task decomposition: `rouge1` ≈ 0.42, `rouge2` ≈ 0.26, and `rougeL` ≈ 0.34. These results complement the LLM-based rubric findings (Figure 5(b)), providing additional evidence supporting LLM based performance checking. A detailed report is provided in Appendix E.7- E.9 to cover results on task execution score, etc. Lower scores can result from outputs that are more verbose than the ground truth.

Ablation Study. We conducted ablation experiments using the **Agent-As-Tool** method. Injecting 10 out-of-domain distractors (e.g., `SREAgent`, `EchoAgent`) into 99 single-agent scenarios unexpectedly *improved* task completion, accuracy, and reasoning, suggesting that distractors may trigger more deliberate reasoning in LLMs. In contrast, removing all in-context examples for 65 single-agent tasks (IoT+FMSR+TSFM) caused performance to collapse—from 80% to 34% for `gpt-4.1` and from 60% to 3% for `granite-3-8b`, highlighting the critical role of in-context learning for ReAct-style coordination (Appendix E.6).

5.2 Emerging Failure Modes Discovery

Trajectory analysis is critical for detecting agent mistakes, but becomes more challenging in multi-agent settings. **AssetOpsBench** extends the fixed agent failure taxonomy Cemri et al. (2025) by enabling continuous monitoring and proactive detection of failure modes in LLM-driven agents. We analyzed **881 execution trajectories** generated using the Agent-As-Tool approach and used `gpt-4.1` to estimate the distribution of 14 predefined failure modes. As shown in Table 2, **system design** emerges as a major failure source. To capture behaviors beyond this taxonomy, we allowed self-discovery of up to two **novel failure modes** per trace, revealing *emergent and compound failures* not covered by existing classifications. Among all trajectories, **185** contained one novel mode and **164** exhibited two. Common emergent failures include **Overstatement of Task Completion** (122 cases, 23.8%), **Extraneous or Ambiguous Output Formatting** (110 cases, 21.4%), and **Ineffective Error Recovery** (160 cases). Appendix F details the discovery procedure and reports how incorporating Table 2 information improves performance.

Table 2: Distribution of Failure Subcategories Across Stages of Execution

Failure Subcategory	Stage & %
System Design (Total 37.38%)	
Disobey Task Spec.	Pre: 13.87%
Disobey Role Spec.	Pre: 0.11%
Step Repetition	Exec.: 16.41%
Loss of Conversation	Pre: 0.00%
Unaware of Termination	Post: 6.99%
Agent Coordination (Total 27.52%)	
Conversation Reset	Execution: 0.00%
Fail to Ask for Clarification	Execution: **10.22%**
Task Derailment	Execution: 4.34%
Information Withholding	Execution: 2.22%
Ignored Agent's Input	Execution: 2.06%
Action Mismatch	Execution: 8.68%
Task Verification (Total 35.10%)	
Premature Termination	Pre: 3.92%
No or Incomplete Veri.	Execution: 15.56%
Incorrect Verification	Execution: 15.62%

5.3 Generality and Production Testing: 162 Scenarios

To evaluate the generality of our system, we applied it to three additional domains (air compressor, hydraulic system, etc) and one internal production use case. Our framework is field-tested to monitor 42 distinct assets across five different asset classes (Air Handling Unit, CRAC, Chiller, Pump, and Boiler), generating asset health insights ("asset needs attention") with varying complexity and temporal context. Table 3 shows a summary of the results. These results demonstrate consistent, high-quality agent behavior across core evaluation metrics. Manual review by an expert confirms that agent outputs align with expected answers in nearly all cases (up to 100%), supporting the robustness and generalization of our approach under real-world deployment conditions. Except for the Metro train, the remaining datasets have good performance, including SLM. Appendix E.15 provides a detailed discussion of example scenarios, asset coverage, and token count distribution for consideration during deployment. Our original 141 Scenarios are complex and tough, whereas Asset health is purely based on work orders and asset profile.

Table 3: Agent performance across multiple domains and scenarios.

Domain / Model	Task Completion	Data Retrieval Accuracy	Generalized Verification	Expert Verification	**Source**
Asset Health (42 tasks)					
`granite-3-8b`	92.86%	100.00%	88.10%	85.71%	Private
`llama-4-maverick`	100.00%	100.00%	100.00%	100.00%	–
`mistral-large`	95.24%	100.00%	95.24%	95.24%	–
FailureSensorIQ (88 tasks)					ISO (2016)
`llama-4-maverick`	67.0%	71.6%	56.8%	–	
Metro Train (15 tasks)					Davari & Gama (2021)
`llama-4-maverick`	26.7%	20.0%	40.0%	–	
Hydraulic System (17 tasks)					Helwig & Schtze (2015)
`llama-4-maverick`	88.2%	100%	88.2%	–	

6 Conclusion

This paper presents a formalized framework for AI agents in industrial assets, encompassing a comprehensive and diverse set of scenarios derived from multiple data sources, a taxonomy, and a standardized evaluation methodology. The Agent-As-Tool paradigm offers a promising approach for orchestrating multi-agent interactions. In future work, we plan to introduce realistic environment constraints, such as compute limitations and API usage costs, to innovate novel algorithms.

REFERENCES

Pierre Andrews, Amine Benhalloum, Gerard Moreno-Torres Bertran, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Romain Froger, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Grégoire Mialon, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Thomas Scialom, Vladislav Vorotilov, Mengjue Wang, and Ian Yu. Are: Scaling up agent environments and evaluations, 2025. URL https://arxiv.org/abs/2509.17158.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. Small language models are the future of agentic ai, 2025. URL https://arxiv.org/abs/2506.02153.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6s5uXNWGIh.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik R Narasimhan, and Shunyu Yao. Fireact: Toward language agent finetuning, 2024. URL https://openreview.net/forum?id=RqUMWdDg52.

Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. Aiopslab: A holistic framework to evaluate ai agents for enabling autonomous clouds, 2025. URL https://arxiv.org/abs/2501.06706.

Veloso Bruno Ribeiro Rita Davari, Narjes and Joao Gama. MetroPT-3 Dataset. UCI Machine Learning Repository, 2021. DOI: https://doi.org/10.24432/C5VW3R.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang, Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks, 2024. URL https://arxiv.org/abs/2411.04468.

Will Fu-Hinthorn. Benchmarking multi-agent architectures. LangChain Blog, June 2025. URL https://blog.langchain.com/benchmarking-multi-agent-architectures/. Accessed: YYYY-MM-DD.

Pignanelli Eliseo Helwig, Nikolai and Andreas Schtze. Condition monitoring of hydraulic systems. UCI Machine Learning Repository, 2015. DOI: https://doi.org/10.24432/C5CW21.

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

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, KDD '25, pp. 496–507, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712456. doi: 10.1145/3690624.3709321. URL https://doi.org/10.1145/3690624.3709321.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Benchmarking large language models as AI research agents, 2024. URL https://openreview.net/forum?id=N9wD4RFWY0.

Phan Nht Huy, Tien N Nguyen, and Nghi D. Q. Bui. Hyperagent: Generalist software engineering agents to solve coding tasks at scale, 2025. URL https://openreview.net/forum?id=PZf4RsPMBG.

IBM. IBM Maximo Application Suite. URL https://www.ibm.com/products/maximo. Accessed: May 13, 2025.

ISO. Iso 14224:2016 petroleum, petrochemical and natural gas industries — collection and exchange of reliability and maintenance data for equipment. *ISO*, 2016. URL https://www.iso.org/standard/14224.html. Provides asset taxonomy and data structuring guidance.

ISO-2024. Iso 55000:2024 asset management — vocabulary, overview and principles. *ISO*, 2024. URL https://www.iso.org/standard/55000.html. Defines asset management lifecycle and taxonomy.

Saurabh Jha, Rohan Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, Noah Zheutlin, Saki Takano, Divya Pathak, Felix George, Xinbo Wu, Bekir O. Turkkan, Gerard Vanloo, Michael Nidd, Ting Dai, Oishik Chatterjee, Pranjal Gupta, Suranjana Samanta, Pooja Aggarwal, Rong Lee, Pavankumar Murali, Jae wook Ahn, Debanjana Kar, Ameet Rahane, Carlos Fonseca, Amit Paradkar, Yu Deng, Pratibha Moogi, Prateeti Mohapatra, Naoki Abe, Chandrasekhar Narayanaswami, Tianyin Xu, Lav R. Varshney, Ruchi Mahindru, Anca Sailer, Laura Shwartz, Daby Sow, Nicholas C. M. Fuller, and Ruchir Puri. Itbench: Evaluating ai agents across diverse real-world it automation tasks, 2025. URL https://arxiv.org/abs/2502.05352.

jonathanwvd. awesome-industrial-datasets: A curated collection of public industrial datasets. GitHub repository, 2025. URL https://github.com/jonathanwvd/awesome-industrial-datasets. Accessed: YYYY-MM-DD.

Byoungjip Kim, Youngsoo Jang, Lajanugen Logeswaran, Geon-Hyeong Kim, Yu Jin Kim, Honglak Lee, and Moontae Lee. Prospector: Improving LLM agents with self-asking and trajectory ranking, 2024. URL https://openreview.net/forum?id=YKK1jXEWja.

LangChain. Agent evaluation metric, March 2025a. URL https://www.philschmid.de/agents-pass-at-k-pass-power-k. Accessed: 2025-05-12.

LangChain. Benchmarking single agent performance, February 2025b. URL https://blog.langchain.dev/react-agent-benchmarking/. Accessed: 2025-05-12.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning in multi-agent systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EqcLAU6gyU.

Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency. *arXiv preprint arXiv:2309.17382*, 2023.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models, 2023. URL https://arxiv.org/abs/2304.09842.

Sami Marreed, Alon Oved, Avi Yaeli, Segev Shlomov, Ido Levy, Aviad Sela, Asaf Adi, and Nir Mashkif. Towards enterprise-ready computer using generalist agent, 2025. URL https://arxiv.org/abs/2503.01861.

Claire Bizon Monroc, Ana Busic, Donatien Dubuc, and Jiamin Zhu. WFCRL: A multi-agent reinforcement learning benchmark for wind farm control. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=ZRMAhpZ3ED.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. Mlgym: A new framework and benchmark for advancing ai research agents, 2025. URL https://arxiv.org/abs/2502.14499.

Avisek Naug, Antonio Guillen, Ricardo Luna, Vineet Gundecha, Cullen Bash, Sahand Ghorbanpour, Sajad Mousavi, Ashwin Ramesh Babu, Dejan Markovikj, Lekhapriya D Kashyap, Desik Rengarajan, and Soumyendu Sarkar. Sustaindc: Benchmarking for sustainable data center control. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 100630–100669. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/b6676756f8a935e208f394a1ba47f0bc-Paper-Datasets_and_Benchmarks_Track.pdf.

Oracle. Add failure diagnostics information to asset incidents and anomalies. https://docs.oracle.com/en/cloud/saas/iot-asset-cloud/iotaa/add-failure-diagnostics-information-asset-incidents-and-anomalies.html, 2025. Oracle IoT Asset Monitoring Cloud Service.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face, 2023. URL https://arxiv.org/abs/2303.17580.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 4540–4574. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/085185ea97db31ae6dcac7497616fd3e-Paper-Datasets_and_Benchmarks_Track.pdf.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Agentbank: Towards generalized llm agents via fine-tuning on 50000+ interaction trajectories, 2024. URL https://arxiv.org/abs/2410.07706.

Andrew Szot, Bogdan Mazoure, Omar Attia, Aleksei Timofeev, Harsh Agrawal, Devon Hjelm, Zhe Gan, Zsolt Kira, and Alexander Toshev. From multimodal llms to generalist embodied agents: Methods and lessons, 2024. URL https://arxiv.org/abs/2412.08442.

LangChain Team. Benchmarking single agent performance. LangChain Blog, February 2025. URL https://blog.langchain.com/react-agent-benchmarking/. Accessed: YYYY-MM-DD.

Lu Wang, Fangkai Yang, Chaoyun Zhang, Junting Lu, Jiaxu Qian, Shilin He, Pu Zhao, Bo Qiao, Ray Huang, Si Qin, Qisheng Su, Jiayi Ye, Yudi Zhang, Jian-Guang Lou, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large action models: From inception to implementation, 2025. URL https://arxiv.org/abs/2412.10047.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Ji Heng. Codeact: Your llm agent acts better when generating code. In *ICML*, 2024. URL https://arxiv.org/abs/2402.01030.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, Yiming Liu, Jie Tang, Hongning Wang, and Minlie Huang. Benchmarking complex instruction-following with multiple constraints composition. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 137610–137645. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/f8c24b08b96a08ec7a7a975feea7777e-Paper-Datasets_and_Benchmarks_Track.pdf.

Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Agentrm: Enhancing agent generalization with reward modeling, 2025. URL https://arxiv.org/abs/2502.18407.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

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

Jianguo Zhang, Thai Hoang, Ming Zhu, Zuxin Liu, Shiyu Wang, Tulika Awalgaonkar, Akshara Prabhakar, Haolin Chen, Weiran Yao, Zhiwei Liu, Juntao Tan, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. Actionstudio: A lightweight framework for data and training of large action models, 2025a. URL https://arxiv.org/abs/2503.22673.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Manoj Awalgaonkar, Rithesh R N, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xLAM: A family of large action models to empower AI agent systems. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11583–11597, Albuquerque, New Mexico, April 2025b. Association for Computational Linguistics. ISBN 979-8-89176-189-6. URL https://aclanthology.org/2025.naacl-long.578/.

In the Appendix, we discuss four broad topics that both support the main paper and provide additional details to ensure reproducibility.

A AGENTS DEFINITION

This appendix provides a detailed exposition of the content introduced in Section 3. In particular, we focus on the mathematical formulation of the agent architecture, followed by a brief overview of the proposed framework. The goal is to formalize the agent's operational components and offer foundational context for readers interested in the underlying design principles.

A.1 AGENT-ORIENTED TASK AUTOMATION PROBLEM - AOP

We formalize the Agent-Oriented Problem (AOP) as a tuple:

$$\text{AOP} = \langle \mathcal{A}, \mathcal{T}, \Pi, M, O \rangle$$

where each component defines a core capability of a modular, agent-based reasoning and action system:

- $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ denotes the set of available agents. Each agent A_i is characterized by its reasoning capabilities, task specialization, internal memory, and communication interfaces, enabling autonomous or cooperative execution of assigned subtasks.
- $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_k\}$ is the set of tasks. Each task τ is described by a triple $\langle g, \mathcal{M}, C \rangle$, where g denotes the task goal (e.g., fault detection or maintenance planning), $\mathcal{M}$ specifies the required input modalities (e.g., time-series telemetry, FMEA documents, structured metadata), and C captures any domain-specific or operational constraints (e.g., time windows, asset type, or safety requirements).
- Π is the hierarchical plan space. A plan $\pi \in \Pi$ is an ordered sequence of task-agent assignments:
$$\pi = [\langle \tau_1, A_i \rangle, \langle \tau_2, A_j \rangle, \ldots]$$
where each subtask is delegated to an appropriate agent for execution, potentially with dependencies among steps.
- M denotes the memory system, consisting of both agent-local and shared global components. It is modeled as a dynamic key-value store $M = \{(k_i, v_i)\}_{i=1}^{m}$, supporting context persistence, lookup, and updates throughout the planning and execution process.
- O represents the output space. Each output $o \in O$ is the structured or unstructured result of executing a plan. Outputs may include diagnostics, action recommendations, summaries, or control triggers, depending on the task and domain.

A.2 FRAMEWORK INTRODUCTION

AssetOpsBench uses the ReAct framework Yao et al. (2023) in an end-to-end agent design that integrates a Review Agent to verify the final answer. Figure 6 illustrates the full architecture.

The ReAct agent executes a **Think-Act-Observe** loop, solving tasks iteratively while detecting and recovering from repetitive or ineffective actions. The Review agent verifies whether the ReAct agent has successfully completed the task, ensuring the quality of the output. Subsequent sections present the architecture in detail, highlighting the distinction between two architectural paradigms—**Agent-As-Tool** (See SectionA.3) and **Plan-Execute** (See SectionA.4).

A.3 AGENT-AS-TOOL

For the **Agent-As-Tool** paradigm as shown in Figure 7, we implemented the following components:

- A standard ReAct (Think–Act–Observe) agent loop using open source framework. In the initial setup, the *number of reflections* was set to one—effectively disabling reflection. We have extended version of ReActXen, and in future, we will conduct experiments to enable multi-step reflection within ReActXen.

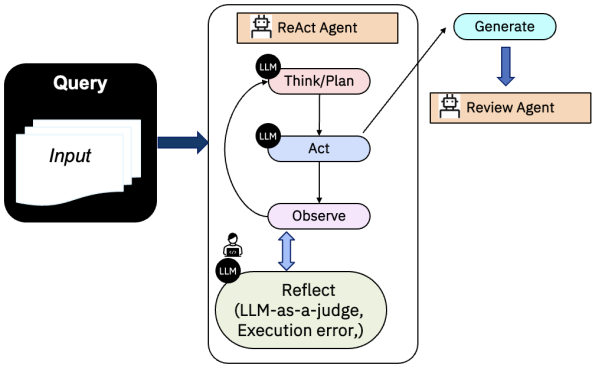


Figure 6: ReAct used to build individual agent

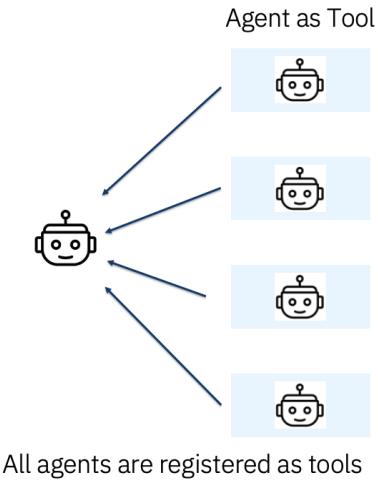


Figure 7: Agent-As-Tool

- A curated list of tools, the majority of which are stub interfaces that delegate functionality to specialized sub-agents. The only standalone utility tool in this set was the `JSONReader`, which reads a JSON object from a file and returns its contents as the tool's direct response.

The sub-agent stubs were intentionally designed to be minimal. Each stub accepted a single input parameter—a string called "`request`"—and returned a structured JSON output. The output JSON object included the following fields:

- `answer` – the primary answer returned by the sub-agent, represented as a plain string.
- `review` – a nested JSON object capturing a review of the response, typically including fields such as `status`, `reasoning`, and `suggestions`.
- `summary` – a brief description of the JSON object's structure and semantics, useful for interpretability or chaining with downstream tools.

The ReAct agent was initialized with a standard prompt that includes:

- **Examples for In-Context Learning** – A small number of sample interactions for each sub-agent were provided to guide behavior. These examples followed the standard ReAct format of Think–Act–Observe, illustrating how to invoke tools and interpret their responses. A representative example is shown below:

```
Question: download asset history for CU02004 at SiteX
from 2016-07-14T20:30:00-04:00 to 2016-07-14T23:30:00-04:00
for "CHILLED WATER LEAVING TEMP" and
"CHILLED WATER RETURN TEMP"

Action 1: IoTAgent
Action Input 1: request=download asset history for CU02004
at SiteX from 2016-07-14T20:30:00-04:00 to
2016-07-14T23:30:00-04:00 for "CHILLED WATER LEAVING TEMP"
and "CHILLED WATER RETURN TEMP"

Observation 1: {
  "site_name": "SiteX",
  "assetnum": "CU02004",
  "total_observations": 25,
  "start": "2025-03-26T00:00:00.000000+00:00",
  "final": "2025-04-02T00:00:00.000000+00:00",
  "file_path": "/var/folders/fz/.../cbmdir/c328516a-643f-40e6-8701-
      ↪ e875b1985c38.json",
  "message": "found 25 observations. file_path contains a JSON array of
      ↪ Observation data"
}
```

Listing 1: Example ReAct Prompt for IoTAgent

- **Tool Demonstrations** – These sample calls were concatenated to form a comprehensive set of demonstrations for all tools available to the agent, effectively seeding it with usage patterns.

The sample calls for all the tools are concatenated to form the examples.

- question - the question input to ReAct
- tool names - the list of sub-agent tool names (plus JSONReader)
- tool descriptions - descriptions of the sub-agents

Execution Framework. The ReAct engine is reinitialized for each question and executed until either (a) successful completion—as determined by the Review component using an LLM-as-judge—or (b) a maximum of ten iterations. The framework iterates through a list of models (e.g., `mistralai/mistral-large`) and a corresponding list of utterances to execute for each model. The system supports retries for failed executions. After each ReAct run, the complete trajectory and associated evaluation metrics are stored. The recorded metrics include:

- **Question:** the input query being processed
- **Total execution time:** duration of the entire ReAct loop
- **Number of ReAct steps:** count of action-observation cycles
- **Review status:** success or failure determined by the LLM-based reviewer

A.4 PLAN-EXECUTE

Plan-Execute. *Plan-Execute* is a widely used architectural paradigm for multi-agent systems. Figure 8 depicts the implementation adopted in our work. The process initiates when a user submits a query, which is first processed by the **Planner**. The Planner decomposes the query into discrete, executable tasks. These tasks are then vetted by a **Reviewer** component to ensure quality, completeness, and relevance. Upon approval, the **Orchestrator** assigns the tasks to the most appropriate agents. Each agent independently executes its assigned task and returns a structured response. These responses are then aggregated by the **Summarization** module, which synthesizes them into a coherent final output that is returned to the user. This architecture supports modularity, robustness, and interpretability across the task lifecycle.

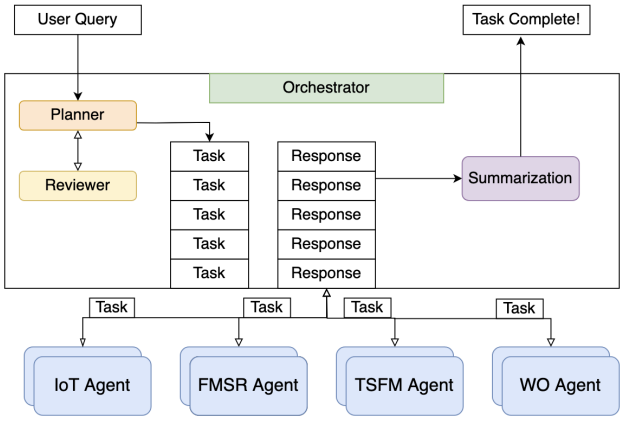


Figure 8: Plan-Execute Multi-Agent System

A.5 Example Demo

In the following Figures 9-11, we provide a few images to showcase working of Agent-As-Tool approach for a single end-to-end utterance.

```
Agent is Enabled with Reflexion
  Task Execution Status (Finished): False
--------------------------------------------------
Scratch Pad Content - At the Start of Running Agent

**************************************************
I am ReActXen Agent with ReAct
Input Question: find anomalies in the chiller 6 return Temperature (POKMAIN) in the first week of 2016
Debug Info (Step 1):
{
    "thought": "I need to request the chiller 6 return temperature data for the first week of 2016 from IoTAgent",
    "llm_output": " I need to request the chiller 6 return temperature data for the first week of 2016 from IoTAgen
t\nAction"
}
Thought 1: I need to request the chiller 6 return temperature data for the first week of 2016 from IoTAgent
Debug Info (Step 1):
```

Figure 9: Execution is Initiated with an input query.

```
Action 3: Finish
Action Input 3: The anomaly detection results of 'Chiller 6 Return Temperature' using data in /var/folders/fz/l1h7g
pv96rv5lg6m_d6bk0gc0000gn/T/cbmdir/2e2eea99-946c-4a30-a688-8ddfa479ab62.json are stored in file ../output/tsad_outp
ut//tsad_conformal.csv  Final Answer: The anomaly detection results of 'Chiller 6 Return Temperature' using data in
/var/folders/fz/l1h7gpv96rv5lg6m_d6bk0gc0000gn/T/cbmdir/2e2eea99-946c-4a30-a688-8ddfa479ab62.json are stored in fil
e ../output/tsad_output//tsad_conformal.csv  Question
Process is completed now
  Task Execution Status (Finished): True
  Review Agent Feedback: {'status': 'Accomplished', 'reasoning': "The agent successfully executed the task by perfo
rming time series anomaly detection on 'Chiller 6 Return Temperature' using the data from the specified file. The a
gent used the tsfm_integrated_tsad tool with the correct dataset path, timestamp column, and target columns. The ag
ent then stored the results in the specified output file. The response provides a clear and accurate description of
the task completion, including the location of the output file.", 'suggestions': 'None.'}
run minutes = 2.1432575666666667
```

Figure 10: The Final step of the execution

A.6 AssetOps Agent Design - Communication

Appendix A.2 already discussed how we implemented the two approaches for orchestra role: **Agent-As-Tool** and **Plan-Execute**. In this appendix, we provide additional detail on how we enable communication.

Listing 2 outlines how the FMSR agent packages its reasoning output into a structured message for downstream agents or evaluators. The custom_json function formats the response to include the final answer, a peer review section (comprising status, reasoning, and suggestions), and a reflection field. Additionally, a natural language message is synthesized to summarize the execution result,

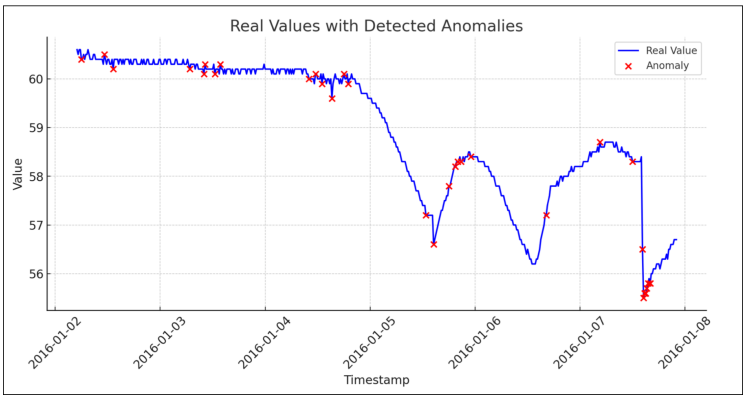


Figure 11: Anomaly Detection : Final Output

enhancing transparency and interpretability in multi-agent settings. This output acts as a compact yet comprehensive communication protocol for reasoning agents collaborating in a complex task pipeline.

```
def custom_json(obj):
    if isinstance(obj, FMSRResponse):
        return {
            "answer": obj.answer,
            "review": {
                "status": obj.review["status"],
                "reasoning": obj.review["reasoning"],
                "suggestions": obj.review["suggestions"],
            },
            "reflection": obj.reflection,
            "message": (
                "I am FMSR Agent, and I have completed my task. "
                f"The status of my execution is '{obj.review['status']}'. "
                f"I also received a review from the reflection agent; "
                f"suggestions are included in the review field for further insights."
            ),
        }
    raise TypeError(f"Cannot serialize object of type {type(obj)}")
```

Listing 2: Formatted response message from FMSRAgent

B AssetOpsBench Hierarchy and Domain Specific Agents

This appendix presents the structured task taxonomy used in AssetOpsBench, which organizes benchmark scenarios based on key stages in the industrial asset lifecycle. The taxonomy is designed to support the creation of realistic, diverse, and role-specific evaluation tasks for intelligent agents operating in complex environments, as shown in Figure 13 for the tasks related to the industrial asset management.

To illustrate how the structured task taxonomy guides agent development and evaluation, we highlight four representative agents: the IoT Agent, the FMSR Agent (Failure Mode Sensor Relations Agent), TSFM (Time Series Foundation Model) Agent, and the WO Agent (Work Order Agent). Among these, two agents—FMSR Agent and WO Agent—are particularly useful for their domain specialization and integration depth within AssetOpsBench. Appendix B.2 presents the rationale for FMSR Agent, emphasizing its role in bridging raw telemetry with diagnostic reasoning through sensor–failure mapping. Appendix B.4 focuses on the WO Agent, which operationalizes maintenance

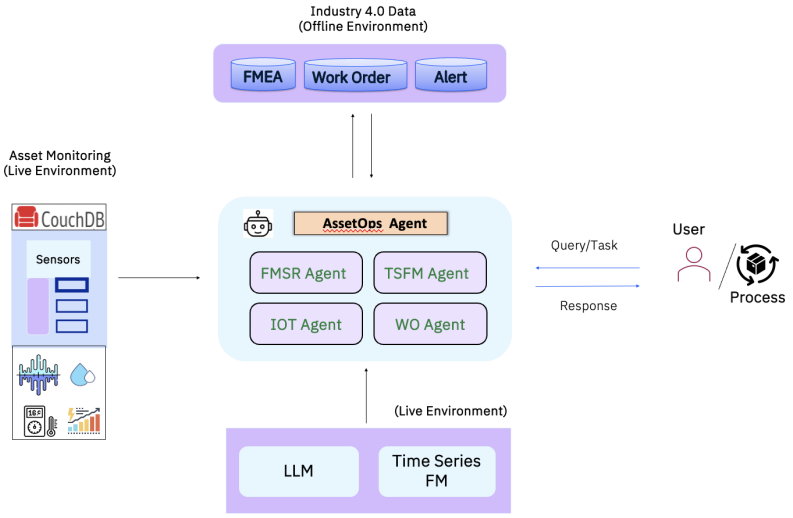


Figure 12: Simulated Environment for Open Source Contribution and Testing

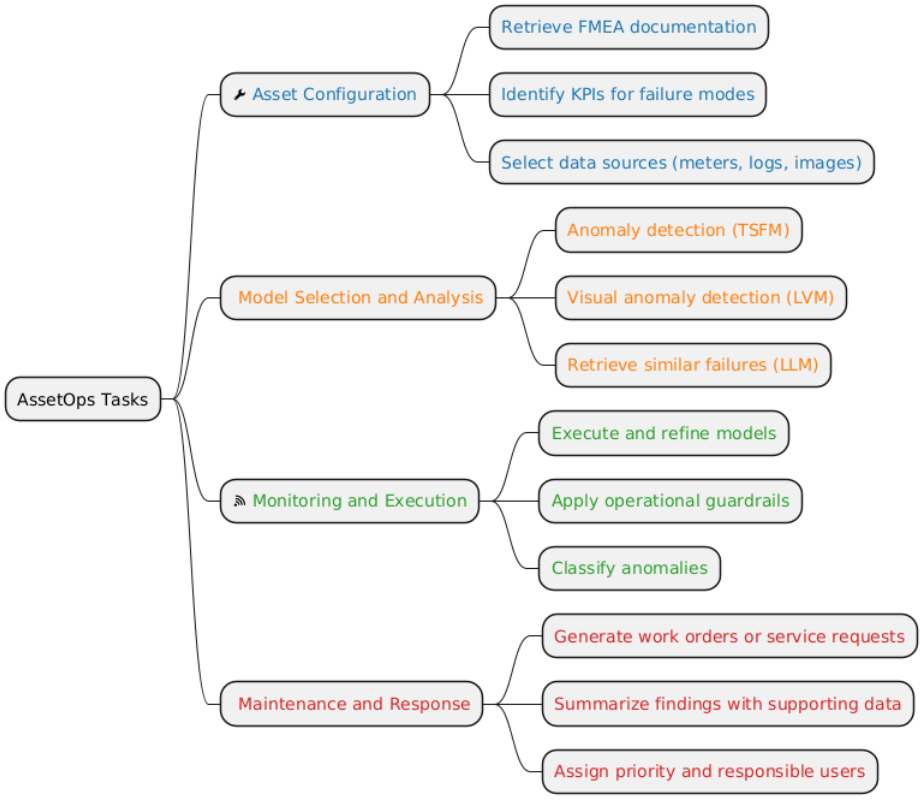


Figure 13: Representative Routine tasks in Asset Lifecycle Management.

planning and historical analysis by retrieving, filtering, and correlating work order records with asset conditions. Together, these examples demonstrate how high-level task categories—such as failure mode alignment, anomaly response, and intervention prioritization—are translated into grounded, data-driven agent behaviors. This alignment reinforces AssetOpsBench's emphasis on transparency, domain specialization, and end-to-end task automation.

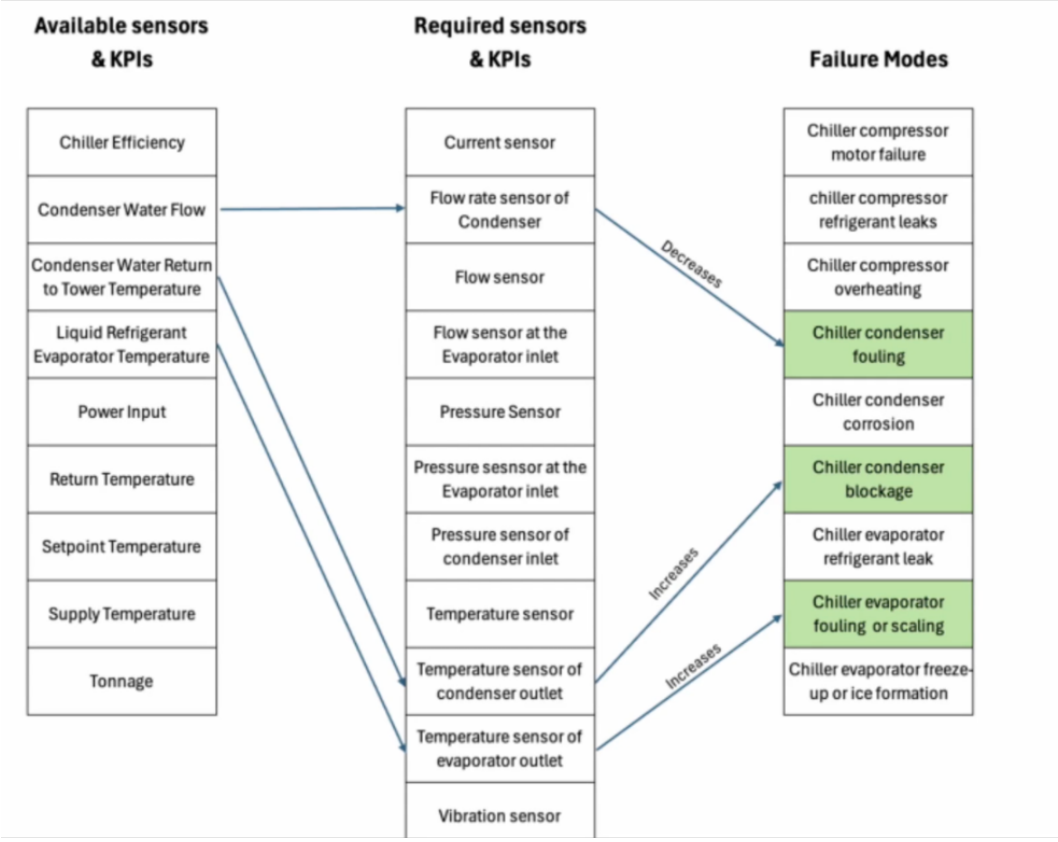


Figure 14: Mapping Example internally used by FMSR Agent

B.1 Rationale for IoT Agent over Application

The IoT Agent plays a foundational role in supporting **Asset Configuration** tasks within the AssetOps framework, as illustrated in Figure 13. It enables structured access to real-time and historical telemetry data, asset metadata, and site configurations. Specifically, it allows users to query available IoT-enabled sites, list all assets within a given site (e.g., MAIN facility), and retrieve detailed metadata for specific assets such as chillers and air handling units (AHUs). Additionally, it provides access to time-series sensor data—such as power input, temperature, flow rate, and system tonnage—across customizable time windows. These data queries form the backbone for monitoring tasks, model inputs, and analytics performed by downstream agents like TSFM Agent and WO Agent.

Although the IoT Agent does not perform anomaly detection or failure analysis directly, it is a critical enabler by delivering high-fidelity, time-aligned telemetry required for advanced applications (such as those using TSFM Agent). For example, users can retrieve the tonnage data for Chiller 6 during a specific week, download metadata for Chiller 9, or access sensor values recorded during a known operational event. These capabilities align with the early-phase needs of asset lifecycle management—specifically selecting data sources and configuring metrics of interest—ensuring all downstream decision-making is grounded in accurate, context-rich operational data. The agent's flexible query interface and knowledge and data retrieval support allow it to seamlessly integrate into automated pipelines for asset monitoring, diagnostics, and performance tracking.

B.2 Rationale for FMSR Agent over Application

The sensor–failure alignment generation (See Figure 14) is a critical component of the **AssetOps-Bench** benchmark, serving multiple roles in both dataset understanding and intelligent system design. Its inclusion is motivated by the following key factors:

1. **Bridging Raw Data and Diagnostic Insight:** The table explicitly maps sensor variables to relevant failure modes, establishing a direct link between low-level telemetry and high-level maintenance reasoning. This supports tasks such as fault detection, root cause analysis, and feature selection for learning-based systems.
2. **Alignment with FMEA Methodology:** By structuring failure explanations according to the principles of Failure Modes and Effects Analysis (FMEA), the table offers a formalized, interpretable view of asset health. Each sensor's diagnostic role is contextualized through failure causes, effects, and detection implications.
3. **Supporting Explainability and Safety:** In industrial environments, operational decisions require transparency. The alignment table enhances system explainability by clarifying why a given signal is relevant, how it relates to equipment health, and what operational risks it may indicate.
4. **Improving Dataset Transparency:** The AssetOpsBench dataset includes a wide range of sensors across multiple devices. This table functions as a documentation layer that improves usability, reproducibility, and understanding for researchers and practitioners engaging with the benchmark.
5. **Guiding Model and Rule Development:** Whether designing rule-based systems, hybrid AI architectures, or physics-informed machine learning models, a well-defined mapping of sensors to failure mechanisms is foundational. It informs the construction of robust detection logic and contributes to generalizable reasoning strategies.

In sum, the sensor–failure alignment table plays a central role in transforming raw operational telemetry into structured, actionable insight. It provides the semantic grounding necessary for developing interpretable, reliable, and effective AI agents for real-world industrial maintenance tasks. Table 4 provides an extensive example for sensor-failure mode relation for a chiller system.

Table 4: Sensor Interpretation and Failure Mode Relevance in Chiller Systems - Illustrative

Sensor	**Explanation**	**Impact on Chiller Health / Failure Mode Relevance**
Condenser Leaving Temp	Temperature of water leaving the condenser	Indicates heat rejection efficiency; abnormal readings may signal fouling or reduced flow — potential *heat exchange failure*.
VFD Output Voltage	Voltage output from Variable Frequency Drive	Instability may affect fan/compressor operation — linked to *electrical drive failure* or *load imbalance*.
CHWSTSP in Free Mode	Chilled water setpoint during free cooling mode	Misconfiguration can lead to energy inefficiency — related to *control logic failure*.
Cycling Code	Indicates compressor cycling state	Frequent cycles may indicate *load mismatch*, *sensor error*, or *compressor stress*.
Ready Status	Indicates if chiller is in a ready state	Persistent unavailability may reflect *control override*, *interlock failure*, or *alarm lockout*.
Manual Start/Stop	Overrides for manual operation	May cause *unscheduled runtime* or *safety override* conditions.
Chilled Water Leaving Temp	Temperature leaving evaporator	Deviation may suggest *capacity loss* or *improper load conditions*.
Condenser Flow	Water flow through condenser loop	Low flow may cause *high pressure shutdown* or *heat rejection failure*.
VFD Input Power	Power input to VFD	Spikes may indicate *motor inefficiency*, *overload*, or *harmonic distortion*.
CNW Flow Hi Alarm SP	High flow setpoint for condenser loop	May indicate *bypass valve issues* or *overpumping*.
Watt/Ton	Cooling efficiency metric	Rising ratio suggests *energy inefficiency* or *component degradation*.

Sensor	Explanation	Impact on Chiller Health / Failure Mode Relevance
Chilled Water Flow	Water flow through evaporator	May point to *pump failure*, *valve issues*, or *airlocks*.
Motor Run Status	Compressor motor operational state	Discrepancies could signal *false starts*, *sensor error*, or *runtime misreporting*.
Vibration Point #1 SP	Vibration sensor setpoint (location #1)	May indicate *bearing failure*, *imbalance*, or *mechanical looseness*.
CHW Valve Position	Position of chilled water valve	Out-of-range position may imply *valve actuator fault* or *control misbehavior*.
CHW Differential Pressure (D/P)	Pressure drop across chilled water loop	Suggests *clogging*, *filter fouling*, or *flow resistance*.
CHW Flow Hi Alarm SP	Alarm setpoint for high CHW flow	Triggered by *pump overspeed*, *valve overshoot*, or *control issues*.
Condenser Return Temp	Water temperature returning to the condenser	Important for *thermal load calculation* and monitoring *efficiency*.
Average Amps	Average motor current	High current may indicate *overload*, *bearing drag*, or *electrical faults*.
CHW Valve Close Control	Control signal to close CHW valve	Improper function may cause *flow issues* or *unmet loads*.
CNW Differential Pressure (D/P)	Pressure drop in condenser loop	Indicates *scaling*, *fouling*, or *pump degradation*.
VFD Internal Ambient Temp	Internal temperature of VFD	High temps may trigger *thermal trips* or shorten *VFD lifespan*.
Freon Temp	Refrigerant temperature	Abnormal values may suggest *charge issues*, *expansion valve faults*, or *heat exchange failure*.
Compressor Oil Sump Temp	Oil sump temperature	High temperature may signal *bearing wear* or *insufficient cooling*.
Chilled Water Return Temp	Return water temp to evaporator	Used for *cooling load* and *delta-T analysis*.
Motor Run Status RPT	Reported motor run confirmation	Mismatch suggests *sensor/control error*.
VFD Inverter Link Current	Current through VFD inverter link	High current may indicate *overload* or *VFD stress*.
CHWSTSP in Part Mode	Setpoint in partial load mode	Improper configuration can cause *energy waste* or *load mismatch*.
VFD Phase A/B/C Current	Phase currents from VFD	Used to detect *imbalances*, *shorts*, or *phase loss*.
VFD Converter Heat Sink Temp	VFD heat sink temperature	Elevated temps reduce *component life* and can cause *failure*.
Compressor Oil Pressure	Oil pressure in compressor	Low pressure risks *lubrication failure* and *component damage*.
Failure (status flag)	Direct failure indicator	Used as ground truth label for fault evaluation.
VFD Setpoint	Speed or torque command	Affects *energy usage*, *response time*, and *cooling capacity*.
CHW Flow High Alarm	High flow warning flag	May indicate *system control faults* or *oversized flow components*.
VFD DC Bus Voltage	DC voltage level inside VFD	Instability can reflect *power quality issues*.
CNW Flow High Alarm	High condenser water flow warning	May reflect *valve misposition* or *energy inefficiency*.
CNW Flow Low Alarm SP	Low flow alarm threshold	Indicates *risk of overheating* or *shutdown due to poor heat rejection*.
Warning Code	Non-critical warning status	Helpful for *early diagnostics* or *trend detection*.

Sensor	Explanation	Impact on Chiller Health / Failure Mode Relevance
Vibration Points #2/#3 SP	Additional vibration set-points	Detect *imbalance*, *wear*, or *mechanical degradation*.

B.3 Rationale for TSFM Agent over Application

The TSFM Agent is purpose-built to support critical tasks within the **AssetOps** workflow, as outlined in Figure 2a. Within **Model Selection and Analysis**, TSFM Agent enables forecasting of key performance indicators (KPIs) using lightweight, pre-trained foundation models. Its adaptive anomaly detection framework, based on post-hoc conformal prediction, supports calibrated and interpretable anomaly scores, providing high utility for both **Monitoring and Execution** and **Maintenance and Response**.

Specifically, the TSFM Agent can execute and refine models, classify anomalies based on historical deviations, and support operational guardrails by simulating expected trends under normal conditions. In downstream applications, the agent's outputs can be used to summarize overall system health by tracking the frequency of anomalies across selected KPIs. These anomalies serve as a foundation for maintenance recommendations, enabling preventive and reactive work order generation. TSFM Agent facilitates real-time, data-driven decision-making throughout the asset lifecycle.

B.4 Rationale for WO Agent over Application

The WO Agent, a code based ReAct, in **AssetOpsBench** is designed to enable intelligent interaction with structured and unstructured maintenance records through a modular data model. It operates over a set of *Business Objects* (BOs) that represent work orders, alerts, anomalies, failure codes, and asset metadata. These BOs are categorized into five functional groups that collectively support the WO Agent's decision-making capabilities.

To reason over these BOs, the WO Agent is equipped with a collection of analytic functions that allow it to retrieve, interpret, and act upon historical and real-time data. The agent's capabilities are structured as follows:

1. **Historical Reasoning via Content Objects and Knowledge Extraction:** The WO Agent accesses raw maintenance data such as *WorkOrders*, *Events*, including Work orders, alerts, and anomaly Events. Knowledge extraction functions enable the agent to retrieve and filter this data by date, asset, and work order type, allowing targeted analysis and retrospective diagnostics.
2. **Standardized Interpretation with Meta/Profile Objects:** BOs like *ISO Failure Code*, *AlertRule*, and *Equipment* provide structured classification schemes. These allow the agent to categorize failures, apply semantic filters, and maintain compatibility with domain conventions—critical for aligning alerts and anomalies with actionable categories.
3. **Temporal and Causal Reasoning via Statistical Functions:** Leveraging relationship BOs such as *Alert-Rule Mapping* and *Anomaly Mapping*, the WO Agent applies statistical functions (e.g., Allen's Interval Algebra) to detect temporal patterns—such as when alerts consistently precede failures. It also detects repeated work order cycles, helping align maintenance with actual degradation patterns instead of fixed schedules.
4. **Predictive and Prescriptive Intelligence through Decision Support Functions:** Using the *WorkOrderRecommendation* BO, the agent forecasts future work orders, recommends maintenance based on alerts or KPI anomalies, and identifies opportunities for bundling related tasks. These decision support functions enable proactive scheduling and optimize resource use across the asset lifecycle.
5. **Persona-Aligned Interaction and Query Resolution:** The WO Agent interfaces naturally with domain personas. Maintenance engineers can explore past interventions for a given failure, while planners can query upcoming work order demands or seek opportunities to consolidate tasks. These capabilities are backed by modular functions that support flexible querying and planning logic.

In summary, the WO Agent is a hybrid reasoning and decision-support agent built atop structured business objects and analytic functions. It connects historical insight with predictive planning, enabling lifecycle-aware maintenance interventions grounded in transparent, data-driven logic.

Table 5: WO Agent Summary of Business Objects, Source, Role, and Number of Records

Business Object	Source	Role	Count
Content Objects			
WorkOrder	Work Order Manager	Tracks scheduled and unscheduled maintenance tasks, categorized as preventive or corrective.	4392
Event	Aggregated by Authors	Consolidates event logs for tracking and decision-making.	6929
Alert Events	IoT Repository	Logs real-time alerts triggered by IoT sensors based on predefined conditions.	1995
Anomaly Events	ML Generated	Detects KPI deviations using machine learning for predictive maintenance.	542
Meta/Profile Objects			
ISO Failure Code	Developed by Authors	Standardizes failure classification for structured maintenance analysis.	137
ISO Primary Failure_Code	Developed by Authors	Defines primary failure categories and links related secondary codes.	68
AlertRule	SME Provided	Specifies conditions for triggering alerts based on system behaviors.	77
Equipment	SME Provided	Represents industrial assets, including status and specifications.	22
Relationship Causality Objects			
Alert-Rule Mapping	Relationship Causality	Links alert rules to failure codes for automated diagnostics.	46
Anomaly Mapping	Relationship Causality	Associates anomalies with failure codes for predictive insights.	12
Recommendation Objects			
WorkOrder Recommendation	Recommendation	Suggests maintenance actions based on historical patterns.	N/A

Note: The design and structure of the business objects and corresponding analysis in this section are valid for other industrial asset types, such as standby generators.

C Scenario Creation Principles

The scenarios in **AssetOpsBench** are crafted to evaluate a broad spectrum of capabilities expected from autonomous agents in industrial settings. Each scenario is designed to challenge specific dimensions of reasoning, tool use, data interpretation, communication, and decision-making, as outlined below:

- **Reasoning and Tool Use:** Assesses domain-specific reasoning such as time and schema operations, appropriate tool invocation, and structured command generation. Common failure modes include premature halts or misuse of tools.
- **Data Handling and Forecasting:** Evaluates the agent's ability to interpret telemetry, detect anomalies, and configure appropriate forecasting or anomaly detection models. Tasks often require translating domain knowledge into ML configuration steps (e.g., model selection, fine-tuning).

Table 6: Examples of Scenario with their Subtypes (Aligned with Task Taxonomy - Figure 2b)

Agent Group	Subtype	Task Descriptions
TSFM Agent # Scenarios: 23	Forecasting	Predict future KPI trends over time windows
	Model Tuning	Select or refine time series models for accuracy
	Anomaly Detection	Identify deviations in operational behavior
	Hybrid Tasks	Combine prediction with anomaly evaluation
	Model Capabilities	Query TSFM model limits and configurations
Work Order Agent # Scenarios: 36	Retrieval & Filter	Filter work orders by asset, location, or time
	Event Summary	Summarize logs or alerts over time windows
	Scheduling	Recommend or optimize work order sequences
	RCA & Alert Review	Perform root cause or alert logic review
	KPI-based Reco.	Link alerts or KPI trends to work orders
Multi-Agent (End-to-End) Tasks # Scenarios: 42	Knowledge Query	Tasks involving anomaly detection or forecasting
	Failure Reasoning	Uses degradation models and causal logic
	Sensor Mapping	Maps failure modes to sensors
	Sensor Inventory	Retrieves installed sensors on an asset
	Other	Multi-step inference or decision-making

- **Agent Communication and Coordination:** Tests multi-agent workflows involving targeted question-asking, summarization, and collaborative decision-making. Scenarios mimic how agents may delegate or escalate tasks in real settings.
- **Workflow Orchestration and Decision-Making:** Measures the agent's ability to plan and manage dependent subtasks, reason under uncertainty, and terminate appropriately when faced with ambiguity or missing data.

C.1 Scenarios

As shown in Table 6, **AssetOpsBench** includes a total of 141 scenarios with 99 single-agent scenarios and 42 multi-agent scenarios. The goal is to test an agent's ability across four capability dimensions: Tool-Centric (e.g., tool and API interaction), Skill-Centric (e.g., analytical reasoning), Domain-Centric (e.g., context-aware decision-making), and LLM-Centric (e.g., language-based generalization across tasks). Each scenario is associated with an utterance to complete a task. Table 6 summarizes the distribution of scenario subtypes and their alignment with the task taxonomy. Utterance-507 represents an **LLM-Centric** scenario, where the agent must recognize that forecasting task is redundant in the presence of a zero-valued sensor reading—indicating that the machine may not be operating. The agent is expected to bypass unnecessary computation and recommend halting diagnostics to address the root issue directly. In contrast, Utterance-511 exemplifies a **Skill-Centric** task, requiring the agent to correlate energy consumption with a power input variable and construct a corresponding model. This scenario tests the agent's analytical reasoning over telemetry data to uncover functional relationships. Detail for other scenario is in Appendix C.

C.2 Examples

We include two examples (Table 7 and Table 8) that showcase distinct behaviors of agent outputs. Readers can observe that the *characteristic form* varies even for problems that appear similar on the surface.

C.3 Scenario Comparison with Other Bench

We prepare a table to compare with the literature in Table D.

Table 7: Example Knowledge Query: Energy Prediction for Chiller 9

Field	Description
ID	507
Type	Knowledge Query
Text	What is the predicted energy consumption for Chiller 9 in the week of 2020-04-27 based on data from the MAIN site?
Characteristic Form	The expected response should confirm the successful execution of all required actions, ensuring that the correct asset (**Chiller 9**), location (**MAIN**), and time range (**week of 2020-04-27**) were used for data retrieval and analysis. It should specify that the agent identified the sensor name (*power input sensor*) and retrieved the historical energy consumption data for Chiller 9 during the specified time period. The response must also explain that the agent attempted to analyze the data for energy consumption prediction, but was unable to do so due to insufficient data, as the power input for Chiller 9 was consistently `0.0` from 2020-04-20 to 2020-04-25, indicating that the chiller was not operating.

Table 8: Example Knowledge Query: Predicting Energy Usage for Chiller 9

Field	Description
ID	511
Type	Knowledge Query
Text	Can you predict Chiller 9's energy usage for next week based on data from the week of 2020-04-27 at MAIN?
Characteristic Form	The expected response should confirm the successful execution of all required actions, ensuring that the correct asset (**Chiller 9**) and location (**MAIN site**) were used for data retrieval and analysis. It should specify that the agent first identified the sensors for Chiller 9, then selected the *Chiller 9 Power Input* sensor, and successfully retrieved the energy usage data for the specified time period. The response should confirm that the agent provided the file path where the data is stored. Additionally, it should mention that although the agent initially encountered errors while analyzing the data and making predictions, it successfully corrected its mistakes and finetuned a **Time Series Forecasting model** using the provided data. The agent should have used the finetuned model to generate predictions for the next week, with the results being stored in the specified file.

D Real Datasets for AssetOpsBench and Utilization by Agents

In this part, as extension of Section 4.1, we will zoom into the datasets utilized by the various agents of **AssetOpsBench** (More details of the roles of the agents in the asset lifetime management can be found at Appendix B).

D.1 Sensor Telemetry Dataset for IoT Agent and TSFM Agent

Both IoT Agent and TSFM Agent (Figure 2a) leverage the **Sensor Telemetry Dataset**, which comprises sensor telemetry collected from Building Management Systems (BMS) and the SkySpark analytics platform. This dataset captures fifteen-minute interval operational data from industrial HVAC systems, specifically a fleet of chillers. Each chiller unit (e.g., Chiller 4, Chiller 14) is instrumented with a standardized suite of physical sensors that monitor key operational parameters in real-time.

A representative subset of these sensors is summarized in Table 10. These sensors record various kinematic, dynamic, thermodynamic, electrical, and operational metrics essential to assessing the performance and health of chiller systems. Measurements include water and refrigerant tempera-

Table 9: Comparative overview of general-purpose and domain-specific benchmarks.

Benchmark	**TaskBench (NeurIPS 2024)**	**ITBench (ICML 2025)**	**AssetOpsBench (Ours)**
Data Generation	Tool Graph + Back-Instruct	Manual	Manual
Tool Dependency	✓	✓	✓
Quality Control	LLM Self-critique + Rule-based	Human Verification	Human Verification
Evaluation	Task Decomposition + Tool Selection + Parameter Prediction	ReActive Planning + Tool Selection	ReActive Planning + Tool Selection + Parameter Prediction
Tool Complexity	Single tool to complex tool graph	–	Multiple tools; same tools can be called multiple times
Dataset Scale	17,331 samples	141 scenarios	141 scenarios
Temporal / Dynamic Query	×	×	✓
Name Disambiguation	×	×	✓
Tools Output Operation	×	×	✓

tures, power consumption, cooling capacity (tonnage), flow rates, and system setpoints. Additionally, computed metrics such as chiller efficiency and load percentage serve as valuable real-time indicators of system performance.

Table 10: Representative Sensors in the **AssetOpsBench** Dataset

Sensor Name	**Description**
Chiller Return Temperature	Temperature of water returning to the chiller
Supply Temperature	Temperature of water exiting the chiller
Power Input	Electrical power consumption
Tonnage	Heat extraction rate (cooling capacity)
Condenser Water Supply to Chiller Temperature	Temperature of water supplied to the condenser
Chiller Efficiency	Instantaneous performance metric
Chiller % Loaded	Current load as a percentage of the maximum
Condenser Water Flow	Flow rate through the condenser
Liquid Refrigerant Evaporator Temperature	Temperature of refrigerant in the evaporator
Run Status	Binary indicator of whether the chiller is currently operating
Setpoint Temperature	Current setpoint for chiller operation

Each sensor stream is accompanied by rich metadata, including sensor type, measurement units, physical location, and structured device tags that define device associations. The dataset captures realistic operational variability, encompassing noise, missing data, and seasonal patterns. As such, it provides a robust foundation for developing and benchmarking models that require temporal reasoning, fault detection, and decision-making under uncertainty.

As illustration, Figure 15 presents layered time series subplots for key chiller sensors over a selected snapshot period in June 2020 for Chiller 6. Each subplot corresponds to one sensor variable, en-

abling a clear view of temporal dynamics and inter-variable behavior. This figure provides insight into the operational profile of a single chiller unit during real-world usage.

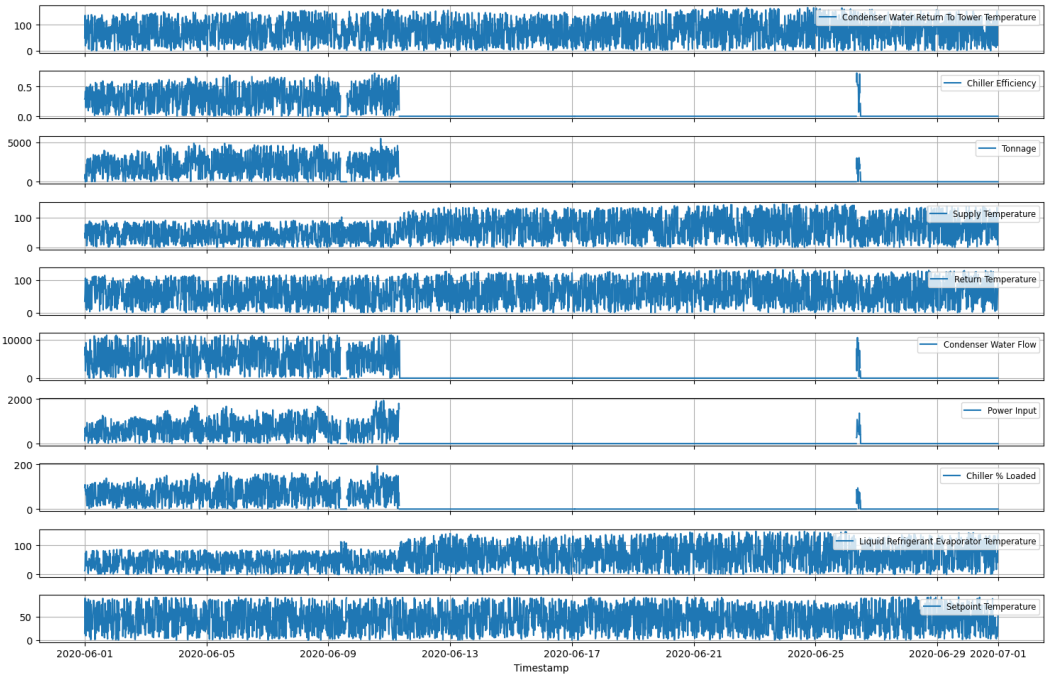


Figure 15: Snapshot of time series data from Chiller 6 for June 2020. Each subplot shows an individual sensor's trend over time.

The IoT Agent interacts with this telemetry data through structured utterances. By leveraging the standardized data provided by **AssetOpsBench**, the agent enables detailed, query-driven access to operational information across HVAC assets such as chillers and air handling units (AHUs) at IoT-enabled sites like the MAIN facility. Through these utterances, users can request both real-time and historical data, retrieve metadata, and download sensor readings for specific timeframes. This functionality supports knowledge and data queries, facilitating asset-level diagnostics, performance monitoring, and intelligent decision-making, even in noisy or incomplete data.

On the other hand, the TSFM Agent operates on sensor telemetry data—either retrieved via the IoT Agent or accessed directly from the sensor repository—to perform advanced time series analysis across HVAC systems. It supports a range of analytical tasks, including multivariate forecasting, and time series anomaly detection. At its core, the agent utilizes pre-trained time-series foundation models. For anomaly detection, the TSFM Agent applies a model-agnostic, post-hoc adaptive conformal method that requires no additional fine-tuning data, making it highly practical for real-world, resource-constrained deployments. By learning dynamic weighting strategies from prediction histories, it can detect distributional shifts and maintain calibrated, interpretable anomaly scores aligned with user-defined false alarm rates. Through structured utterances, users can invoke forecasting on specific variables (e.g., "Chiller 9 Condenser Water Flow"), fine-tune models with minimal data, or detect anomalies in historical trends—all with minimal configuration. This seamless integration of pre-trained models, adaptive analytics, and user-guided queries enables transparent, robust, and immediately deployable monitoring solutions tailored for critical industrial systems.

D.2 Failure Mode Datasets for FMSR Agent

The failure mode datasets in **AssetOpsBench** are modeled using the principles of *Failure Modes and Effects Analysis* (FMEA), a structured framework used in reliability engineering to identify failure risks, assess root causes and effects, and inform condition-based maintenance strategies. Each failure is defined by its mode, degradation mechanism, detection opportunity, and operational impact, enabling structured reasoning for both rule-based diagnostics and machine learning.

Failures in the dataset are annotated at the asset and subsystem levels, with a primary focus on centrifugal chillers. These failures reflect realistic degradation pathways and operational stressors derived from field experience. Each record in the failure model includes:

- **Failure Location and Component:** The subsystem or part where failure occurs, such as *bearings*, *gearboxes*, *impellers*, or *lubrication systems*.
- **Degradation Mechanism:** The underlying physical process driving the failure, including *wear*, *erosion*, *oil degradation*, *vibration-induced fatigue*, and *misalignment*.
- **Degradation Influences:** External or internal stressors such as *run time*, *two-phase process fluid*, *personnel error*, or *shock loading*.
- **Functional Failure Mode:** The resulting operational defect, such as *decreased oil pressure*, *audible noise*, *low head pressure*, or *capacity loss*.
- **Detection Opportunities:** Observable precursors or symptoms, including sensor readings (e.g., oil sampling, vibration signals), condition-based alarms, or inspection results.
- **Repair Time and Criticality:** Estimated downtime and classification of failure risk, supporting cost-based prioritization and scheduling.
- **Preventive Task Type:** Associated maintenance activity, such as *oil analysis*, *vibration analysis*, or visual inspection, tagged with effectiveness ratings and intervention intervals.

For example, *bearing wear*—a recurring failure across chiller subsystems—may arise from lubrication failure, misalignment, or fluid shock loading. This degradation is detectable via a combination of oil analysis and vibration monitoring, with failure symptoms including increased vibration, reduced oil pressure, and audible anomalies. Similarly, impeller erosion is linked to aging and two-phase fluid exposure, typically presenting as reduced capacity and lower head pressure.

Each maintenance task in the dataset is mapped to its detection mechanism and action type (e.g., condition monitoring vs. corrective repair), along with documentation on task content and recommended frequency. These structured records not only support early fault detection and diagnostics but also facilitate benchmarking of intelligent agents' reasoning over real-world degradation patterns and maintenance decisions.

Failures are temporally aligned with telemetry, enabling the study of degradation trajectories and pre-failure conditions. This integrated design makes the dataset suitable for supervised learning, causal inference, and evaluation of digital twins or predictive maintenance agents under realistic operating uncertainty.

To utilize the failure modes and their association with the sensors, we design FMSR (Failure Mode Sensor Relations) to interpret failure mode datasets within the **AssetOpsBench** framework, leveraging structured FMEA (Failure Modes and Effects Analysis) principles to link sensor telemetry with degradation mechanisms and operational failures. Using annotated failure records for assets such as centrifugal chillers, the FMSR Agent builds knowledge graphs and reasoning models that connect specific failure modes—like compressor overheating, evaporator fouling, or refrigerant valve failure—to their underlying causes and detectable symptoms. These failure modes are mapped to available sensor measurements (e.g., supply temperature, power input, vibration, flow rate) to identify observable precursors. For example, compressor overheating may be monitored through trends in power input, chiller efficiency, and evaporator temperature, while condenser fouling can manifest in abnormal return temperatures and flow rate deviations. Through structured utterances, users can query which failure modes are associated with specific sensors, which are critical for detecting a given failure, or even construct machine learning recipes for predictive modeling—such as anomaly models for chiller trips or excessive purging. The agent leverages this data to perform rule-based diagnostics, support causal analysis, and assist in condition-based maintenance planning. By aligning temporal sensor patterns with known failure signatures, the FMSR Agent enables explainable fault detection and root cause inference, ultimately enhancing reliability, maintainability, and transparency in HVAC operations.

D.3 Work Order Datasets for WO Agent

Table 5 provide the summary of datasets (as business objects) and the size for each dataset. Those work order datasets in **AssetOpsBench** provide a structured view of maintenance activity across in-

Table 11: Work Order Event Schema Definition

Field Name	Type	Description
wo_id	String	Unique identifier for the work order. Example: "L247402"
wo_description	String	Description of the work being done. Example: "CHILLER COMP OIL ANALYSIS"
collection	String	Broad group or system the work relates to. Example: "compressor"
components	String	Specific part or component being serviced. Example: "compressor"
primary_code	String	Code representing the main type of work. Example: "MT010"
primary_code_desc.	String	Description of the primary work code. Example: "Oil Analysis"
secondary_code	String	Sub-code under the primary category. Example: "MT010b"
secondary_code_desc	String	Description of the secondary code. Example: "Routine Oil Analysis"
equipment_id	String	Unique ID of the equipment. Example: "CU02013"
equipment_name	String	Human-readable name of the equipment. Example: "Chiller 13"
preventive	Boolean	Indicates if this is preventive maintenance. Example: TRUE
work_priority	Integer	Priority level of the work (e.g., 1–5). Example: 5
actual_finish	DateTime	Date and time when the work was completed. Example: "4/6/16 14:00"
duration	Duration	Total job time. Format: HH:MM. Example: "0:00"
actual_labor_hours	Duration	Actual labor time spent. Format: HH:MM. Example: "0:00"

Table 12: Alert Event Schema Definition

Field Name	Type	Description
equipment_id	String	Unique identifier for the equipment that triggered the alert. Example: "CWC04701"
equipment_name	String	Human-readable name of the equipment. Example: "Chiller 1"
rule_id	String	Identifier for the rule or condition that triggered the alert. Example: "RUL0021"
start_time	DateTime	Timestamp when the alert or event started. Example: "11/24/20 19:00"
end_time	DateTime	Timestamp when the alert or event ended. Example: "11/24/20 23:59"

dustrial assets, encompassing both preventive and corrective interventions using work orders. Each work order is associated with rich contextual data including equipment metadata, failure classification codes (e.g., ISO Failure Code, ISO Primary Failure Code), event logs, sensor-triggered alerts, and machine-generated anomalies. These records are linked temporally and causally, allowing agents to reason about asset history, detect recurring failure patterns, and recommend actions based on past interventions.

Table 13: Anomaly Event Schema Definition

Field Name	Type	Description
timestamp	DateTime	The date and time when the anomaly event was recorded. Example: "4/26/2014:14"
KPI	String	The key performance indicator being monitored (e.g., "Cooling Load").
asset_name	String	The name of the asset or equipment being measured. Example: "chiller 9"
value	Numeric	The actual measured value of the KPI at the given timestamp. Example: 25978710
upper_bound	Numeric	The upper threshold for the KPI. Exceeding this may indicate an anomaly.
lower_bound	Numeric	The lower threshold for the KPI. Falling below this may indicate an anomaly.
anomaly_score	Float	A score indicating how likely the data point is an anomaly (typically 0 to 1).

Table 14: Mapping Table: KPI Anomalies to Failure Codes

Field Name	Type	Example	Description
kpi_name	String	Cooling Load	Name of the key performance indicator exhibiting anomaly.
anomaly_type	String	High	Indicates the direction or nature of the anomaly (e.g., High, Low, Spike).
category	String	Operational Failures	Broad class of the failure (e.g., Control System, Structural, External, Human).
primary_code	String	OP004	Primary failure code associated with the anomaly.
pri._code_des	String	Incorrect Cooling Zone Operation	Explanation of the primary failure code.
seco._code	String	OP004c	More specific sub-code refining the root cause.
seco._code_des	String	Improperly Controlled or Shut Off Zones	Description of the secondary failure code.

The group of datasets distinguishes between core content objects (e.g., WorkOrders, Alerts, Events, Anomalies), metadata profiles, and relational structures that map alerts and anomalies to failure codes.

The individual event tables — *work orders* (Table 11), *alert events* (Table 12), and *anomaly events* (Table 13) — capture different but complementary signals related to equipment condition and behavior. To enable integrated analysis and causal reasoning, these events are unified into a common *event table schema* (Table 15), allowing temporal alignment and cross-type relationship discovery between maintenance actions, system warnings, and performance anomalies.

In addition, to support the linkage of failure code over the events, we provide two mapping tables: one that connects alert rules to likely failure codes, and another that maps KPI-based anomalies to structured failure categories (Tables of 16 and 14). These mappings enable agents to infer probable root causes from real-time signals and integrate data-driven insights with expert failure taxonomies.

Table 15: Unified Event Table Schema Definition

Field Name	Type	Description
event_id	String	Unique identifier for the event (can be work order ID, alert ID, anomaly ID, etc.). Example: "WO-16170"
event_group	String	High-level classification of the event source (e.g., "WORK_ORDER", "ALERT", "ANOMALY").
event_category	String	Sub-classification such as preventive maintenance ("PM"), corrective maintenance ("CM"), etc.
event_type	String	Specific code/type of the event (e.g., "MT001", "RUL0021").
description	String	Human-readable description of the event. Example: "Vibration Analysis" or "Refrigerant Leak".
equipment_id	String	Unique ID of the equipment involved in the event. Example: "CWC04701"
equipment_name	String	Name of the equipment. Example: "Chiller 1"
event_time	DateTime	Timestamp when the event occurred or was logged. Format: YYYY-MM-DD HH:MM:SS
note	String	Additional description for this event if necessary

Table 16: Mapping Table: Alert Rule to Failure Code

Field Name	Type	Example	Description
rule_id	String	RUL0012	Identifier for the alert rule triggered by a monitoring system.
rule_name	String	Chiller - Low Supply Temperature	Descriptive name of the alert rule logic or threshold condition.
primary_code	String	CS005	ISO failure code associated with the likely root cause.
primary_code_	String	Control System Malfunction	Human-readable explanation of the failure code.

This help us to develop WO agent to support grounded evaluation of diagnostic reasoning, task generation, and repair recommendation. More particularly, the WO agent analyze historical work orders to identify repeated maintenance issues and improve task scheduling. It processed historical work order, alerts (from IoT Agent) and anomalies (from TSFM agent) event, linking them to failure codes to support predictive maintenance recommendations. In the potential industrial applications, WO agent can complete to tasks of automating the interpretation of maintenance data, predicting future work orders, and bundling related tasks to reduce operational downtime.

E Additional Experiments

E.1 Example of Sample Scenario with Ground truth

We first prepared the ground truth that is verifiable.

Listing 3: Example FMSR task specification.

```
{
  "id": 105,
  "type": "FMSR",
  "deterministic": false,
  "characteristic_form": "the answer should contain a list of sensor
      names for asset wind turbine.",
  "text": "Provide some sensors of asset Wind Turbine.",
  "planning_steps": [
    "Provide some sensors of asset Wind Turbine."
  ],
  "execution_steps": [
    {
      "name": "get_available_sensor_information",
      "action": "Get Available Sensor Information",
      "arguments": "Wind Turbine",
      "outputs": "[a list of sensor names]"
    },
    {
      "name": "finish",
      "action": "Finish",
      "arguments": "",
      "outputs": ""
    }
  ],
  "execution_links": [
    {
      "source": "get_available_sensor_information",
      "target": "finish"
    }
  ]
}
```


E.2 AssetOpsBench: Execution Efficiency

In this section, we analyze AssetOpsBench execution efficiency of 7 LLMs, complementing the Leaderboard results in Section 5.1. Tables 17 and 18 present results from two multi-agent implementations. Metrics include the average number of steps taken per task and the average runtime (in seconds) per task.

In the **Agent-As-Tool** execution mode, most models demonstrate relatively stable planning behavior across both single-agent and multi-agent tasks. Compared to the Plan-Execute setting, models here generally take more steps but operate with greater runtime efficiency. `gpt-4.1` again exhibits strong performance, balancing a higher number of steps with moderate runtime, indicating precise control over tool invocation. Interestingly, `llama-3-70b-instruct` shows competitive efficiency, achieving the lowest runtime in both task categories despite slightly fewer steps, suggesting quicker tool usage or lower overhead per step. On the other hand, `mistral-large` exhibits extreme runtime variability, skewed by a pathological case involving prolonged `JSONReader` calls over large datasets. These results suggest that while tool-based execution benefits from more direct action control, its efficiency is highly sensitive to the invoked tools and data volume.

In the **Plan-Execute** setting, the number of steps required for single-agent tasks closely mirrors those of multi-agent tasks, indicating a tendency among LLMs to *over-plan even for relatively simple objectives*. This pattern reflects limited sensitivity to task complexity during the planning phase. Among all evaluated models, `gpt-4.1` consistently outperforms others, demonstrating both *minimal average steps* and *lowest runtime*, particularly in multi-agent tasks. This suggests that `gpt-4.1` leverages more effective internal representations and decision strategies, enabling efficient decomposition and execution of plans. In contrast, models like `granite-3-3-8b` and `llama-3-70b-instruct` show pronounced inefficiency, often executing significantly more steps and incurring higher computational costs. These results highlight a critical trade-off in Plan-Execute agents: while the architecture enforces task structure, its effectiveness heavily depends on the model's reasoning efficiency. Models lacking strong planning priors or execution alignment

Table 17: Execution Statistics for Agent-As-Tool: Average Steps and Runtime Per Task

Model	Single-Agent Tasks		Multi-Agent Tasks	
	Steps	Runtime (sec)	Steps	Runtime (sec)
gpt-4.1	6.0 ± 2.4	104 ± 178	6.4 ± 2.5	218 ± 371
mistral-large	4.9 ± 2.6	347 ± 19871	5.2 ± 2.2	289 ± 443
llama-3-405b-instruct	4.8 ± 2.5	250 ± 773	5.6 ± 2.2	255 ± 248
llama-3-70b-instruct	3.9 ± 1.6	101 ± 107	4.3 ± 2.1	151 ± 220
llama-4-maverick-17b-128e	4.3 ± 1.5	120 ± 258	4.5 ± 1.7	137 ± 175
llama-4-scout-17b-16e-instruct	4.4 ± 2.0	101 ± 87	5.8 ± 2.9	178 ± 157
granite-3-3-8b	5.3 ± 3.1	197 ± 240	6.6 ± 3.6	228 ± 256

High standard deviation is due to one outlier task requiring nearly 5 hours. It repeatedly invoked the `JSONReader` tool to process two years of historical data.

tend to generate unnecessarily long or suboptimal action sequences, especially in low-complexity settings.

Table 18: Execution Statistics of Plan-Execute Agents: Average Steps and Runtime per Task

Model	Single-Agent Tasks		Multi-Agent Tasks	
	Steps	Runtime (sec)	Steps	Runtime (sec)
gpt-4.1	2.6 ± 1.0	93.3 ± 105.6	2.9 ± 1.5	180.2 ± 122.6
mistral-large	2.7 ± 1.3	186.9 ± 206.9	3.0 ± 1.4	209.7 ± 139.1
llama-3-405b-instruct	3.1 ± 1.9	208.3 ± 176.5	4.0 ± 1.5	224.4 ± 99.7
llama-3-70b-instruct	6.7 ± 1.5	381.8 ± 240.2	6.5 ± 0.9	369.6 ± 151.9
llama-4-maverick-17b-128e	4.0 ± 1.9	384.6 ± 611.6	3.9 ± 1.2	376.8 ± 281.0
llama-4-scout-17b-16e	3.9 ± 2.0	172.1 ± 114.7	4.4 ± 1.5	218.1 ± 105.4
granite-3-3-8b	5.2 ± 1.4	413.3 ± 418.2	5.1 ± 1.3	432.9 ± 294.7

Conclusion. While the **Plan-Execute** architecture demonstrates greater efficiency—requiring fewer steps and exhibiting lower runtime variability across tasks—our evaluation shows that **Agent-As-Tool** significantly outperform in task performance metrics. For example, `gpt-4.1` achieves 65% task completion, 77% data retrieval accuracy in the Agent-As-Tool setting, compared to only 38–44% on most metrics in Plan-Execute. Similarly, `llama-4-maverick-17b-128e-instruct` excels in both setups but scores notably higher in Agent-As-Tool, achieving 59–78% on core performance metrics versus 45–57% in Plan-Execute.

This pattern is consistent across most models: **Agent-As-Tool** incur higher execution costs but deliver better reasoning fidelity. Conversely, **Plan-Execute** agents—while faster and more structured—often struggle with complex retrieval, verification, and consistency tasks. These findings suggest a fundamental trade-off: Plan-Execute offers process efficiency, while Agent-As-Tool yield higher end-task quality—a crucial insight for selecting agent architectures based on application goals such as throughput vs. correctness.

E.2.1 Deep Investigation of Agent-As-Tool Performance

To evaluate the capabilities of various large language models (LLMs) across a range of industrial-relevant task categories, we present a radar chart (See Figure 16) comparison covering five key dimensions: *IoT-focused reasoning*, *Failure Mode and Sensor Reasoning (FMSR)*, *Time Series and Fault Modeling (TSFM)*, *Work Order (WO) understanding*, and *End-to-End task integration*. The chart illustrates normalized performance scores for each model based on task-specific benchmarks, with higher values indicating stronger task alignment. Among the models compared, `gpt-4.1-2025-04-14` demonstrates the most consistent and well-rounded performance, achieving near-saturation in FMSR (100%) and strong results in End-to-End integration. In contrast, `granite-3-3-8b-instruct` and `llama-3-3-70b-instruct` perform well in TSFM and FMSR but underperform in WO-related tasks, which are particularly challenging due to their dependence on structured document comprehension and task planning. Notably, the

`llama-4-maverick` model shows promising results in WO and End-to-End integration, indicating a potential optimization for cross-domain contextual reasoning. This visualization provides a holistic view of model strengths and trade-offs, offering insights for selecting and fine-tuning LLMs in complex, multimodal industrial applications.

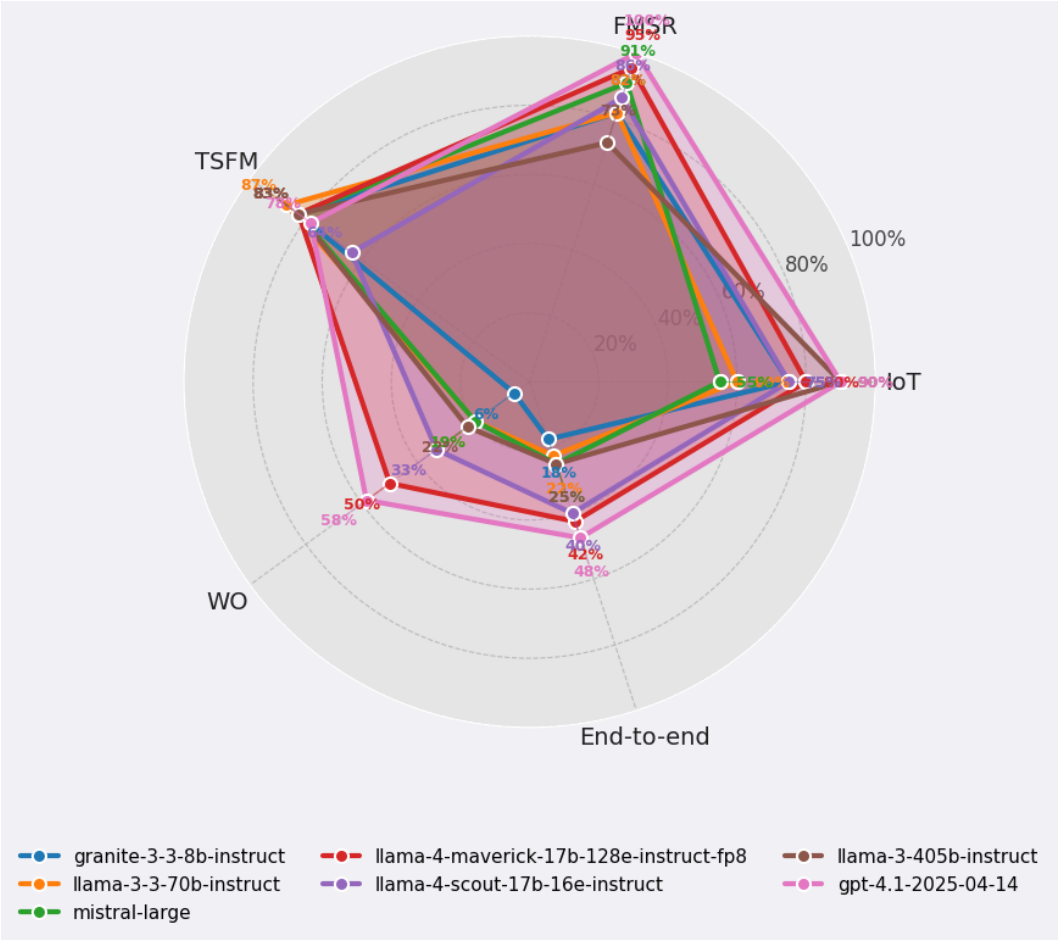


Figure 16: Task wise distribution of the Accomplished Tasks.

E.3 Uncertainty Analysis

As discussed in Section 5, the evaluation agent was run five times to produce reliable performance metrics. Table 19 shows the inter-rater agreement across these five evaluation runs, along with the derived uncertainty (computed as 1 – agreement). The average agreement and uncertainty across all metrics are also reported.

Table 19: Inter-rater agreement and derived uncertainty across five evaluation runs.

Metric	Agreement	Uncertainty
Task Completion	0.9731	2.69%
Data Retrieval Accuracy	0.9697	3.03%
Generalized Result Verification	0.9681	3.19%

E.4 HUMAN VALIDATION

We conducted human validation tests using Google Forms. As shown in Screenshots 17, each domain experts are presented with the original task description, the agent's reasoning and final answer, and a checklist of six evaluation dimensions. Each dimension was assessed using binary judgments (True/False), allowing for structured comparison between human and model evaluations. We distributed 4 forms where each form consists of 10 samples and eventually collect 240 data points. We report the results in Section 5.1.

AssetOpsBench Human Judge Test

Group: 1, Index: 0

Task:
Generate a machine learning recipe for detecting overheating failure for Chiller 6. Result should include feature sensors and target sensor.

Final answer:
For detecting "Compressor Overheating" failure in Chiller 6, the relevant feature sensors are: 'Chiller 6 Power Input', 'Chiller 6 Evaporator Water Outlet Temperature', 'Chiller 6 Condenser Water Inlet Temperature', 'Chiller 6 Condenser Water Outlet Temperature', and 'Chiller 6 Compressor Discharge Temperature'. The target sensor is 'Chiller 6 Compressor Discharge Temperature'. A machine learning model can be trained using these features to predict the likelihood of compressor overheating. ## Step 1: Understand the Task The task is to generate a machine learning recipe for detecting overheating failure for Chiller 6, including feature sensors and target sensor. ## Step 2: Identify Failure Modes for Chiller 6 The failure modes for Chiller 6 are identified as: 'Compressor Overheating: Failed due to Normal wear, overheating', 'Heat Exchangers: Fans: Degraded motor or worn bearing due to Normal use', 'Evaporator Water side fouling', 'Condenser Water side fouling', 'Condenser Improper water side flow rate', 'Purge Unit Excessive purge', 'Refrigerant Operated Control Valve Failed spring'. ## Step 3: Determine Relevant Sensors for "Compressor Overheating" The relevant sensors for monitoring "Compressor Overheating" are: 'Chiller 6 Power Input', 'Chiller 6 Evaporator Water Outlet Temperature', 'Chiller 6 Condenser Water Inlet Temperature', 'Chiller 6 Condenser Water Outlet Temperature', and 'Chiller 6 Compressor Discharge Temperature'. ## Step 4: Compile Machine Learning Recipe For detecting "Compressor Overheating" failure in Chiller 6, the relevant feature sensors are: 'Chiller 6 Power Input', 'Chiller 6 Evaporator Water Outlet Temperature', 'Chiller 6 Condenser Water Inlet Temperature', 'Chiller 6 Condenser Water Outlet Temperature'. The target sensor is 'Chiller 6 Compressor Discharge Temperature'. A machine learning model can be trained using these features to predict the likelihood of compressor overheating. ## Step 5: Provide Final Answer For detecting "Compressor Overheating" failure in Chiller 6, the relevant feature sensors are: 'Chiller 6 Power Input', 'Chiller 6 Evaporator Water Outlet Temperature', 'Ch

Thinking process:
Thought: I need to generate a machine learning recipe for detecting overheating failure for Chiller 6, which includes identifying the relevant feature sensors and the target sensor. I will use FMSRAgent to achieve this.

Action: Tool Name: FMSRAgent , Tool Parameter: Action Input 1: request=Generate a machine learning recipe for detecting overheating failure for Chiller 6, including feature

Figure 17: Google Forms: questionnaire to domain experts for human validation

E.5 LLM-AS-A-JUDGE EVALUATION AGENT

Following the system prompt that we used for generating a rubric metric for the evaluation agent.

You are a critical reviewer tasked with evaluating the effectiveness and accuracy of an AI agent's response to a given task. Your goal is to determine whether the agent has successfully accomplished the task correctly based on the expected or characteristic behavior.
Evaluation Criteria:
1. **Task Completion**:
- Verify whether the agent executed all required actions (e.g., using the correct tools, retrieving data, performing the necessary analysis).
- Ensure the response aligns with the predefined expected behavior for task completion.
2. **Data Retrieval & Accuracy**:
- Confirm that the correct asset, location, time period, and sensor (if applicable) were used.
- Check that the retrieved data and results (forecasting, anomaly detection, etc.) are correct and consistent with the task requirements.
3. **Generalized Result Verification**:
- Task Type Verification: Assess if the agent returned the expected results for the task type (forecasting, anomaly detection, classification, etc.).
- Forecasting: Ensure forecasts cover the specified future period.
- Anomaly Detection: Verify that anomalies were correctly detected when expected.
- Other Tasks (e.g., classification): Check that results match expected format and values.
- Comparison with Expected Output: Validate that results match the characteristic answer.
- Data Integrity: Ensure correct data (sensor, time period) was used and output format is consistent.
Inputs:
Question: {question}
Characteristic Answer (Expected Behavior): {characteristic_answer}
Agent's Thinking: {agent_think}
Agent's Final Response: {agent_response}
Output Format:
Provide your review strictly in JSON format without any additional text or Markdown.
{
"task_completion": true/false,
"data_retrieval_accuracy": true/false,
"generalized_result_verification": true/false,
"suggestions": "Optional. Recommended actions to improve the agent's response if needed."
}
(END OF RESPONSE)
Evaluate the agent's performance according to the above criteria.

Table 20: Prompt instruction for LLM-as-a-judge evaluation agent

Based on the human validation results shown in Section 5.1, `llama-4-maverick` is selected to be the LLM of evaluation agent. Table 20 is the prompt instruction to the evaluation agent, which outlines the specific evaluation dimensions, constraints, and response formatting guidelines that the model follows when scoring task outputs. The evaluation criteria is also provided to human judges which ensures consistency across evaluations.

E.6 ABLATION EXPERIMENT

In this section, we present the detailed report of the ablation study. We fixed the Agent-As-Tool paradigm and conducted both sets of experiments.

E.6.1 DISTRACTOR AGENTS DETAIL

We have introduced 10 distractor agents to intentionally increase the complexity and ambiguity for global agents. Table 21 categorizes these agents based on their respective domains and functional roles. The set includes both general-purpose agents, such as those for echoing inputs or handling off-topic queries, and domain-specific agents focused on tasks like predictive maintenance, sensor data summarization, and edge ML deployment. This taxonomy enhances the realism of multi-agent environments by supporting modular integration and introducing controlled confusion.

Table 21: Agent Types and Their Roles

Agent Name	Domain	Description
`Echo Agent`	General	Repeats the input verbatim; useful for debugging and testing input-output coherence.
`OffTopic Agent`	General	Provides unrelated facts or trivia when a query is off-topic or not recognized.
`Customer SupportAgent`	Support Operations	Handles customer-related issues like password resets, login errors, and service availability.
`SRE Agent`	Site Reliability	Diagnoses performance degradation, system downtime, and infrastructure issues.
`Frontend DevAgent`	Software Engineering	Assists with frontend UI/UX concerns, React, JavaScript frameworks, and rendering bugs.
`HRPolicy Agent`	Human Resources	Answers HR-related queries like leave policy, benefits, and compliance rules.
`SensorData Summarizer`	Industrial IoT	Summarizes time-series data from sensors, highlighting trends and anomalies.
`Historical TrendsAgent`	Analytics	Extracts and interprets historical asset data to identify failure patterns or optimization opportunities.
`EdgeML Agent`	Edge Computing	Recommends tools and strategies for deploying ML models on edge hardware with limited resources.
`RULPredictor Agent`	Predictive Maintenance	Estimates the remaining useful life (RUL) of assets using sensor data and degradation models.

E.6.2 Impact of in-context examples

Table 22 provides a detailed comparison of gpt-4.1 and granite-3-3-8b with and without in-context examples on a subset of single-agent benchmark tasks. Consistent with our main findings, in-context examples were critical for enabling effective reasoning and coordination.

Table 22: Comparison of `gpt-4.1` and `granite-3-3-8b` With/Without In-Context Examples (# of Tasks = 65)

Model	In-Context Examples	Task Completion	Data Retrieval Accuracy	Generalized Result Verification
gpt-4.1	Yes	52	57	55
granite-3-3-8b	Yes	40	44	41
gpt-4.1	No	22	21	24
granite-3-3-8b	No	2	3	3

Key Results: Removing in-context examples led to a dramatic drop in performance for both models. gpt-4.1 dropped from an average of 80% (with context) to 33% (without), while granite-3-3-8b

fell from 60% to just 3% (Section E.6). These results reinforce the conclusion that in-context examples are essential for ReAct-style reasoning in LLM-based agents. We did not select tasks from WO and E2E since their performance is already poor.

E.7 Plan-Execute Reference-Based Scoring

Evaluation Setup

To assess the fidelity of generated outputs, we perform **reference-based scoring** using ROUGE metrics. This evaluation is limited to the **Plan-Execute** paradigm to maintain consistency and preserve the experimental flow.

ROUGE metrics used include:

- `rouge1`: unigram (1-gram) overlap between generated and reference outputs.
- `rouge2`: bigram (2-gram) overlap.
- `rougeL`: longest common subsequence between generated and reference sequences.
- `rougeLsum`: line-wise longest common subsequence for multi-line outputs.

Results Summary

ROUGE scores highlight model differences in n-gram and sequence-level fidelity. Table 23 presents sample scores for representative models across Plan-Execute outputs.

Table 23: ROUGE-based reference scoring for Plan-Execute outputs (selected models).

Model	`rouge1`	`rouge2`	`rougeL`	`rougeLsum`
`llama-3-405b-instruct`	0.406	0.243	0.337	0.381
`mixtral-8x7b-instruct-v01`	0.424	0.259	**0.343**	0.401
`llama-3-3-70b-instruct`	0.297	0.172	0.242	0.280
`gpt-4.1-2025-04-14`	0.354	0.182	0.289	0.335
`granite-3-3-8b-instruct`	0.373	0.214	0.291	0.353
`mistral-large`	**0.420**	**0.251**	**0.343**	**0.404**
`llama-4-maverick`	0.403	0.240	0.325	0.383

Analysis

- Top-performing models such as `llama-3-405b-instruct` and `mixtral-8x7b-instruct-v01` achieve `rouge1` $\approx$ 0.42, `rouge2` $\approx$ 0.26, and `rougeL` $\approx$ 0.34, indicating strong n-gram and sequence-level fidelity.
- Smaller or older models exhibit lower ROUGE scores, reflecting weaker lexical alignment with reference trajectories.
- Overall, Plan-Execute outputs maintain higher alignment with reference trajectories, demonstrating that this paradigm supports more faithful generation for skilled reasoning tasks.
- The distribution of ROUGE metrics also reflects diversity in output complexity, as longer or multi-step reasoning tasks tend to lower ROUGE scores despite semantic correctness.

Reference-based scoring provides a quantitative measure of textual fidelity across different models under the Plan-Execute paradigm. These results support model comparison, highlight the impact of LLM size and capabilities, and offer a reproducible benchmark for future studies.

E.8 Reference-Based Scoring for Agent-As-Tool

In the **Agent-As-Tool** setting, the agent follows a *think–act–observe* cycle without a pre-planning phase. To evaluate reasoning quality, we extract the internal *thinking* segments and compute ROUGE scores against concise reference trajectories. Because ROUGE measures lexical overlap, differences in verbosity strongly affect the outcome.

Table 24: ROUGE-based comparison for the **Agent-As-Tool** setting. Scores are computed on the extracted *thinking* segments of each trajectory. Longer generations reduce lexical overlap with concise references, lowering ROUGE despite potentially richer content.

Model	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-Lsum	#Samples	Pred. Avg. Words	GT Avg. Words
mistral-large	**0.3691**	**0.1933**	**0.2971**	**0.3124**	40	**83.0**	29.85
llama-3-3-70b-instruct	0.3661	**0.1963**	0.2971	**0.3177**	40	47.8	29.85
llama-3-405b-instruct	0.3394	0.1673	0.2740	0.2787	40	82.42	29.85
llama-4-scout-17b-16e-instruct	0.3126	0.1522	0.2398	0.2621	38	100.32	29.84
llama-4-maverick-17b-128e-instruct-fp8	0.2560	0.1252	0.2067	0.2273	29	112.66	26.34
granite-3-3-8b-instruct	0.2473	0.1001	0.1867	0.2079	36	164.36	29.19
gpt-4.1-2025-04-14	**0.1628**	**0.0816**	**0.1332**	**0.1389**	40	**277.12**	29.85

Results. Table 24 reports ROUGE-1/2/L scores along with generation lengths. `mistral-large` achieves the highest performance with `ROUGE-1` ≈0.37, `ROUGE-2` ≈0.19, and `ROUGE-L` ≈0.30, followed closely by `llama-3-3-70b-instruct` and `llama-3-405b-instruct`. These models generate reasoning traces of moderate length (48–83 words on average), which aligns well with the reference answers (30 words) and preserves lexical fidelity.

In contrast, models such as `gpt-4.1` and `granite-3-3-8b-instruct` produce significantly longer outputs (up to 277 words on average), resulting in the lowest ROUGE scores despite potentially valid reasoning steps.

Summary. Models with output lengths closer to the reference (e.g., `mistral-large`, `llama-3-70B`) achieve higher lexical alignment. However, low-scoring models like `gpt-4.1` may still exhibit rich and correct reasoning, suggesting that token length and prompting strategy—rather than reasoning quality alone— drive ROUGE differences in the Agent-As-Tool paradigm.

E.9 Execution Chain Evaluation

To systematically evaluate agent task execution, we design a **chain-based execution scoring** method. In many scenarios, an agent performs a sequence of steps corresponding to *Think-Act-Observe* cycles. Ground truth data provides the expected sequence of steps for each task. Each executed step contains a `name` (representing the action) and an `arguments` field.

E.10 Scoring Method

Our scoring approach compares an agent's executed sequence with the ground truth sequence using three criteria:

1. **Step Matching:** The `name` of each executed step is matched to the corresponding ground truth step. Unlike exact matching, we allow fuzzy matching based on string similarity using a threshold to account for minor variations in step names.
2. **Argument Similarity:** Step arguments are treated as strings and compared using a ROUGE-like similarity metric (via `difflib.SequenceMatcher`). This captures cases where the agent produces slightly different or paraphrased arguments.
3. **Sequence Coverage and Order:**
   - Coverage penalizes missing ground truth steps.
   - Extra steps are penalized proportionally.
   - Order preservation is evaluated: steps executed out-of-order incur a penalty.

The final **Execution Chain Score** for a single trajectory is computed as:

$$\text{Score} = (\text{Average argument similarity over matched steps}) \times (1 - \text{extra step penalty}) \times (1 - \text{order penalty})$$

This produces a single scalar in $[0, 1]$ summarizing how closely an agent's execution matches the ground truth.

E.11 RESULTS

We applied this method to evaluate several state-of-the-art LLMs across trajectory datasets using the **Agent-As-Tool** paradigm, where agents do not perform upfront planning but instead follow *Think-Act-Observe* cycles.

Table 25 reports the **average execution scores** per model:

Model	Average Execution Score
meta-llama/llama-3-405b-instruct	0.118
meta-llama/llama-4-maverick-17b-128e-instruct-fp8	0.077
meta-llama/llama-4-scout-17b-16e-instruct	0.092
ibm/granite-3-3-8b-instruct	0.040
meta-llama/llama-3-3-70b-instruct	0.031
mistralai/mistral-large	0.113
openai-azure/gpt-4.1-2025-04-14	0.117

Table 25: Average Execution Chain Scores for different LLM models. Scores reflect alignment with ground truth sequences in terms of step name, argument similarity, and sequence coverage.

E.12 DISCUSSION

The results indicate that:

- `meta-llama/llama-3-405b-instruct`, `mistral-large`, and `gpt-4.1-2025-04-14` achieve the highest alignment with ground truth steps, demonstrating better handling of multi-step task execution in the Agent-As-Tool setting.
- Larger models such as `llama-4-maverick` and `llama-4-scout` have moderate scores, suggesting that complexity alone does not guarantee faithful execution.
- Smaller or older models, including `granite-3-3-8b` and `llama-3-3-70b`, exhibit lower scores, primarily due to missing steps, extra steps, or argument discrepancies.

Overall, this evaluation provides a quantitative, interpretable measure of how closely an agent's executed actions match the intended ground truth, complementing other performance metrics such as reference-based scoring (ROUGE) or semantic verification.

E.13 RUNTIME AND COST ANALYSIS

Table 26 provides a representative comparison of total runtime and estimated cost for executing the full 140+ utterance task suite using the Agent-As-Tool paradigm. Average tokens per task and total cost are shown for different LLMs.

Table 26: Runtime and estimated cost for executing 140+ utterance tasks using the Agents-as-Tools paradigm.

LLM	Provider	Avg Tokens per Task	Total Cost (USD)
gpt-4.1	OpenAI	≈3,664	$300.00
llama-4-maverick	Watsonx	≈3,730	$130.00

E.14 ALTERNATE DATASETS AND SCENARIOS

We use two recent datasets for condition monitoring of industrial assets: the Metro Train MetroPT-3 dataset and the Hydraulic System dataset. Both datasets are hosted on UCI and provide programmatic access to descriptions and metadata. Using the dataset description and failure locations, we created 15 scenarios for MetroPT-3 and 17 scenarios for Hydraulic System pumps. Two representative examples are provided below.

Table 27: Sample predictive maintenance scenarios for MetroPT-3 and Hydraulic System assets.

ID	Scenario Description
1	For asset `hp_1`, can severe internal pump leakage on 2024-01-31 be detected using sensor data from the preceding 100 days? Which sensor trends provide key clues within this timeframe?
2	Consider asset `mp_1`. From the compressor sensor data collected between May 29 and June 4, 2020, can we assess the likelihood of an air leak failure occurring within the subsequent week starting June 5? Is preventive maintenance advisable?

E.15 Scenarios Creation and Distribution for Product Testing

Based on business unit requirements and in collaboration with domain experts, we created 42 scenarios for detecting asset health to conduct the benchmark study, primarily using work order data released as part of this project. Each scenario follows the prescribed format described in the main paper. A representative example is shown below:

```
{
  "id": 1000,
  "file": "Air Handling Unit_615152AC_insights_prompt.txt",
  "text": "You are an expert in Air Handling Unit maintenance and
           reliability analysis. Your task is to analyze provided
           asset_details_facts and workorder_facts...",
  "type": "System Health",
  "category": "Asset Analysis",
  "deterministic": true,
  "characteristic_form": "The expected condition of the asset is
  'Not enough data' because only 4 work orders are available."
}
```

As mentioned in the main text (Page 4), one of the task types is **System Health**, aimed at evaluating the condition of an asset based on recent system records (typically work orders, alerts, etc.) and raising flags such as *good* or *needs attention*. Table 28 summarizes the coverage of the 42 scenarios across asset classes.

Table 28: Distribution of scenarios across asset types/classes.

Asset Type/Class	Number of Unique Instances
Air Handling Unit	9
CRAC	10
Chiller	10
Pump	8
Boiler	5

This diversity spans both horizontal coverage (different asset classes) and vertical variation (multiple instances within each class), providing a robust testbed for evaluating agent generalizability and performance across operational conditions.

Scenarios Task Distribution

All 42 scenarios fall under the **Asset Health** category and primarily rely on work order information. Each scenario captures distinct aspects of asset behavior, reflecting operational variability. Token count analysis provides insight into scenario complexity.

Over 60% of scenarios (26/42) fall in the 767–2,841 token range, reflecting mostly concise formats. A long-tailed distribution exists to ensure LLMs handle both compact and extended input contexts.

Table 29: Token count statistics for 42 Asset Health scenarios.

Statistic	Value
Total scenarios	42
Median	2,277 tokens
Mean	3,695 tokens
Standard Deviation	3,125 tokens
Minimum–Maximum	777–11,098 tokens
Mode	1,316 tokens

Table 30: Token count distribution across scenarios.

Token Range	# Scenarios
(767 – 2,841]	26
(2,841 – 4,905]	6
(4,905 – 6,970]	1
(6,970 – 9,034]	3
(9,034 – 11,098]	6

Scenario Execution and Evaluation

The 42 scenarios were executed across three models, resulting in 126 executions. Each execution generates an output trajectory, which is subsequently analyzed by the Evaluation Agent across five runs, yielding 630 evaluation instances. The Evaluation Agent compares outputs against the characteristic form described in the scenario examples to calculate automated metrics. Manual review was used to validate the final column of results, identifying only one case (Granite) where the model overconfidently claimed task completion.

Performance Insights

The scenarios primarily assess LLMs' analytical skill—the ability to interpret provided information and generate appropriate conclusions. Agents such as FMSR, which excel in skill-based reasoning tasks, demonstrate strong performance, particularly in single-agent communication settings.

F Algorithmic Procedure for New Emerging Failure Mode Discovery

To support adaptive evaluation of multi-agent LLM systems, this appendix outlines the implementation details behind the failure discovery process. While the main text presents the empirical distribution of failure types—including emergent patterns—this appendix focuses on the structured methodology used to extract and cluster novel failure behaviors beyond the MAST (Multi-Agent System Failure Taxonomy) Cemri et al. (2025). The evaluation spanned 881 multi-agent trajectories, drawn from diverse language model configurations. Trajectory distribution by model is as follows:

- `mistral-large`: 145 trajectories
- `llama-3-405b-instruct`: 145 trajectories
- `llama-3-3-70b-instruct`: 145 trajectories
- `llama-4-maverick-17b-128e-instruct-fp8`: 125 trajectories
- `llama-4-scout-17b-16e-instruct`: 111 trajectories
- `gpt-4.1-2025-04-14`: 105 trajectories
- `granite-3-3-8b-instruct`: 105 trajectories

Among the 881 utterance execution trajectories analyzed using an LLM-as-a-judge framework (selected LLM judge model - *openai-azure/gpt-4.1-2025-04-14* as the LLM judge) to identify the causes of multi-agent AI failures, we found that—beyond the existing MAST categories—185 trajectories exhibited one additional failure reason, while 164 trajectories contained two distinct additional failure reasons. This highlights the empirical necessity of taxonomy expansion to capture compound and emergent failure patterns in real-world deployments. To extend the original MAST taxonomy, we conducted a structured analysis of novel multi-agent system failures observed in recent interaction traces. This subsection details our identification methodology and explains how the resulting failure modes align with the MAST framework.

F.0.1 Algorithm for Emerging Failure Modes Clustering

To systematically identify and normalize *emerging failure modes* observed in multi-agent LLM system interactions, we introduce a structured algorithmic framework based on semantic embedding and unsupervised clustering. This process abstracts unanticipated failure patterns into representative categories that either align with or extend the predefined MAST taxonomy.

Definitions and Notation. Let:

- $T = \{t_1, \ldots, t_n\}$: Set of multi-agent execution trajectories.
- $\mathcal{M}$: The predefined MAST taxonomy of failure types.
- $F = \{f_1, f_2, \ldots, f_m\}$: Set of *emerging failure mode* descriptions not covered by $\mathcal{M}$, extracted from LLM-as-a-judge evaluations.
- $\phi : \mathcal{S} \to \mathbb{R}^d$: Sentence embedding function (e.g., Sentence-BERT).
- $\mathbf{E} = [\phi(f_1), \ldots, \phi(f_m)]^\top \in \mathbb{R}^{m \times d}$: Matrix of embedded failure descriptions.
- $\mathcal{C} = \{C_1, \ldots, C_k\}$: Partition of F into k clusters, each with centroid $\boldsymbol{\mu}_j$.

Step 1: Emerging Failure Mode Extraction. Each trajectory $t_i \in T$ is evaluated by an LLM-as-a-judge to identify:

- Labeled failure types from the MAST taxonomy $\mathcal{M}$.
- Up to two *emerging* failure descriptions $f_{i1}, f_{i2} \notin \mathcal{M}$.

The full set of novel descriptions is aggregated as:

$$F = \bigcup_{i=1}^{n} \{f_{i1}, f_{i2}\} \setminus \text{NULL}$$

Step 2: Semantic Embedding. Each emerging failure mode $f_i \in F$ is transformed into a d-dimensional vector:

$$\mathbf{e}_i = \phi(f_i), \quad \forall f_i \in F$$

$$\mathbf{E} = \begin{bmatrix} \phi(f_1)^\top \\ \phi(f_2)^\top \\ \vdots \\ \phi(f_m)^\top \end{bmatrix} \in \mathbb{R}^{m \times d}$$

Step 3: Optimal Clustering via K-Means. To discover latent groups of semantically similar failure descriptions, we apply K-Means clustering over the embeddings $\mathbf{E}$. The silhouette score for a given point i is:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Where:

- $a(i)$: Mean distance from $\mathbf{e}_i$ to other points in the same cluster.
- $b(i)$: Minimum mean distance from $\mathbf{e}_i$ to points in a different cluster.

The optimal number of clusters is selected as:

$$k^* = \arg\max_k \text{SilhouetteScore}(k)$$

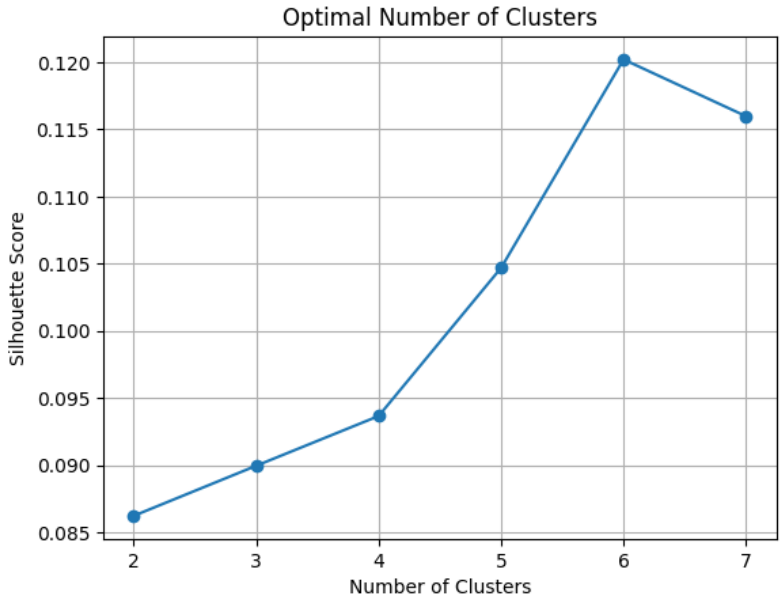


Figure 18: Silhouette analysis showing optimal number of clusters $k^* = 6$.

Step 4: Cluster Center Selection. For interpretability, we select a representative f_j^* from each cluster C_j as the most centrally located failure mode:

$$f_j^* = \arg\min_{f_i \in C_j} \|\phi(f_i) - \boldsymbol{\mu}_j\|_2$$

Step 5: Taxonomy Alignment. Each representative failure mode f_j^* is reviewed and mapped to one or more MAST categories:

- **Specification Failures**
- **Inter-Agent Failures**
- **Task Verification Failures**

Failures that exhibit characteristics of multiple categories are marked as *compound* or *intersectional*, suggesting the need for extensions to the base taxonomy.

Outputs. The algorithm yields:

- A clustered taxonomy $\mathcal{C} = \{C_1, \ldots, C_{k^*}\}$ of emerging failure modes.
- Canonical representatives $\{f_1^*, \ldots, f_{k^*}^*\}$ for each cluster.
- Category mappings for taxonomy refinement or extension.
- Frequency statistics per failure type for prioritization.

F.0.2 Methodology: Semantic Clustering of Emergent Failures

Building on the formal clustering algorithm outlined above, we implemented a practical instantiation of the pipeline to organize the large volume of emerging failure mode descriptions identified by the LLM-as-a-judge. We found lots of new and different behaviors when we first looked. But a closer look showed that many of them were either just repeating the same idea or were only slightly

different versions of the same core problems. To distill these into interpretable categories, we applied a semantic clustering methodology grounded in high-dimensional language representations.

Each emerging failure description was manually or programmatically summarized into a concise label and explanatory text. These summaries were then embedded into a semantic vector space using the `all-MiniLM-L6-v2` model from the SentenceTransformer library, yielding a set of dense, comparable embeddings suitable for clustering.

We applied the KMeans algorithm to group these embeddings into semantically coherent clusters. To determine the optimal number of clusters, we computed silhouette scores for values of k ranging from 2 to 7 and selected the value that maximized mean silhouette score (see Figure 18). This analysis yielded an optimal configuration of $k^* = 6$ clusters.

For interpretability, each cluster was assigned a canonical label derived from the failure mode description closest to the cluster centroid. This process produced six representative categories of emerging failure modes, summarized below:

- *Cluster 0: Lack of Error Handling for Tool Failure* (53 cases, 10.3%)
  Agents fail to detect or appropriately respond to tool invocation errors.
- *Cluster 1: Failure to Incorporate Feedback* (41 cases, 8.0%)
  Agents ignore or inadequately adjust to feedback from other agents or tools.
- *Cluster 2: Invalid Action Formatting* (27 cases, 5.3%)
  Output includes syntactic or structural errors that prevent execution.
- *Cluster 3: Overstatement of Task Completion* (122 cases, 23.8%)
  Agents claim completion without satisfying task criteria or producing valid outcomes.
- *Cluster 4: Extraneous or Confusing Output Formatting* (110 cases, 21.4%)
  Responses contain unnecessary verbosity, ambiguous structure, or misleading formatting.
- *Cluster 5: Ineffective Error Recovery* (160 cases, 31.2%)
  Agents fail to resolve prior mistakes or restart workflows effectively after failure.

These cluster-derived failure modes serve as canonical extensions to the base MAST taxonomy, revealing previously unclassified behaviors that frequently arise in multi-agent LLM interactions. Their emergence underscores the value of inductive, embedding-based clustering for scalable failure mode discovery and taxonomy refinement.

F.0.3 Taxonomic Alignment with MAST of Emergent Failures

These emergent failure modes reveal both alignment and tension with the original MAST taxonomy. Each cluster can be mapped to one or more of MAST's three core failure categories, but many straddle boundaries or reveal overlapping failure dynamics:

- **Specification Failures:**
  - *Overstatement of Task Completion* and *Extraneous Output Formatting* reflect unclear success criteria, misunderstood task scopes, or ambiguous output specifications.
- **Inter-Agent Failures:**
  - *Failure to Incorporate Feedback* and *Lack of Error Handling for Tool Failure* indicate coordination breakdowns or limited adaptivity in dynamic environments.
- **Task Verification Failures:**
  - *Invalid Action Formatting* and *Ineffective Error Recovery* highlight failures in runtime execution monitoring, verification, and correction procedures.

Several emergent failure types cut across multiple categories, underscoring the complexity and interdependence of failure dynamics in real-world multi-agent systems. These findings motivate future refinement of MAST to support cross-category failure representation and compound behavior tracking.

This failure mode analysis contributes both methodologically and substantively to multi-agent system evaluation. Methodologically, it introduces a scalable pipeline for inductively discovering and

structuring new failure behaviors using LLM-judged outputs and semantic clustering. Substantively, it extends the empirical coverage of the MAST taxonomy by surfacing nuanced, real-world failure patterns that reflect the increasing complexity of autonomous agent collaboration.

These insights not only validate the need for flexible taxonomic frameworks but also point to the importance of diagnostics that evolve with model behavior. As LLM-based agents continue to scale in capability and deployment scope, the ability to detect emergent, intersectional failures becomes a foundational requirement for reliable multi-agent orchestration.

F.1 Impact of Agent Communication on Benchmark Performance

In our benchmark, the parameter `enable_agent_ask` controls whether the agent can ask clarifying questions during task execution. In the Agent-As-Tool architecture, planning is performed incrementally, and agent communication can influence task performance, unlike the Plan-Execute paradigm where planning is done upfront.

For a fair comparison, our initial experiments used the default setting (`enable_agent_ask=False`), preventing agents from asking questions beyond the given task. Table 2 highlights that certain failures, such as not asking clarifying questions, contribute to approximately 10% of errors. To evaluate the impact of agent communication, we set `enable_agent_ask=True` and re-ran the experiments across multiple models. Table 31 summarizes the results.

Table 31: Benchmark performance with and without agent communication enabled.

Model	**enable_agent_ask=True**	**enable_agent_ask=False**
gpt-4.1-2025-04-14	63%	65%
lama-4-maverick	66%	59%
llama-3-405b-instruct	61%	44%
mistral-large	58%	40%
llama-3-3-70b-instruct	35%	40%
granite-3-3-8b-instruct	32%	35%

These results indicate that enabling agent communication improves performance substantially for certain models (e.g., LLaMA-4 Maverick and LLaMA-3 405b), likely due to better multi-turn handling and the ability to clarify ambiguous information. For other models, performance is less sensitive to this parameter.

This experiment offers a compelling insight, highlighting the impact of hidden architectural features on benchmark results. Furthermore, it demonstrates that our benchmark can capture subtle differences in agent behavior and encourages transparent reporting of configuration parameters for reproducibility.