

# 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 ASSETOPSBENCH: BENCHMARKING AI AGENTS FOR TASK AUTOMATION IN INDUSTRIAL ASSET OPERA- TIONS AND MAINTENANCE

Anonymous authors

Paper under double-blind review

## ABSTRACT

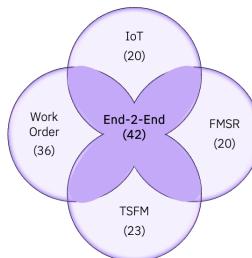
AI for Industrial Asset Lifecycle Management aims to automate complex operational workflows, including condition monitoring, maintenance planning, and intervention scheduling, thereby reducing human workload and minimizing system downtime. Traditional AI/ML approaches have primarily tackled these problems 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 (WikiSCADA) (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) (WikiALM). 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 1(a), 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 appropriate actions. 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 (WikiEAM), 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, maintenance report, work-order and business-level insights.

Despite recent advances in LLM-based agent frameworks, such as ReAct (Yao et al., 2023), HuggingGPT (Shen et al., 2023), Chameleon (Lu et al., 2023), and recent generalist agent models (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), TheAgentCompany (Xu et al., 2024b), CRM Arena-Pro (Huang et al., 2025)) are tailored toward machine learning, IT, customer-service domains, or purely digital, knowledge-work settings rather than physical, sensor-driven industrial operations. These benchmarks 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 (e.g., reliability engineers and data scientists).

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 1(b)), 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-Execute**, highlighting tradeoffs between interleaved decomposition or decomposition-first execution.
- 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.

Our motivation for a **multi-agent architecture** arises from industrial deployment experience, where heterogeneous workflows benefit from modular, task-specific agents (Figure 1(a)). We complement this practical insight with empirical evidence (Section 5.1) showing that multi-agent orchestration outperforms single-agent baselines on complex, composite tasks. For instance, 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,

108 ensuring agents not only retrieve correct answers but also follow reasoning patterns aligned with  
 109 domain expectations.  
 110

111 Finally, we experimented with an additional **closed-source 162 scenarios** to demonstrate generality,  
 112 spanning 10 asset classes, 53 failure modes, and 20 sensors. These include 42 live-deployment  
 113 scenarios (>90% correctness verified by a domain expert), 17 hydraulic system, 15 metro train, and  
 114 88 failure-mode scenarios encompassing diverse asset–failure–sensor relationships.  
 115

## 116 2 RELATED WORK 117

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

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

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

148 **Fine-Tuned and Compact Models.** Recent work has improved agent performance via fine-tuned  
 149 language models, often called *Large Action Models (LAMs)*, designed to execute structured actions  
 150 within environments. Systems such as TaskBench(Shen et al., 2024), xLAM(Zhang et al., 2025b),  
 151 AgentGen (Hu et al., 2025), AgentBank (Song et al., 2024), AgentRM (Xia et al., 2025), Fire-  
 152 Act (Chen et al., 2024), and ActionStudio (Zhang et al., 2025a) exemplify this trend. They are often  
 153 trained in grounded environments (e.g., Windows-based (Wang et al., 2025)) and evaluated on tasks  
 154 such as arithmetic, programming, and web interaction. While effective, these models remain lim-  
 155 ited to textual or web environments and have yet to demonstrate applicability to complex industrial  
 156 automation with hybrid agent compositions.

157 **Open Challenges.** Despite these advances, several gaps remain. First, there is a lack of comprehen-  
 158 sive benchmark datasets targeting industrial asset domains, particularly those involving condition-  
 159 based monitoring, predictive maintenance, automated diagnostics, and work order planning. To  
 160 support this claim, we analyzed a **catalog of 135 public datasets** (jonathanwvd, 2025) and found  
 161 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

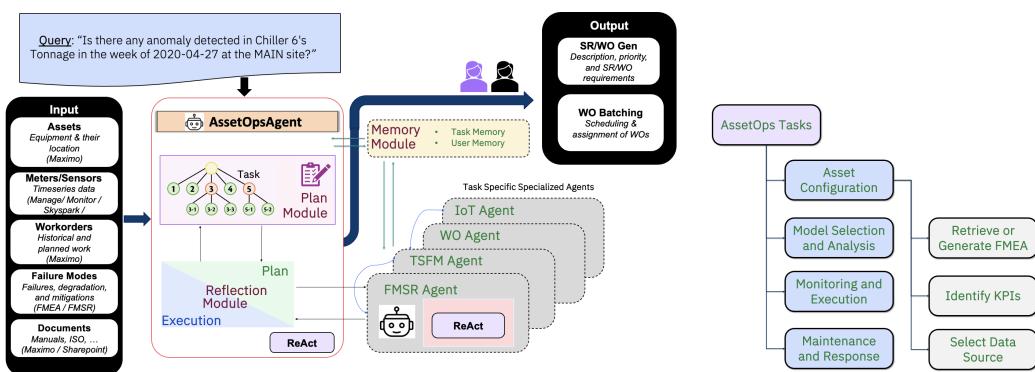
162 two modes, and none of the datasets support agentic applications. Second, time-series data, which  
 163 plays a central role in industrial and infrastructure-related applications, remains underrepresented  
 164 in existing agentic benchmarks. Finally, few systems support orchestration across heterogeneous  
 165 agents, including those based on text, code, or simulations, nor do they offer modular reasoning  
 166 strategies adaptable to complex, multi-agent workflows. Addressing these gaps is essential to ad-  
 167 vance general-purpose agent intelligence in high-stakes, real-world domains.

168

### 169 3 PROBLEM AND APPROACH: INTELLIGENT AGENT-BASED ASSET 170 OPERATIONS

171

172 Industrial asset operations involve complex, heterogeneous workflows where maintenance engi-  
 173 neers, reliability specialists, and facility planners must interpret multi-modal sensor data, detect  
 174 anomalies, and make timely operational decisions. Interdependent tasks such as root cause analysis,  
 175 predictive maintenance planning, work order bundling, and service request initiation often require  
 176 reasoning across historical telemetry, asset metadata, and operational constraints. Meeting these de-  
 177 mands requires intelligent agents that can decompose high-level requests into structured, executable  
 178 subtasks, coordinate across multiple domain-specific modules, and integrate outputs into actionable  
 179 recommendations. For example, a user might request: “Help configure an anomaly detection model  
 180 to monitor power consumption of CUXP and trigger alerts when usage is projected to exceed 8 Watts  
 181 above the maximum deviation observed over the past 30 days,” enabling timely corrective actions  
 182 such as service request creation. The diversity and interdependence of these tasks, spanning data  
 183 interpretation, anomaly reasoning, and operational decision-making, underscore the need for a co-  
 184 ordinated, intelligent agent framework capable of handling complex industrial workflows.



185

186 ((a)) Architecture of the Multi-Agent System: **Time Series (TSFM)** ((b)) Exemplar AssetOps Task Hier-  
 187 **Agent**, **Failure Mode Sensor Relations (FMSR) Agent**, **Work Order** archy  
 188 (WO) Agent

189

200 Figure 2(a) illustrates the core components of our proposed framework. At the center is the **Asset-  
 201 Ops Agent**, which functions as a global coordinator. It interprets high-level user queries in natural  
 202 language, decomposes them into structured subtasks, delegates these to specialized functional sub-  
 203 agents, and integrates their outputs into coherent responses, such as generating service requests or  
 204 work orders. To handle tasks like configuring anomaly detection models or triggering alerts for  
 205 assets such as CUXP, this coordination is essential. While typical multi-agent systems for general-  
 206 purpose tasks (e.g., Magentic (Fourney et al., 2024)) consist of an orchestrator or supervisor agent  
 207 coordinating sub-agents such as coders, file system handlers, terminals, or web-surfing agents, in  
 208 industrial settings these sub-agents are replaced by domain-inspired, task-specific agents. Examples  
 209 include an IoT agent, a failure mode to sensor mapping (FMSR) agent, a foundation model-driven  
 210 time series analyst (TSFM) agent, and a work order (WO) agent. These agents are specifically  
 211 tailored to monitor, analyze, and generate work orders or service requests for physical assets.

212 Building on this multi-agent architecture, defining a systematic benchmark requires determining the  
 213 set of tasks that accurately reflect real-world industrial operations. In this paper, we leveraged ISO  
 214 standards to construct a structured task taxonomy aligned with the stages of physical asset manage-  
 215 ment (ISO-2024, 2024; ISO, 2016). This taxonomy provides a consistent and scalable approach for

216 scenario generation, ensuring that each task maps to realistic operational objectives and decision-  
 217 making workflows. We refer to this methodology as **intent-driven** scenario generation, in contrast  
 218 to the **API-driven** scenario generation popularized in (Yao et al., 2024; Shen et al., 2024).

219 As illustrated in Figure 2(b), 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**  
 220 **Analysis**, where data scientists apply anomaly detection models and use LLM-powered retrieval to  
 221 surface relevant historical failures. In the **Monitoring and Execution** phase, operations teams manage  
 222 live telemetry, refine detection models, and enforce safety guardrails. Finally, the **Maintenance**  
 223 and **Response** phase focuses on actionable outputs, including generating work orders, summarizing  
 224 system health, and prioritizing interventions tasks typically handled by maintenance engineers.  
 225 Grounding task definitions and APIs in ISO standards allows the benchmark to generalize across  
 226 diverse industrial software platforms (Oracle, 2025; IBM).

## 229 4 ASSETOPS BENCH

230 **AssetOpsBench** consists of a real multi-asset, multi-source dataset from a data center, 141 man-  
 231 ually constructed task scenarios, and a benchmarking environment with task-specific AI agents and  
 232 an evaluation framework. The scenarios were developed over 18 months in collaboration with reli-  
 233 ability engineers, controls specialists, and domain experts overseeing assets such as AHUs, chillers,  
 234 boilers, and compressors. Experts identified key failure modes, drafted scenario templates capturing  
 235 realistic fault signatures and cross-sensor interactions, and iteratively refined them through multi-  
 236 ple review cycles to ensure plausibility and alignment with diagnostic reasoning. Each scenario  
 237 is grounded in operational and reference data, including sensor telemetry from industrial HVAC  
 238 systems (fifteen-minute intervals from BMS(WikiBMS) and SkySpark(SkyFoundry contributors)),  
 239 work orders from a product-level Maximo, and FMEA information from the Reliability Strategy  
 240 Library for data center operations. This combination ensures that the scenarios are both expert-  
 241 validated and data-driven, faithfully reflecting real-world industrial conditions.

### 244 4.1 MULTI-SOURCE DATASET

245 A key distinguishing feature of **AssetOpsBench** is its integration of richly structured, expert-curated  
 246 multi-source data that reflects the complexity of real-world industrial asset operations. Unlike a  
 247 simple data-gathering effort, constructing this benchmark required extensive data cleaning, the de-  
 248 velopment of a novel failure taxonomy, and careful alignment across heterogeneous sources.

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

250 <b>Data Source</b>	251 <b>Field</b>	252 <b>Description</b>
253 <b>Sensor Data*</b>	254 Chiller Return Temp.	255 Measures temperature of water returning to chiller
256 # Industrial Assets: 6	257 Chiller % Loaded	258 Indicates current load as a fraction of the maximum
259 Quantity: 2.3M points	260 Condenser Water Flow	261 Indicates the current flow rate through the condenser
262 <b>FMEA</b>	263 Failure Location / Comp.	264 Subsystem/part where failure occurs (e.g., bearings,)
265 # Industrial Assets: 3	266 Degradation Mechanism	267 Physical process driving failure (e.g., wear, erosion)
268 Quantity: 53 records	269 Degradation Influences	270 Stressors like runtime, fluid quality, or shock loading
271 <b>Work Orders</b>	272 ISO Failure Code	273 Standardized classification of the failure category.
274 # Ind. Assets: 10+	275 Event Log Timestamp	276 Time-marked entry recording an operational event
277 Quantity: 4.2K records	278 Linked Anomaly / Alert	279 References to alerts or anomalies tied to work order

280 As shown in Table 1, the benchmark includes over 2.3 million sensor data points across 6 assets (4  
 281 *Chillers* and 2 *AHUs*), capturing time-series signals such as *chiller return temperature*, *load percentage*,  
 282 and *condenser water flow*. The structured failure models, derived from Failure Mode Effects  
 283 Analysis (FMEA) records, encompass 53 failure entries across three equipment assets. FMEA pro-  
 284 vides provide detailed insights into the physical locations of failures, degradation mechanisms (such  
 285 as *wear* and *erosion*), and the influencing factors (including *runtime*, *fluid conditions*, and *shock loading*)  
 286 that contribute to each failure. Work order histories span 4.2K records across 10+ assets

270 and 11 years and incorporate ISO-standard failure codes, event timestamps, and linkages to alerts  
 271 and detected anomalies.

272 Additionally, the operational system generates a temporal sequence of alarm logs and also lever-  
 273 ages domain-specific technical rules obtained from experts, enabling contextual grounding of op-  
 274 erational anomalies. This diverse data foundation, comprising 9 modalities (Sensor, Work Order,  
 275 Alert, Alarm, FMEA, Anomaly, KPI 2 Failure Codes, Events, Rule 2 Failure Code), facilitates a  
 276 comprehensive evaluation of decision-making, tool usage, and multi-hop reasoning in industrial en-  
 277 vironments.

## 279 4.2 SCENARIO DESIGN AND COVERAGE

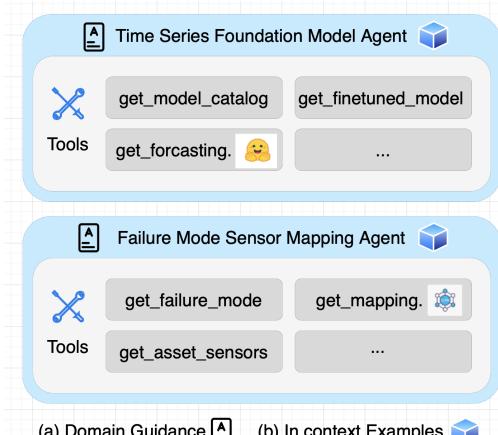
281 Each scenario in **AssetOpsBench** represents a structured operational query grounded in the  
 282 lifecycle-aligned task taxonomy (Figure 2(b)) and asset-specific datasets (Table 1). Each scenario is  
 283 formalized as:

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

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

294 **Scenario # 507**  
 295  
 296 **Text :** What is the predicted energy consumption for Chiller 9  
 297 in the week of 2020-04-27 based on data from the MAIN site?  
 298  
 299 **Type :** End-to-End: TSFM Agent, IoT Agent  
 300  
 301 **Category :** Analytical  
 302  
 303 **Characteristic Form :** The expected response should confirm the  
 304 successful execution of all required actions, ensuring that the  
 305 correct asset **Chiller 9**, location **MAIN**, and time range week of  
 306 2020-04-27 were used for data retrieval and analysis. It should  
 307 specify that the agent identified the sensor name **power input**  
 308 **sensor** and retrieved the historical energy consumption data for  
 309 Chiller 9 during the specified time period. The response must also  
 310 explain that the agent attempted to analyze the data for energy  
 311 consumption prediction, but was unable to do so due to  
 312 insufficient data, as the power input for Chiller 9 was consistently  
 313 0.0 from 2020-04-20 to 2020-04-25, indicating that the chiller was  
 314 not operating.

315 ((a)) Scenario example



316 (a) Domain Guidance

317 (b) In context Examples

318 Figure 3: Left: Scenario illustration. Right: Overview of two representative agent tools.

319 Figure 3(a) presents **Utterance 507**, an instructive case where a user requests a prediction of future  
 320 energy consumption. To address this query, the agent must first reason about which sensor variable  
 321 to use, specifically the power input, and after retrieving the data recognize that most values are zero,  
 322 indicating an insufficient data condition. This scenario highlights the importance of subject matter  
 323 experts (SMEs) in designing tasks that assess the reasoning capabilities of LLMs, rather than merely  
 324 testing tool functionality. In its characteristic form, we further emphasize key lexical markers such  
 325 as Chiller 9, MAIN, power input sensor, etc that also enable a semantic-based evaluation.

326 Our dataset also enables end users to design new scenarios, such as: “Examine whether the year-  
 327 over-year increase in corrective maintenance for CWC04009 warrants shifting resources from an-  
 328 nual repairs to multi-year replacement planning.” Existing scenarios (IDs 407–413) support strate-  
 329 gic work-order management tasks, including trend analysis, bundling, and probability forecasting.  
 330 Overall, the benchmark covers analytical reasoning (e.g., coding, model fine-tuning), context-aware  
 331 decision-making, and language-based generalization.

324  
325

## 4.3 DOMAIN SPECIFIC SINGLE AGENT AND MULTI-AGENT IMPLEMENTATION

326  
327  
328  
329  
330  
331  
332  
333

**AssetOpsBench** includes four domain-specific AI agents: IoT, TSFM, WO, and FMSR. To illustrate tool-level complexity, we highlight two representative agents (Figure 3(b)): the TSFM agent, which uses a pretrained time-series foundation model from Hugging Face, and the FMSR agent, which leverages an LLM to generate failure-mode-to-sensor mappings via the `get_mapping` function. In total, the platform comprises over 15 tools across these agents, each with domain-specific guidance, making them unique in industrial settings. Three agents (TSFM, IoT, FMSR) use ReAct Yao et al. (2023), while the WO agent uses CodeReAct (Wang et al., 2024); alternative strategies such as RAFA (Liu et al., 2023) are also supported.

334  
335  
336  
337  
338  
339  
340  
341  
342  
343

Given this mix of text- and code-based agents, a global coordinator, the **AssetOps Agent**, facilitates collaboration, operating under either an **Agent-As-Tool** paradigm or a **Plan-Execute** strategy. The components used to build these paradigm are widely adopted in modern open-source toolkits (Marreid et al., 2025; LangChain, 2025b; NVIDIA, 2025). In **Agent-As-Tool**, each agent is registered as a tool within a meta- or supervisor agent instantiated using ReAct, emulating layered decision-making in hierarchical organizations. In **Plan-Execute**, a **Planner** and **Reviewer** generate a plan as a directed acyclic graph (DAG), executed by an **Orchestrator** with a memory module that stores and transfers information between agents. This strategy adapts ReWoo (Xu et al., 2024a) with an additional review component inspired by (Li et al., 2025). We packaged the datasets, scenarios, domain-specific agents, and orchestration strategies into a dockerized environment.

344

345  
346

## 5 EXPERIMENTS AND LEADERBOARD

347  
348  
349  
350  
351

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).

352  
353  
354  
355  
356  
357  
358  
359  
360  
361

**LLM-As-Judge Scoring.** 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?).

362  
363  
364  
365  
366  
367  
368  
369  
370

**Reference-Based Scoring.** For each scenario, we construct a structured ground truth 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 `execution_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.

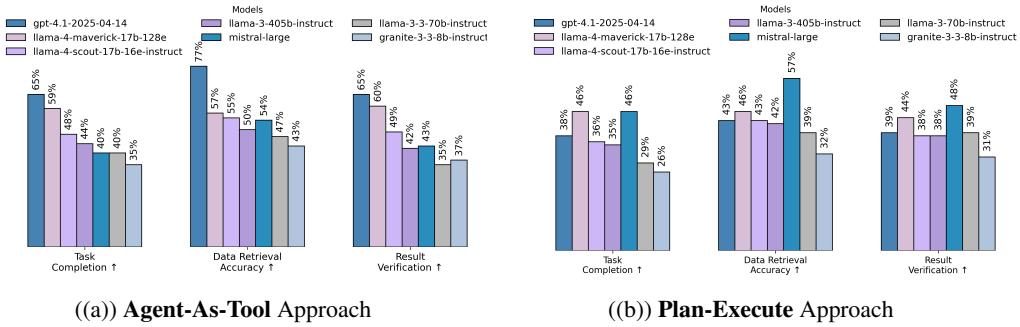
371  
372  
373  
374  
375  
376  
377

**Experimental Setting.** To quantify agent effectiveness in scenario evaluations, we adopt the **Pass<sup>k</sup>** metric. Unlike the widely used **Pass@k**, which measures the probability that at least one of  $k$  independent attempts succeeds, **Pass<sup>k</sup>** estimates the probability that an agent succeeds on *all*  $k$  attempts—a stricter criterion that better reflects the reliability requirements of industrial environments, where retries are often impractical and consistent behavior is essential for production deployment (LangChain, 2025a; Yao et al., 2024). In our benchmark, we report **Pass<sup>1</sup>** by default, as agents are executed once per task instance. The evaluation agent used for LLM-As-Judge scoring is run five times to derive stable performance estimates. Agents within the AgentOps framework

378 operate with a sampling temperature of 0, while the evaluation agent uses a temperature of 0.3, and  
 379 all reported results follow this configuration.  
 380

### 381 5.1 ASSETOPS BENCH LEADERBOARD

383 **Models.** We conducted a series of benchmark experiments to evaluate a diverse set of lan-  
 384 guage models, including closed-source models (e.g., `gpt-4.1`), frontier open-source models  
 385 (e.g., `llama-4-maverick`, `llama-4-scout`, `mistral-large`, `llama-3-405b`), and  
 386 medium-to-small open-source models (e.g., `llama-3-70b`, `granite-3-8b`). We have eval-  
 387 uated two different multi-agent strategies: *Agent-As-Tool* and *Plan-Execute* and also compared them  
 388 with single agent.  
 389



400 Figure 4: Approach-wise Performance Evaluation. The order is based on the task completion rate.  
 401

402 **Agent-As-Tool vs Plan-Execute Approach.** Figure 4 shows the combined performance of both ap-  
 403 proaches using the rubric method. Overall, the Agent-As-Tool approach, as illustrated in Figure 4(a),  
 404 demonstrates that `gpt-4.1` leads across nearly all metrics. `llama-4-maverick` also performs  
 405 competitively, particularly in result verification (60%) and clarity (78%). Also, Data retrieval ac-  
 406 curacy tend to higher than the task completion, yet another indirect validation of Evaluation Judge.  
 407 But wait, `gpt-4.1` did not maintain its leadership position in Plan-Execute Approach, infact it  
 408 see a largest drop in performance across all model. `mistral-large` and `llama-4-maverick`  
 409 are top pick models for Plan-Execute strategy. Given that `llama-4-maverick` demonstrates  
 410 balanced performance across both strategies, we select it as the default model for all ablation studies.  
 411

412 **Plan-Execute Approach Analysis.** We conducted a deep-dive analysis of the Plan-Execute ap-  
 413 proach to understand its relatively poor performance. First, we examined the length of the planning  
 414 steps and observed that larger models tend to generate shorter plans in the Plan-Execute approach  
 415 (typically 2–3 steps) compared to the Agent-As-Tool strategy, which generally requires 5–6 steps.  
 416 Given that Agent-As-Tool performs better and uses longer plans, this suggests a known limitation of  
 417 the Plan-Execute approach: reduced flexibility in handling unexpected failures or incorporating new  
 418 information that may require plan revision (Li et al., 2025; NVIDIA, 2025). Next, we obtained the  
 419 reference-based score of `gpt-4.1`, which is a `rougeL` of 0.354 and `rougeL` of 0.289 on the task  
 420 decomposition aspect. This score is substantially lower than the top-performing `mistral-large`  
 421 (`rougeL` 0.420, `rougeL` 0.343), indicating that, despite strong reasoning capabilities, `gpt-4.1`  
 422 generates outputs that are less lexically aligned with the reference ground truth trajectories. And  
 423 such behaviors may confuse down-stream agent in generating solution.  
 424

425 **Small Language Models Analysis.** Within the Agent-As-Tool evaluation, models such as  
 426 `granite-3-8b` and `llama-3-3-70b` show weaker overall performance, yet they reveal clear  
 427 areas of specialization as shown in Figure 5. Both models perform strongly on structured sens-  
 428 ing and diagnostic tasks: for example, `granite-3-3-8b-instruct` achieves 15/20 on IoT,  
 429 18/22 on FMSR, and 19/23 on TSFM, while `llama-3-3-70b-instruct` reaches 12/20, 18/22,  
 430 and 20/23 on the same categories. However, they struggle substantially on Work Order tasks, with  
 431 scores of only 2/36 and 7/36, indicating that procedural, multi-step coordination remains difficult  
 432 even under the Agent-As-Tool mechanism. This underscores a key insight: industrial deployments  
 433 may benefit most from **hybrid LLM-SLM agent architectures**, where strong specialists handle  
 434 sensing and diagnostics while more capable generalist models manage planning, coordination, and  
 435 end-to-end reasoning (Belcak et al., 2025).  
 436

432 **Human Validation.** To assess the reliability of  
 433 using LLMs as automatic evaluators for bench-  
 434 marking tasks, we compare model-generated  
 435 judgments against human annotations on a sam-  
 436 ple of 40 tasks. Each task is evaluated along  
 437 three dimensions by four domain experts, all  
 438 operating under the same information con-  
 439 straints as the LLMs. Before selecting a de-  
 440 fault evaluator, we compared several candidate  
 441 judge models, including *gpt-4.1*. In this  
 442 comparison, *gpt-4.1* showed only mod-  
 443 erate alignment with expert assessments, achiev-  
 444 ing 69% accuracy and Cohen’s  $\kappa$  of 0.44. In  
 445 contrast, *llama-4-maverick* provided sub-  
 446 stantially stronger agreement with human judg-  
 447 ments and was therefore selected as the default  
 448 judge model for the main analysis. Across ex-  
 449 perts, inter-rater reliability scores indicate sub-  
 450 stantial agreement on key evaluation dimen-  
 451 sions, with *Data Retrieval Accuracy* exhib-  
 452 iting the strongest consistency (Cohen’s  $\kappa = 0.79$ , 90.48% accuracy). *Task Completion* ( $\kappa = 0.62$ ) and  
 453 *Generalized Result Verification* ( $\kappa = 0.71$ ) also show high alignment among evaluators.

453 **Ablation Study.** We study the effect of adding additional distractor agents to the system and remov-  
 454 ing guidance (i.e., in-context examples). The ablation experiments are conducted using the **Agent-  
 455 As-Tool** method with *llama-4-maverick* as the default LLM. Injecting 10 out-of-domain dis-  
 456 tractors (e.g., SREAgent, EchoAgent) into 99 single-agent scenarios unexpectedly *improved* task  
 457 completion accuracy (from 44 to 46), suggesting that distractors may induce more deliberate rea-  
 458 soning in LLMs. Similar effects have been reported in prior parallel work (Fu-Hinthon, 2025).  
 459 Extending this experiment across our full model portfolio revealed consistent, though modest, gains  
 460 within the Llama family (particularly *llama-3-70b* and *llama-3-405b*), while other model  
 461 families showed slight performance reductions or no improvement. In contrast, removing all in-  
 462 context examples for 65 single-agent tasks (IoT+FMSR+TSFM) caused performance to collapse  
 463 (from 80% to 34% for *gpt-4.1* and from 60% to 3% for *granite-3-8b*).

463 **Baseline using Single-Agent.** Instead of using four domain-specific sub-agents and an orchestra-  
 464 tion agent, we build a tool-calling ReAct agent with a single prompt as a baseline, giving it access  
 465 to tools and in-context examples from all agents. In doing so, we increase the complexity of the  
 466 problem, as it must handle many tools as well as an expanded context. We run a default LLM,  
 467 *llama-4-maverick*, on all 141 scenarios. As a single-agent baseline, it achieves task comple-  
 468 tion of 26.95%, data retrieval accuracy of 34.04%, and generalized result verification of 28.37%.  
 469 Under the Agent-As-Tool setup, the same model achieves roughly two-fold improvements (See Fig-  
 470 ure 4(a)).

471

## 472 5.2 ERROR ANALYSIS VIA AGENT TRAJECTORIES AND EMERGING FAILURE MODES

473 Trajectory analysis is critical for detecting agent mistakes, but becomes more challenging in multi-  
 474 agent settings. We collected approximately 881 trajectories across different runs of models for  
 475 Agent-As-Tool strategy. These trajectories were leveraged for further error analysis on two aspects:  
 476 (a) tool-related errors and (b) agent failure modes.

477 **Failure Analysis on Tool Use.** Each agent step in a trajectory is logged as a structured JSON  
 478 record capturing the *action type* and *execution state*. At the sub-agent level, we distinguish be-  
 479 tween **Tool-oriented actions**, which invoke predefined functions with well-defined inputs and out-  
 480 puts, and **CodeReAct-oriented actions**, where agents dynamically generate and execute Python  
 481 code. Our analysis shows that Tool-oriented actions achieve higher valid-execution rates, whereas  
 482 CodeReAct-oriented actions incur more runtime failures due to the variability of the generated  
 483 code. Tool-oriented failures are concentrated in a small number of tools, including *jsonreader*,  
 484 *tsfm\_integrated\_tsad*, and *Read Sensors From File*, highlighting challenges related  
 485 to input validation and hallucinated parameter passing.

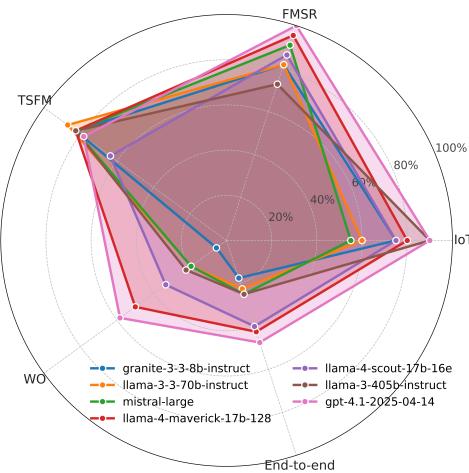


Figure 5: Agent Level Task Accomplishment with respect to Agent-As-Tool Approach

486 **Emerging Failure Modes Discovery.** Now we investigate  
 487 trajectories from a semantic perspective. Recent work (Cemri  
 488 et al., 2025) defines 14 failure modes for agent trajectories.  
 489 Table 2 shows the distribution of failure mode on our 881  
 490 trajectories across this taxonomy. We found that **system de-  
 491 sign** is the most common source of failures. This taxonomy  
 492 provides guidance for improving agent development. For  
 493 instance, since the “Fail to Ask for Clarification” mode occurs  
 494 around 10% of the time, we introduced a feature in the Agent-  
 495 As-Tool strategy that allows sub-agents to ask the parent agent  
 496 questions at any point during execution. We reran the entire  
 497 benchmark on the default LLM, and this change led to signif-  
 498 icant performance improvements for `llama-4-maverick`,  
 499 increasing task completion from 59% to 66%, surpassing  
 500 `gpt-4.1`. To capture failure mode behaviors beyond this  
 501 taxonomy, we allowed self-discovery of up to two **novel failure  
 502 modes** per trace, revealing *emergent and compound failures*  
 503 not covered by existing classifications. Common emerg-  
 504 ent failures include **Overstatement of Task Completion**  
 505 (122 cases, 23.8%), **Extraneous or Ambiguous Output For-  
 506 matting** (110 cases, 21.4%), and **Ineffective Error Recovery**  
 507 (160 cases).

### 5.3 GENERALIZATION ACROSS INDUSTRIAL DOMAINS

508 With the help of experts and the product team, we prepared an additional 162 scenarios across four  
 509 datasets to evaluate generalization: Metro Train MetroPT-3 (15 scenarios) for compressor faults,  
 510 UCI Hydraulic System (17 scenarios) for hydraulic component faults, Asset Health internal dataset  
 511 (42 scenarios) based on work orders, and FailureSensorQA (88 scenarios) using ISO-standardized  
 512 documentation for sensor-to-failure mapping. Table 3 presents one representative scenario from  
 513 each dataset along with the performance `llama-4-maverick`. Among all the datasets, scenarios  
 514 of MetroPT-3 are difficult as we observed poor performance (task completion rate = 26.7%).

515 Table 3: Representative scenario from each dataset with LLaMA-4 Maverick performance.  
 516

517 <b>Dataset</b>	518 <b>Representative Scenario with LLaMA-4 Maverick Performance</b>
519 MetroPT-3	520 Consider asset <code>mp_1</code> . After maintenance on May 30, 2020, how has the compres- 521 sor’s condition evolved from May 31 to June 6, and are further repairs or monitoring 522 needed? <b>Performance:</b> Task Completion 26.7%, Data Retrieval Accuracy 20.0%, General- 523 ized Verification 40.0%
524 Hydraulic System	525 For asset <code>hp_1</code> , can severe internal pump leakage on 2024-01-31 be detected using 526 sensor data from the preceding 100 days? <b>Performance:</b> Task Completion 88.2%, Data Retrieval Accuracy 100.0%, Gener- 527 alized Verification 88.2%
528 Asset Health	529 Analyze the provided <code>Air Handling Unit_615152AC</code> work orders and asset 530 details to determine the expected system condition. <b>Performance:</b> Task Completion 100.0%, Data Retrieval Accuracy 100.0%, Gener- 531 alized Verification 100.0%
532 FailureSensorQA	533 For an aero gas turbine, list all failure modes that can be detected or indicated by 534 abnormal readings from vibration, speed, or fuel flow sensors. <b>Performance:</b> Task Completion 67.0%, Data Retrieval Accuracy 71.6%, Gener- 535 alized Verification 56.8%

## 536 6 CONCLUSION

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

Table 2: Distribution of Failure Sub-categories Across Stages of Execution

Failure Subcategory	Stage & %
<b>System Design (Total 37.38%)</b>	
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%
<b>Agent Coordination (Total 27.52%)</b>	
Conversation Reset	Execution: 0.00%
Fail to Ask for Clarification	Execution: <b>10.22%</b>
Task Derailment	Execution: 4.34%
Information Withholding	Execution: 2.22%
Ignored Agent’s Input	Execution: 2.06%
Action Mismatch	Execution: 8.68%
<b>Task Verification (Total 35.10%)</b>	
Premature Termination	Pre: 3.92%
No or Incomplete Veri.	Execution: 15.56%
Incorrect Verification	Execution: 15.62%

540 REFERENCES  
541

542 Pierre Andrews, Amine Benhalloum, Gerard Moreno-Torres Bertran, Matteo Bettini, Amar Budhi-  
543 raja, Ricardo Silveira Cabral, Virginie Do, Romain Froger, Emilien Garreau, Jean-Baptiste Gaya,  
544 Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Grégoire Mi-  
545 alon, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Thomas Scialom,  
546 Vladislav Vorotilov, Mengjue Wang, and Ian Yu. Are: Scaling up agent environments and evalua-  
547 tions, 2025. URL <https://arxiv.org/abs/2509.17158>.

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

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

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

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

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

563 Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang, Zhu,  
564 Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang,  
565 Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and  
566 Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks,  
567 2024. URL <https://arxiv.org/abs/2411.04468>.

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

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

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

581 Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar  
582 Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. Crmarena-pro:  
583 Holistic assessment of llm agents across diverse business scenarios and interactions. *arXiv*  
584 *preprint arXiv:2505.18878*, 2025.

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

594 Phan Nht Huy, Tien N Nguyen, and Nghi D. Q. Bui. Hyperagent: Generalist software engineering  
 595 agents to solve coding tasks at scale, 2025. URL <https://openreview.net/forum?id=PZf4RSPMBG>.  
 596

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

599

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

602

603 ISO-2024. Iso 55000:2024 asset management — vocabulary, overview and principles. *ISO*, 2024.  
 604 URL <https://www.iso.org/standard/55000.html>. Defines asset management life-  
 605 cycle and taxonomy.

606

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

616

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

620

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

624

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

626

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

628

629 Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning  
 630 in multi-agent systems. In *The Thirteenth International Conference on Learning Representations*,  
 631 2025. URL <https://openreview.net/forum?id=EqcLAU6gyU>.

632

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

636

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

640

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

643

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

647

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

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

656 NVIDIA. About nvidia agent intelligence toolkit workflows, version 1.1.0. <https://docs.nvidia.com/aiqtoolkit/1.1.0/workflows/about/index.html>, 2025. Accessed: 2025-11-23.  
 657

660 Oracle. Add failure diagnostics information to asset incidents and anomalies.  
 661 <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.  
 663

664 Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize  
 665 Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chat-  
 666 Dev: Communicative agents for software development. In Lun-Wei Ku, Andre Martins, and  
 667 Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Com-  
 668 putational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, August  
 669 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL  
 670 <https://aclanthology.org/2024.acl-long.810/>.  
 671

672 Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hug-  
 673 ginggapt: Solving ai tasks with chatgpt and its friends in hugging face, 2023. URL <https://arxiv.org/abs/2303.17580>.  
 674

675 Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng  
 676 Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation.  
 677 In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang  
 678 (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 4540–4574. Curran  
 679 Associates, Inc., 2024. URL [https://proceedings.neurips.cc/paper\\_files/paper/2024/file/085185ea97db31ae6dcac7497616fd3e-Paper-Datasets\\_and\\_Benchmarks\\_Track.pdf](https://proceedings.neurips.cc/paper_files/paper/2024/file/085185ea97db31ae6dcac7497616fd3e-Paper-Datasets_and_Benchmarks_Track.pdf).  
 681

682 SkyFoundry contributors. Skyspark (software). <https://skyfoundry.com/>. Proprietary  
 683 software; not a Wikipedia page.  
 684

685 Yifan Song, Weimin Xiong, Xutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng,  
 686 and Sujian Li. Agentbank: Towards generalized llm agents via fine-tuning on 50000+ interaction  
 687 trajectories, 2024. URL <https://arxiv.org/abs/2410.07706>.  
 688

689 Andrew Szot, Bogdan Mazoure, Omar Attia, Aleksei Timofeev, Harsh Agrawal, Devon Hjelm, Zhe  
 690 Gan, Zsolt Kira, and Alexander Toshev. From multimodal llms to generalist embodied agents:  
 691 Methods and lessons, 2024. URL <https://arxiv.org/abs/2412.08442>.  
 692

693 LangChain Team. Benchmarking single agent performance. LangChain Blog, February 2025.  
 694 URL <https://blog.langchain.com/react-agent-benchmarking/>. Accessed:  
 695 YYYY-MM-DD.  
 696

697 Lu Wang, Fangkai Yang, Chaoyun Zhang, Junting Lu, Jiaxu Qian, Shilin He, Pu Zhao, Bo Qiao, Ray  
 698 Huang, Si Qin, Qisheng Su, Jiayi Ye, Yudi Zhang, Jian-Guang Lou, Qingwei Lin, Saravan Raj-  
 699 mohan, Dongmei Zhang, and Qi Zhang. Large action models: From inception to implementation,  
 700 2025. URL <https://arxiv.org/abs/2412.10047>.  
 701

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

702 Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin  
 703 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,  
 704 L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural*  
 705 *Information Processing Systems*, volume 37, pp. 137610–137645. Curran Associates, Inc., 2024.  
 706 URL [https://proceedings.neurips.cc/paper\\_files/paper/2024/file/f8c24b08b96a08ec7a7a975feeaa777e-Paper-Datasets\\_and\\_Benchmarks\\_Track.pdf](https://proceedings.neurips.cc/paper_files/paper/2024/file/f8c24b08b96a08ec7a7a975feeaa777e-Paper-Datasets_and_Benchmarks_Track.pdf).

710 WikiALM. Asset lifecycle management. [https://en.wikipedia.org/wiki/Enterprise\\_asset\\_management](https://en.wikipedia.org/wiki/Enterprise_asset_management). Accessed: 2025-11-25.

711 WikiBMS. Building management system. [https://en.wikipedia.org/wiki/Energy\\_management\\_system\\_\(building\\_management\)](https://en.wikipedia.org/wiki/Energy_management_system_(building_management)). Accessed: 2025-11-25.

712 WikiEAM. Enterprise asset management. [https://en.wikipedia.org/wiki/Enterprise\\_asset\\_management](https://en.wikipedia.org/wiki/Enterprise_asset_management). Accessed: 2025-11-25.

713 WikiSCADA. Scada. <https://en.wikipedia.org/wiki/SCADA>. Accessed: 2025-11-25.

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

717 Binfeng Xu, Zhiyuan PENG, Bowen Lei, Subhabrata Mukherjee, and Dongkuan Xu. DE-  
 718 COUPLING REASONING FROM OBSERVATIONS FOR EFFICIENT AUGMENTED LAN-  
 719 GUAGE MODELS, 2024a. URL <https://openreview.net/forum?id=Cpgo06j6W1>.

720 Frank F Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z Wang,  
 721 Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on  
 722 consequential real world tasks. *arXiv preprint arXiv:2412.14161*, 2024b.

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

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

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

731 Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao,  
 732 Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Manoj Awalgaoonkar,  
 733 Rithesh R N, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese,  
 734 and Caiming Xiong. xLAM: A family of large action models to empower  
 735 AI agent systems. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025*  
*736 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,  
 737 New Mexico, April 2025b. Association for Computational Linguistics. ISBN 979-8-89176-189-6.  
 738 URL <https://aclanthology.org/2025.naacl-long.578/>.

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

## APPENDIX OVERVIEW

In the Appendix, we discuss several topics that complement the main paper and provide additional technical detail to ensure clarity and reproducibility. These sections elaborate on our agentic system formulation, dataset design choices, hierarchical structuring of AssetOpsBench, and further empirical analyses.

- Agentic System Definition
- AssetOpsBench: Environment, Hierarchy and Domain Specific Agents
- Datasets Utilized in AssetOpsBench
- AssetOpsBench Scenarios
- Ground Truth Preparation for Reference-based Evaluation
- Additional Benchmark Experiments
- Generality: New Datasets and Scenarios
- Emerging Failure Mode Discovery and Agent Development

## AGENTIC SYSTEM DEFINITION

This section provides a **generic** 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. We also discussed detailed design of two approaches for multi-agent system development: “Agent-As-Tool” and “Plan-Execute”.

## 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:

- $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$  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.
- $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_k\}$  is the set of tasks. Each task  $\tau$  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).
- $\Pi$  is the hierarchical plan space. A plan  $\pi \in \Pi$  is an ordered sequence of task-agent assignments:

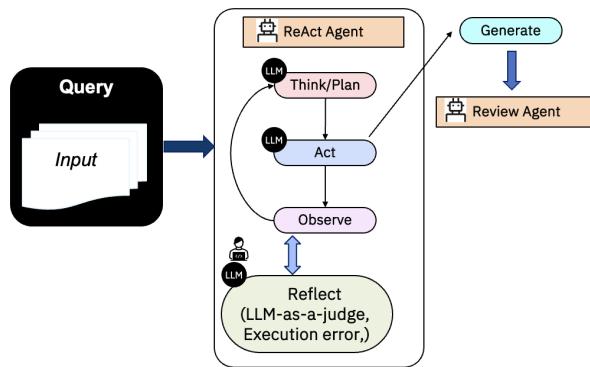
$$\pi = [\langle \tau_1, A_i \rangle, \langle \tau_2, A_j \rangle, \dots]$$

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.

810 A.2 BASE AGENT: REACT  
811

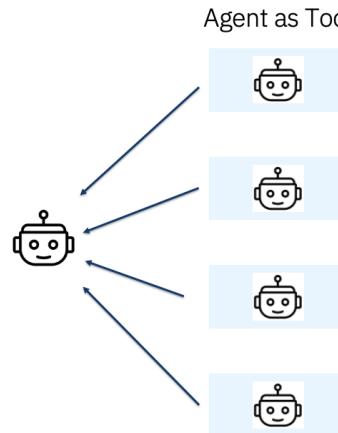
812 AssetOpsBench uses the ReAct framework Yao et al. (2023) in an end-to-end agent design that  
813 integrates a Review Agent to verify the final answer. Figure 6 illustrates the full architecture. The  
814 ReAct agent executes a **Think-Act-Observe** loop, solving tasks iteratively while detecting and re-  
815 covering from repetitive or ineffective actions. The Review agent verifies whether the ReAct agent  
816 has successfully completed the task, ensuring the quality of the output. Subsequent sections present  
817 the architecture in detail, highlighting the distinction between two architectural paradigms:**Agent-  
818 As-Tool** (See Section A.3) and **Plan-Execute** (See Section A.4). Note that, we can replace ReAct by  
819 any other agent development methodology such as Reflect, RAFA, etc.

834 Figure 6: ReAct used to build individual agent  
835  
836837 A.3 AGENT-AS-TOOL  
838

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

840

- 841 • A standard ReAct (Think–Act–Observe) agent loop using open source framework. In the  
842 initial setup, the *number of reflections* was set to one—effectively disabling reflection.
- 843 • A curated list of tools, the majority of which are stub interfaces that delegate functionality to  
844 specialized sub-agents. The only standalone utility tool in this set was the `JSONReader`,  
845 which reads a JSON object from a file and returns its contents as the tool’s direct response.

861 Figure 7: Agent-As-Tool  
862  
863

864 The sub-agent stubs were intentionally designed to be minimal. Each stub accepted a single input  
 865 parameter: a string called `request` and returned a structured JSON output. The output JSON  
 866 object included the following fields:  
 867

- 868 • `answer` – the primary answer returned by the sub-agent, represented as a plain string.
- 869 • `review` – a nested JSON object capturing a review of the response, typically including  
 870 fields such as `status`, `reasoning`, and `suggestions`.
- 871 • `summary` – a brief description of the JSON object’s structure and semantics, useful for  
 872 interpretability or chaining with downstream tools.  
 873

874 The ReAct agent was initialized with a standard prompt that includes:  
 875

- 876 • **Examples for In-Context Learning** – A small number of sample interactions for each sub-  
 877 agent were provided to guide behavior. These examples followed the standard ReAct for-  
 878 mat of Think–Act–Observe, illustrating how to invoke tools and interpret their responses.  
 879 A representative example is shown below:  
 880
- 881 • **Tool Demonstrations** – These sample calls were concatenated to form a comprehensive  
 882 set of demonstrations for all tools available to the agent, effectively seeding it with usage  
 883 patterns.  
 884

884 The sample calls for all the tools are concatenated to form the examples.  
 885

- 886 • `question` - the question input to ReAct
- 887 • `tool names` - the list of sub-agent tool names (plus `JSONReader`)
- 888 • `tool descriptions` - descriptions of the sub-agents  
 889

```
890
891 Question: download asset history for CU02004 at SiteX
892 from 2016-07-14T20:30:00-04:00 to 2016-07-14T23:30:00-04:00
893 for CHILLED WATER LEAVING TEMP and
894 CHILLED WATER RETURN TEMP
895
896 Action 1: IoTAgent
897 Action Input 1: request=download asset history for CU02004
898 at SiteX from 2016-07-14T20:30:00-04:00 to
899 2016-07-14T23:30:00-04:00 for CHILLED WATER LEAVING TEMP
  and CHILLED WATER RETURN TEMP
900
901 Observation 1: {
902   site_name: SiteX,
903   assetnum: CU02004,
904   total_observations: 25,
905   start: 2025-03-26T00:00:00.000000+00:00,
906   final: 2025-04-02T00:00:00.000000+00:00,
907   file_path: /var/folders/fz/.../cbmdir/c328516a-643f-40e6-8701-
908   ↪ e875b1985c38.json,
909   message: found 25 observations. file_path contains a JSON array of
   ↪ Observation data
}
```

910 Listing 1: Example of Trajectory using ReAct Agent for IoTAgent  
 911  
 912

913 **Execution Framework.** The ReAct engine is reinitialized for each question and executed until  
 914 either (a) successful completion, as determined by the Review component using an LLM-as-  
 915 judge or (b) a maximum of ten iterations. The framework iterates through a list of models (e.g.,  
 916 `mistralai/mistral-large`) and a corresponding list of utterances to execute for each model.  
 917 The system supports retries for failed executions. After each ReAct run, the complete trajectory and  
 918 associated evaluation metrics are stored. We have provided a sample (partial) trajectory trace in

918 Listing 1, which show how patent agent call one of the tool (in this case IoTAgent) and receive a  
 919 response. The recorded metrics include:  
 920

- 921 • **Question:** the input query being processed
- 922 • **Total execution time:** duration of the entire ReAct loop
- 923 • **Number of ReAct steps:** count of action-observation cycles
- 924 • **Review status:** success or failure determined by the LLM-based reviewer
- 925
- 926
- 927

928 Listing 2 outlines how the FMSR agent packages its reasoning output into a structured message  
 929 for downstream agents or evaluators. The custom.json function formats the response to include the  
 930 final answer, a peer review section (comprising status, reasoning, and suggestions), and a reflection  
 931 field. Additionally, a natural language message is synthesized to summarize the execution result,  
 932 enhancing transparency and interpretability in multi-agent settings. This output acts as a compact  
 933 yet comprehensive communication protocol for reasoning agents collaborating in a complex task  
 934 pipeline.

```
935
936 def custom_json(obj):
937     if isinstance(obj, FMSRResponse):
938         return {
939             answer : obj.answer,
940             review : {
941                 status : obj.review[ status ],
942                 reasoning : obj.review[ reasoning ],
943                 suggestions : obj.review[ suggestions ],
944             },
945             reflection : obj.reflection,
946             message : (
947                 I_am_FMSR_Agent,_and_I_have_completed_my_task._  

948                 f The_status_of_my_execution_is_{obj.review['status']}.  

949                 →_  

950                 f I_also_received_a_review_from_the_reflection_agent;_  

951                 f suggestions_are_included_in_the_review_field_for_  

952                 → further_insights.
953             ),
954         }
955     raise TypeError(f Cannot_serialize_object_of_type_{type(obj)} )
```

956 Listing 2: Formatted response message from FMSRAgent

957 **A.4 PLAN-EXECUTE**

958 **Plan-Execute.** *Plan-Execute* is a widely used architectural paradigm for multi-agent systems. Figure 8 depicts the implementation adopted in our work. It is derived from specialized multi-agent system Marreed et al. (2025). 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.

959 This architecture supports modularity, robustness, and interpretability across the task lifecycle. We  
 960 have provided two system prompts where first prompt guides an AI to generate a structured step-by-  
 961 step plan using external agents, while the second prompt instructs a reviewer agent to evaluate the  
 962 plan’s correctness and completeness in JSON format.

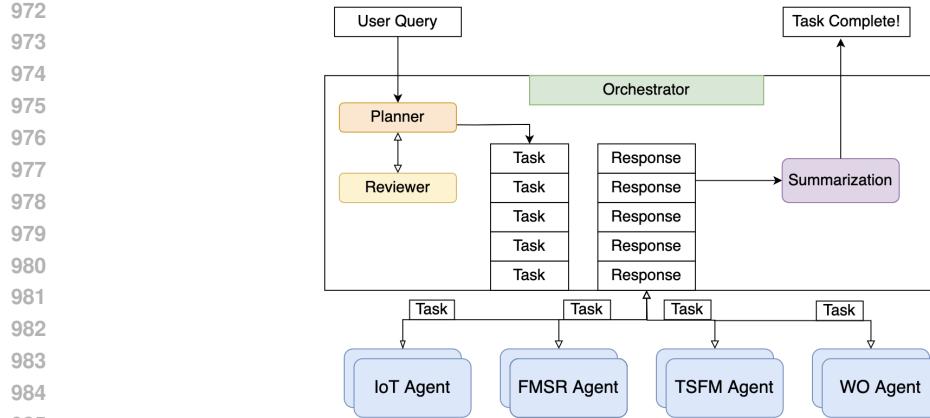


Figure 8: Plan-Execute Multi-Agent System

## System Prompt (Planning Agent)

```

You are an AI assistant who makes step-by-step plan to solve a
↳ complicated problem under the help of external agents.
For each step, make one task followed by one agent-call.
Each step denoted by #S1, #S2, #S3 ... can be referred to in later
↳ steps as a dependency.

Each step must contain Task, Agent, Dependency and ExpectedOutput.
1. **Task**: A detailed description of what needs to be done in
↳ this step. It should include all necessary details and
↳ requirements.
2. **Agent**: The external agent to be used for solving this task.
↳ Agent needs to be selected from the available agents.
3. **Dependency**: A list of previous steps (denoted as `#S1`,
↳ `#S2`, etc.) that this step depends on. If no previous steps are
↳ required, use `None`.
4. **ExpectedOutput**: The anticipated result from the agent's
↳ execution.

## Output Format (Replace '<...>') ##

## Step 1
#Task1: <describe your task here>
#Agent1: <agent_name>
#Dependency1: None
#ExpectedOutput1: <describe the expected output of the call>

## Step 2
#Task2: <describe next task>
#Agent2: <agent_name>
#Dependency2: [<you can use #S1 and more to represent previous
↳ outputs as a dependency>]
#ExpectedOutput2: <describe the expected output of the call>

And so on...

Here are the available agents:
{agent_descriptions}

You are going to solve the following complicated problem:
{task.description}

Guidelines:

```

1026  
 1027     - Task should be something that can be solved by the agent.  
 1028     - A plan usually contains less than 5 steps.  
 1029     - Only output the generated plan.

1030     Output (your generated plan):  
 1031

1032

### 1033     System Prompt (Review Agent)

1034

```
1035 review_plan_system_prompt_template = """You are a critical reviewer
1036   ↳ tasked with evaluating the effectiveness and accuracy of a
1037   ↳ plan. Your goal is to determine whether the plan is valid or
1038   ↳ not given the context of the input question and agent
1039   ↳ expertise. A valid plan should:
1040
1041 1. **Ensure all necessary actions are addressed:**  

1042   The plan must cover all required steps to successfully complete  

1043   ↳ the task as specified in the question. Ensure that each  

1044   ↳ action directly contributes to the task goal.
1045 2. **Include appropriate dependencies between steps:**  

1046   Actions should be logically ordered with clear dependencies.  

1047   ↳ Each step must rely on the completion of the previous step  

1048   ↳ to ensure a coherent and efficient workflow.
1049 3. **Ensure no crucial steps are missed:**  

1050   The plan must not overlook any essential actions required to  

1051   ↳ solve the task. If any crucial steps are absent, the plan  

1052   ↳ must be flagged as incomplete.
1053 4. **Confirm all actions align with agent capabilities:**  

1054   Each step in the plan must fall within the designated expertise  

1055   ↳ of the agents involved. No action should require expertise  

1056   ↳ or knowledge outside of the agent's specified capabilities.  

1057   ↳ Any plan that violate this condition is an invalid plan.
1058 5. **Strictly follow the task's question:**  

1059   Carefully compare the provided question with the task. The plan  

1060   ↳ should only include actions that directly relate to the  

1061   ↳ question's explicit requirements, without introducing any  

1062   ↳ unnecessary tasks or assumptions.
1063 6. **Avoid Abstract task/step:**  

1064   Ensure steps/tasks are grounded with respect to the data  

1065   ↳ generated by previous steps or the question.
1066
1067 #### Evaluation Criteria:
1068 1. **Completeness:**  

1069   - Verify that the system prompt leads to a plan that includes  

1070   ↳ all necessary steps to accomplish the task.
1071   - Ensure the description of each step contains all the relevant  

1072   ↳ information needed to execute the step, including any  

1073   ↳ required parameters or inputs that are mentioned in the  

1074   ↳ task's question.
1075 2. **Relevance:**  

1076   - Confirm that each step in the plan directly contributes to  

1077   ↳ solving the task.
1078   - Eliminate any steps that do not serve a clear purpose in  

1079   ↳ achieving the goal.
1080
1081 3. **Correctness:**  

1082   - Ensure that all steps are logically consistent and ordered  

1083   ↳ correctly.
1084   - Ensure that the dependencies between the steps are valid and  

1085   ↳ follow a correct sequence.
1086
1087 4. **Expertise Alignment:**
```

```

1080
1081      - Confirm that the steps in the plan are within the capabilities
1082      ↳ of the agent.
1083      - Validate that the agents used in each steps are among the
1084      ↳ available agents mentioned in the agents' expertise.
1085
1086      5. Efficiency:
1087          - Make sure the plan doesn't introduce redundant actions.
1088          - Avoid unnecessary complexity in the plan.
1089
1090      6. Clarity:
1091          - Ensure that the plan is easy to understand and logically
1092          ↳ structured.
1093
1094      ---
1095
1096      Question:
1097      {question}
1098
1099      Agents' Expertise:
1100      {agent_expertise}
1101
1102      Plan:
1103      {plan}
1104
1105      ---
1106
1107      Output Format:
1108      Your review must always be in JSON format. Do not include any
1109      ↳ additional formatting or Markdown in your response.
1110
1111      ``json
1112      {{{
1113          "status": "Valid | Invalid | Other",
1114          "reasoning": "A concise explanation for your evaluation. If a
1115          ↳ specific step is wrong, point it out directly.",
1116          "suggestions": "Actions or improvements for rectifying the plan
1117          ↳ if applicable."
1118      }}}
1119
1120
1121      Output:
1122      """
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

```

## 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 IoTAgent
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.

```
1134 Action 3: Finish
1135 Action Input 3: The anomaly detection results of 'Chiller 6 Return Temperature' using data in /var/folders/fz/1lh7gpv96rv5lg6m_d6bk0gc000gn/T/cbmdir/2e2eea99-946c-4a30-a688-8ddf479ab62.json are stored in file ./output/tsad_output//tsad_conformal.csv Final Answer: The anomaly detection results of 'Chiller 6 Return Temperature' using data in /var/folders/fz/1lh7gpv96rv5lg6m_d6bk0gc000gn/T/cbmdir/2e2eea99-946c-4a30-a688-8ddf479ab62.json are stored in file ./output/tsad_output//tsad_conformal.csv Question
1137 Process is completed now
1138 Task Execution Status (Finished): True
1139 Review Agent Feedback: {'status': 'Accomplished', 'reasoning': 'The agent successfully executed the task by performing time series anomaly detection on \'Chiller 6 Return Temperature\' using the data from the specified file. The agent used the tsfm_integrated_tsad tool with the correct dataset path, timestamp column, and target columns. The agent 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.'}
1140 run minutes = 2.1432575666666667
1141
```

Figure 10: The Final step of the execution

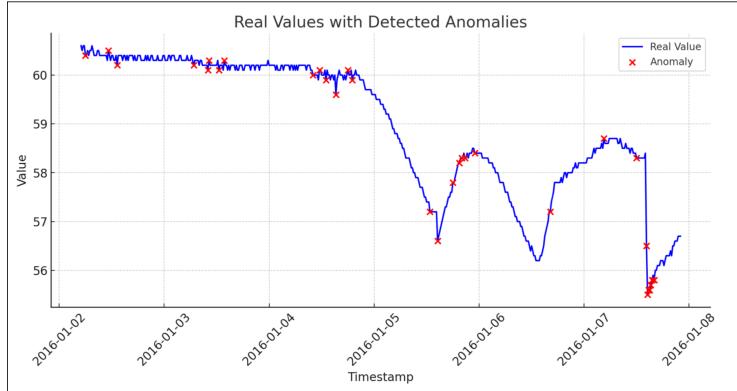


Figure 11: Anomaly Detection : Final Output

## B ASSETOPSBENCH: ENVIRONMENT, HIERARCHY AND DOMAIN SPECIFIC AGENTS

This section presents the simulated environment for agentic evaluation, structured task taxonomy used in AssetOpsBench, which organizes benchmark scenarios based on key stages in the industrial asset lifecycle.

## B.1 SIMULATED ENVIRONMENT

Figure 12 provides a simulated docker environment for executing the task. The environment consists of domain specific agents (FMSR, IoT, TSFM, WO), and model inference APIs (LLM and TSFM), and access to telemetry data and Industry 4.0 data such as FMEA, Work Order, Alert and etc. The system also comes with implementation of orchestration such as Agent-As-Tool and Plan-Execute that are interfacing with user query.

First, we discuss the taxonomy that is used 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.3 presents the rationale for FMSR Agent, emphasizing its role in bridging raw telemetry with diagnostic reasoning through sensor–failure mapping. Appendix B.5 focuses on the WO Agent, which operationalizes maintenance 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.

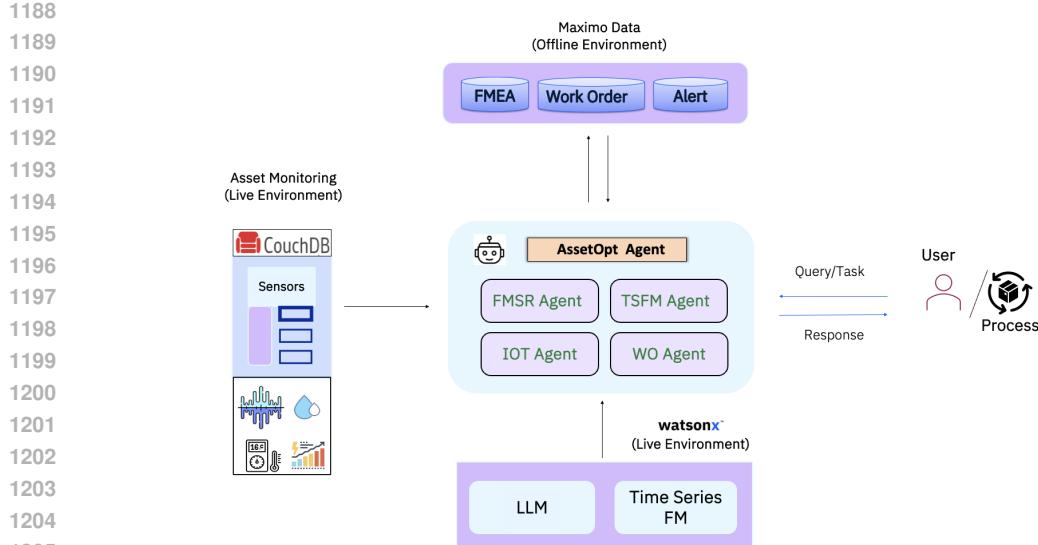


Figure 12: Simulated Environment for Open Source Contribution and Testing



Figure 13: Representative Routine tasks in Asset Lifecycle Management.

## B.2 RATIONALE FOR IoT AGENT OVER APPLICATION

1239  
 1240  
 1241

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.3 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:

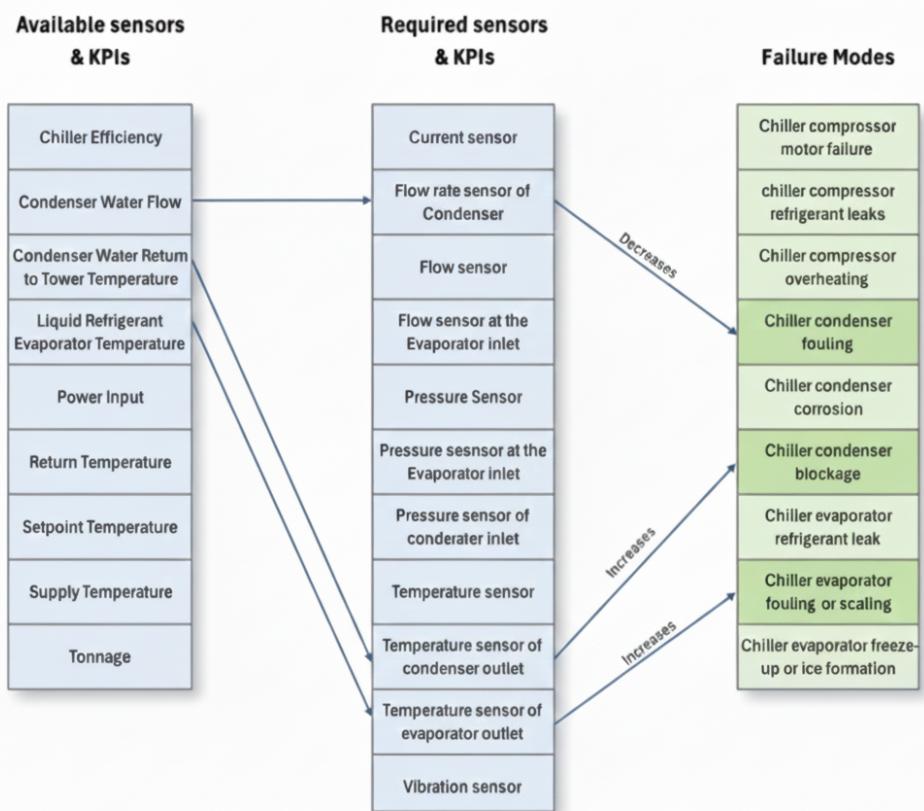


Figure 14: Mapping Example internally used by FMSR Agent

1296        1. **Bridging Raw Data and Diagnostic Insight:** The table explicitly maps sensor variables to  
 1297        relevant failure modes, establishing a direct link between low-level telemetry and high-level  
 1298        maintenance reasoning. This supports tasks such as fault detection, root cause analysis, and  
 1299        feature selection for learning-based systems.  
 1300        2. **Alignment with FMEA Methodology:** By structuring failure explanations according to  
 1301        the principles of Failure Modes and Effects Analysis (FMEA), the table offers a formalized,  
 1302        interpretable view of asset health. Each sensor's diagnostic role is contextualized through  
 1303        failure causes, effects, and detection implications.  
 1304        3. **Supporting Explainability and Safety:** In industrial environments, operational decisions  
 1305        require transparency. The alignment table enhances system explainability by clarifying  
 1306        why a given signal is relevant, how it relates to equipment health, and what operational  
 1307        risks it may indicate.  
 1308        4. **Improving Dataset Transparency:** The AssetOpsBench dataset includes a wide range  
 1309        of sensors across multiple devices. This table functions as a documentation layer that  
 1310        improves usability, reproducibility, and understanding for researchers and practitioners en-  
 1311        gaging with the benchmark.  
 1312        5. **Guiding Model and Rule Development:** Whether designing rule-based systems, hybrid  
 1313        AI architectures, or physics-informed machine learning models, a well-defined mapping  
 1314        of sensors to failure mechanisms is foundational. It informs the construction of robust  
 1315        detection logic and contributes to generalizable reasoning strategies.  
 1316

1317        In sum, the sensor–failure alignment table plays a central role in transforming raw operational  
 1318        telemetry into structured, actionable insight. It provides the semantic grounding necessary for de-  
 1319        veloping interpretable, reliable, and effective AI agents for real-world industrial maintenance tasks.  
 1320        Table 4 provides an extensive example for sensor–failure mode relation for a chiller system build  
 1321        using our SME inputs.

1322        Table 4: Sensor Interpretation and Failure Mode Relevance in Chiller Systems - Illustrative  
 1323

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

	<b>Sensor</b>	<b>Explanation</b>	<b>Impact on Chiller Health / Failure Mode Relevance</b>
1350	Chilled Water Flow	Water flow through evaporator	May point to <i>pump failure, valve issues, or airlocks</i> .
1351	Motor Run Status	Compressor motor operational state	Discrepancies could signal <i>false starts, sensor error, or runtime misreporting</i> .
1352	Vibration Point #1 SP	Vibration sensor setpoint (location #1)	May indicate <i>bearing failure, imbalance, or mechanical looseness</i> .
1353	CHW Valve Position	Position of chilled water valve	Out-of-range position may imply <i>valve actuator fault or control misbehavior</i> .
1354	CHW Differential Pressure (D/P)	Pressure drop across chilled water loop	Suggests <i>clogging, filter fouling, or flow resistance</i> .
1355	CHW Flow Hi Alarm SP	Alarm setpoint for high CHW flow	Triggered by <i>pump overspeed, valve overshoot, or control issues</i> .
1356	Condenser Return Temp	Water temperature returning to the condenser	Important for <i>thermal load calculation and monitoring efficiency</i> .
1357	Average Amps	Average motor current	High current may indicate <i>overload, bearing drag, or electrical faults</i> .
1358	CHW Valve Close Control	Control signal to close CHW valve	Improper function may cause <i>flow issues or unmet loads</i> .
1359	CNW Differential Pressure (D/P)	Pressure drop in condenser loop	Indicates <i>scaling, fouling, or pump degradation</i> .
1360	VFD Internal Ambient Temp	Internal temperature of VFD	High temps may trigger <i>thermal trips or shorten VFD lifespan</i> .
1361	Freon Temp	Refrigerant temperature	Abnormal values may suggest <i>charge issues, expansion valve faults, or heat exchange failure</i> .
1362	Compressor Oil Sump Temp	Oil sump temperature	High temperature may signal <i>bearing wear or insufficient cooling</i> .
1363	Chilled Water Return Temp	Return water temp to evaporator	Used for <i>cooling load and delta-T analysis</i> .
1364	Motor Run Status RPT	Reported motor run confirmation	Mismatch suggests <i>sensor/control error</i> .
1365	VFD Inverter Link Current	Current through VFD inverter link	High current may indicate <i>overload or VFD stress</i> .
1366	CHWSTSP in Part Mode	Setpoint in partial load mode	Improper configuration can cause <i>energy waste or load mismatch</i> .
1367	VFD Phase A/B/C Current	Phase currents from VFD	Used to detect <i>imbalances, shorts, or phase loss</i> .
1368	VFD Converter Heat Sink Temp	VFD heat sink temperature	Elevated temps reduce <i>component life and can cause failure</i> .
1369	Compressor Oil Pressure	Oil pressure in compressor	Low pressure risks <i>lubrication failure and component damage</i> .
1370	Failure (status flag)	Direct failure indicator	Used as ground truth label for fault evaluation.
1371	VFD Setpoint	Speed or torque command	Affects <i>energy usage, response time, and cooling capacity</i> .
1372	CHW Flow High Alarm	High flow warning flag	May indicate <i>system control faults or oversized flow components</i> .
1373	VFD DC Bus Voltage	DC voltage level inside VFD	Instability can reflect <i>power quality issues</i> .
1374	CNW Flow High Alarm	High condenser water flow warning	May reflect <i>valve misposition or energy inefficiency</i> .
1375	CNW Flow Low Alarm SP	Low flow alarm threshold	Indicates <i>risk of overheating or shutdown due to poor heat rejection</i> .
1376	Warning Code	Non-critical warning status	Helpful for <i>early diagnostics or trend detection</i> .
1377			
1378			
1379			
1380			
1381			
1382			
1383			
1384			
1385			
1386			
1387			
1388			
1389			
1390			
1391			
1392			
1393			
1394			
1395			
1396			
1397			
1398			
1399			
1400			
1401			
1402			
1403			

1404	Sensor	Explanation	Impact on Chiller Health / Failure Mode
1405			Relevance
1406			
1407	Vibration Points #2/#3 SP	Additional vibration set-points	Detect <i>imbalance, wear, or mechanical degradation.</i>
1408			

#### 1409 1410 B.4 RATIONALE FOR TSFM AGENT OVER APPLICATION 1411

1412 The TSFM Agent is purpose-built to support critical tasks within the **AssetOps** workflow, as outlined  
1413 in Figure 2(a). Within **Model Selection and Analysis**, TSFM Agent enables forecasting of key per-  
1414 formance indicators (KPIs) using lightweight, pre-trained foundation models. Its adaptive anomaly  
1415 detection framework, based on post-hoc conformal prediction, supports calibrated and interpretable  
1416 anomaly scores, providing high utility for both **Monitoring and Execution** and **Maintenance and**  
1417 **Response**.

1418 Specifically, the TSFM Agent can execute and refine models, classify anomalies based on historical  
1419 deviations, and support operational guardrails by simulating expected trends under normal condi-  
1420 tions. In downstream applications, the agent’s outputs can be used to summarize overall system  
1421 health by tracking the frequency of anomalies across selected KPIs. These anomalies serve as a  
1422 foundation for maintenance recommendations, enabling preventive and reactive work order genera-  
1423 tion. TSFM Agent facilitates real-time, data-driven decision-making throughout the asset lifecycle.

#### 1424 1425 B.5 RATIONALE FOR WO AGENT OVER APPLICATION 1426

1427 The WO Agent, a code based ReAct, in **AssetOpsBench** is designed to enable intelligent interaction  
1428 with structured and unstructured maintenance records through a modular data model. It operates  
1429 over a set of *Business Objects* (BOs) that represent work orders, alerts, anomalies, failure codes, and  
1430 asset metadata. These BOs are categorized into five functional groups that collectively support the  
1431 WO Agent’s decision-making capabilities.

1432 To reason over these BOs, the WO Agent is equipped with a collection of analytic functions that  
1433 allow it to retrieve, interpret, and act upon historical and real-time data. The agent’s capabilities are  
1434 structured as follows:

- 1435 **1. Historical Reasoning via Content Objects and Knowledge Extraction:** The WO Agent  
1436 accesses raw maintenance data such as *WorkOrders*, *Events*, including Work orders, alerts,  
1437 and anomaly Events. Knowledge extraction functions enable the agent to retrieve and filter  
1438 this data by date, asset, and work order type, allowing targeted analysis and retrospective  
1439 diagnostics.
- 1440 **2. Standardized Interpretation with Meta/Profile Objects:** BOs like *ISO Failure Code*,  
1441 *AlertRule*, and *Equipment* provide structured classification schemes. These allow the agent  
1442 to categorize failures, apply semantic filters, and maintain compatibility with domain con-  
1443 ventions—critical for aligning alerts and anomalies with actionable categories.
- 1444 **3. Temporal and Causal Reasoning via Statistical Functions:** Leveraging relationship BOs  
1445 such as *Alert-Rule Mapping* and *Anomaly Mapping*, the WO Agent applies statistical func-  
1446 tions (e.g., Allen’s Interval Algebra) to detect temporal patterns—such as when alerts con-  
1447 sistently precede failures. It also detects repeated work order cycles, helping align main-  
1448 tenance with actual degradation patterns instead of fixed schedules.
- 1449 **4. Predictive and Prescriptive Intelligence through Decision Support Functions:** Using  
1450 the *WorkOrderRecommendation* BO, the agent forecasts future work orders, recommends  
1451 maintenance based on alerts or KPI anomalies, and identifies opportunities for bundling  
1452 related tasks. These decision support functions enable proactive scheduling and optimize  
1453 resource use across the asset lifecycle.
- 1454 **5. Persona-Aligned Interaction and Query Resolution:** The WO Agent interfaces naturally  
1455 with domain personas. Maintenance engineers can explore past interventions for a given  
1456 failure, while planners can query upcoming work order demands or seek opportunities to  
1457 consolidate tasks. These capabilities are backed by modular functions that support flexible  
1458 querying and planning logic.

1458  
 1459 In summary, the WO Agent is a hybrid reasoning and decision-support agent built atop structured  
 1460 business objects and analytic functions. It connects historical insight with predictive planning, en-  
 1461 abling lifecycle-aware maintenance interventions grounded in transparent, data-driven logic.  
 1462

1463 **B.6 TOOLS USED BY AGENTS**

1464  
 1465 In this section, we describe the development of over fifteen LangChain-based tools that form the  
 1466 backbone of our agent framework. We follow a standardized methodology for tool construction,  
 1467 and, with the exception of the WO agent, all agents operate through tool-calling APIs. Table 5 lists  
 1468 thirteen of these tools along with their names, descriptions, and parameters. For brevity, we omit  
 1469 some of the lower-level parameters associated with the time-series tool suite. In case of WO agent,  
 1470 which is a coding agent, we needed to build a generic business driven object, as given in Table 6.  
 1471

1472 Table 5: List of Available Tools and Their Parameters.  
 1473

1474 1475 <b>Tool Name</b>	1476 <b>Description</b>	1477 <b>Parameters (Required Fields)</b>
1476 <b>Get Failure Modes</b>	1477 Retrieves failure modes linked to a <code>asset_name</code> : name of the asset. specific asset.	
1478 <b>Get Failure Mode and Sensor Rele- vancy Mapping</b>	1479 Returns relevancy mapping between <code>input_str</code> : string with asset name, failure modes and sensors for down- stream tasks.	1480 failure modes, and sensors.
1481 <b>Read Sensors From File</b>	1482 Reads available sensors of an asset <code>input_str</code> : sensor file path. from a file and outputs sensor vari- able names.	
1483 <b>sites</b>	1484 Retrieves a list of available sites.	1485 <code>v__args</code> : optional array ( <i>default: null</i> ).
1486 <b>history</b>	1487 Returns sensor values for an asset <code>site_name</code> , <code>assetnum</code> , <code>start</code> , within a given time range. <code>final</code> .	
1488 <b>assets</b>	1489 Lists all assets available at a given <code>site_name</code> . site.	
1490 <b>sensors</b>	1491 Lists all sensors for an asset at a <code>site_name</code> , <code>assetnum</code> . given site.	
1492 <b>jsonreader</b>	1493 Parses a JSON file and returns its <code>file_name</code> . content.	
1494 <b>currentdatetime</b>	1495 Returns current date and time as <code>v__args</code> : optional array ( <i>default: JSON</i> ). null).	
1496 <b>aitasks</b>	1497 Lists available AI tasks and <code>v__args</code> : optional array ( <i>default: their methods (task_id, null)</i> ). <code>description</code> ).	
1498 <b>tsfmmodels</b>	1499 Lists supported forecasting models <code>v__args</code> : optional array ( <i>default: (ID, checkpoint, description)</i> ).	
1500 <b>tsfm_forecasting</b>	1501 Forecasts sensor or KPI variables us- ing pretrained time-series models.	1502 <code>dataset_path</code> , <code>model_checkpoint</code> , <code>timestamp_column</code> , <code>target_columns</code> .
1503 <b>tsfm_forecasting</b> <b>finetune</b>	1504 Finetunes a pretrained forecasting model on new data.	1505 <code>dataset_path</code> , <code>model_checkpoint</code> , <code>timestamp_column</code> , <code>target_columns</code> .
1506 <b>tsfm_integrated</b> <b>tsad</b>	1507 Performs time-series anomaly detec- tion using model predictions.	1508 <code>dataset_path</code> , <code>timestamp_column</code> , <code>target_columns</code> .

1512 Table 6: WO Agent Summary of Business Objects, Source, Role, and Number of Records  
1513

1514 <b>Business Object</b>	1515 <b>Source</b>	1516 <b>Role</b>	1517 <b>Count</b>
<b>Content Objects</b>			
1518 WorkOrder	1519 Work Order Manager	1520 Tracks scheduled and unscheduled maintenance tasks, categorized as preventive or corrective.	1521 4392
1522 Event	1523 Aggregated by Authors	1524 Consolidates event logs for tracking and decision-making.	1525 6929
1526 Alert Events	1527 IoT Repository	1528 Logs real-time alerts triggered by IoT sensors based on predefined conditions.	1529 1995
1530 Anomaly Events	1531 ML Generated	1532 Detects KPI deviations using machine learning for predictive maintenance.	1533 542
<b>Meta/Profile Objects</b>			
1534 ISO Failure Code	1535 Developed by Authors	1536 Standardizes failure classification for structured maintenance analysis.	1537 137
1538 ISO Primary Failure Code	1539 Developed by Authors	1540 Defines primary failure categories and links related secondary codes.	1541 68
1542 AlertRule	1543 SME Provided	1544 Specifies conditions for triggering alerts based on system behaviors.	1545 77
1546 Equipment	1547 SME Provided	1548 Represents industrial assets, including status and specifications.	1549 22
<b>Relationship Causality Objects</b>			
1549 Alert-Rule Mapping	1550 Relationship Causality	1551 Links alert rules to failure codes for automated diagnostics.	1552 46
1553 Anomaly Mapping	1554 Relationship Causality	1555 Associates anomalies with failure codes for predictive insights.	1556 12
<b>Recommendation Objects</b>			
1556 WorkOrder Recommendation	1557 Recommendation	1558 Suggests maintenance actions based on historical patterns.	1559 N/A

1546 *Note:* The design and structure of the business objects and corresponding analysis in this section are valid for  
1547 other industrial asset types, such as standby generators.

## 1548 C DATASETS UTILIZED IN ASSETOPSBENCH

1551 In this part, as extension of Section 4.1, we will zoom into the datasets utilized by the various agents  
1552 of **AssetOpsBench** (More details of the roles of the agents in the asset lifetime management can be  
1553 found at Appendix B).

### 1554 C.1 SENSOR TELEMETRY DATASET FOR IOT AGENT AND TSFM AGENT

1557 Both IoT Agent and TSFM Agent (Figure 2(a)) leverage the **Sensor Telemetry Dataset**, which  
1558 comprises sensor telemetry collected from Building Management Systems (BMS) and the SkySpark  
1559 analytics platform. This dataset captures fifteen-minute interval operational data from industrial  
1560 HVAC systems, specifically a fleet of chillers. Each chiller unit (e.g., Chiller 4, Chiller 14) is in-  
1561 strumented with a standardized suite of physical sensors that monitor key operational parameters in  
1562 real-time.

1563 A representative subset of these sensors is summarized in Table 7. These sensors record various  
1564 kinematic, dynamic, thermodynamic, electrical, and operational metrics essential to assessing the  
1565 performance and health of chiller systems. Measurements include water and refrigerant tempera-  
1566 tures, power consumption, cooling capacity (tonnage), flow rates, and system setpoints. Addition-

ally, computed metrics such as chiller efficiency and load percentage serve as valuable real-time indicators of system performance.

Table 7: 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 of water supplied to the condenser
Chiller Temperature	
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, enabling 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.

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 that are 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.

## C.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

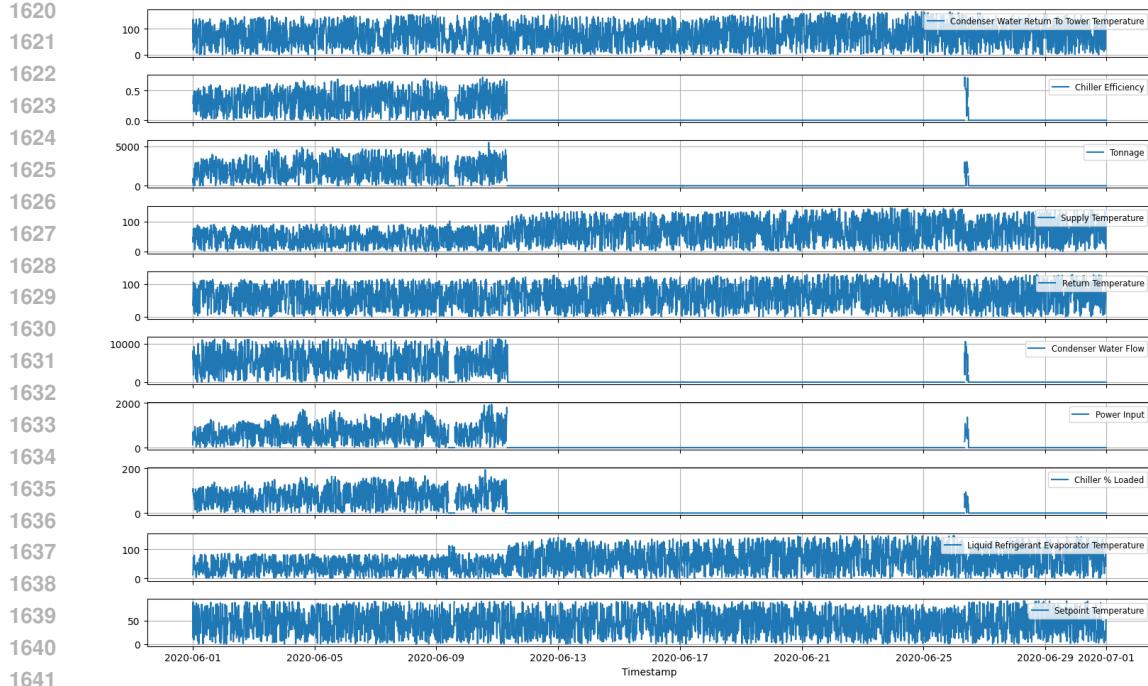


Figure 15: Snapshot of time series data from Chiller 6 for June 2020. Each subplot shows an individual sensor’s trend over time.

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.

1674 Each maintenance task in the dataset is mapped to its detection mechanism and action type (e.g.,  
 1675 condition monitoring vs. corrective repair), along with documentation on task content and recom-  
 1676 mended frequency. These structured records not only support early fault detection and diagnostics  
 1677 but also facilitate benchmarking of intelligent agents' reasoning over real-world degradation patterns  
 1678 and maintenance decisions. Failures are temporally aligned with telemetry, enabling the study of  
 1679 degradation trajectories and pre-failure conditions. This integrated design makes the dataset suitable  
 1680 for supervised learning, causal inference, and evaluation of digital twins or predictive maintenance  
 1681 agents under realistic operating uncertainty.

1682 To utilize the failure modes and their association with the sensors, we design FMSR (Failure Mode  
 1683 Sensor Relations) to interpret failure mode datasets within the **AssetOpsBench** framework, leverag-  
 1684 ing structured FMEA (Failure Modes and Effects Analysis) principles to link sensor telemetry with  
 1685 degradation mechanisms and operational failures. Using annotated failure records for assets such as  
 1686 centrifugal chillers, the FMSR Agent builds knowledge graphs and reasoning models that connect  
 1687 specific failure modes like compressor overheating, evaporator fouling, or refrigerant valve failure to  
 1688 their underlying causes and detectable symptoms. These failure modes are mapped to available sen-  
 1689 sor measurements (e.g., supply temperature, power input, vibration, flow rate) to identify observable  
 1690 precursors. For example, compressor overheating may be monitored through trends in power input,  
 1691 chiller efficiency, and evaporator temperature, while condenser fouling can manifest in abnormal  
 1692 return temperatures and flow rate deviations. Through structured utterances, users can query which  
 1693 failure modes are associated with specific sensors, which are critical for detecting a given failure, or  
 1694 even construct machine learning recipes for predictive modeling such as anomaly models for chiller  
 1695 trips or excessive purging. The agent leverages this data to perform rule based diagnostics, support  
 1696 causal analysis, and assist in condition based maintenance planning. By aligning temporal sensor  
 1697 patterns with known failure signatures, the FMSR Agent enables explainable fault detection and  
 1698 root cause inference, ultimately enhancing reliability, maintainability, and transparency in HVAC  
 1699 operations.

### 1700 C.3 WORK ORDER DATASETS FOR WO AGENT

1701 Table 6 provide the summary of datasets (as business objects) and the size for each dataset. Those  
 1702 work order datasets in **AssetOpsBench** provide a structured view of maintenance activity across in-  
 1703 dustrial assets, encompassing both preventive and corrective interventions using work orders. Each  
 1704 work order is associated with rich contextual data including equipment metadata, failure classi-  
 1705 fication codes (e.g., ISO Failure Code, ISO Primary Failure Code), event logs, sensor-triggered  
 1706 alerts, and machine-generated anomalies. These records are linked temporally and causally, allow-  
 1707 ing agents to reason about asset history, detect recurring failure patterns, and recommend actions  
 1708 based on past interventions.

1709 The group of datasets distinguishes between core content objects (e.g., WorkOrders, Alerts, Events,  
 1710 Anomalies), metadata profiles, and relational structures that map alerts and anomalies to failure  
 1711 codes.

1712 The individual event tables: *work orders* (Table 8), *alert events* (Table 9), and *anomaly events* (Ta-  
 1713 ble 10) captures different but complementary signals related to equipment condition and behavior.  
 1714 To enable integrated analysis and causal reasoning, these events are unified into a common *event ta-  
 1715 ble schema* (Table 12), allowing temporal alignment and cross-type relationship discovery between  
 1716 maintenance actions, system warnings, and performance anomalies.

1717 In addition, to support the linkage of failure code over the events, we provide two mapping tables:  
 1718 one that connects alert rules to likely failure codes, and another that maps KPI-based anomalies to  
 1719 structured failure categories (Tables of 13 and 11). These mappings enable agents to infer probable  
 1720 root causes from real-time signals and integrate data-driven insights with expert failure taxonomies.

1721 This help us to develop WO agent to support grounded evaluation of diagnostic reasoning, task  
 1722 generation, and repair recommendation. More particularly, the WO agent analyze historical work  
 1723 orders to identify repeated maintenance issues and improve task scheduling. It processed historical  
 1724 work order, alerts (from IoT Agent) and anomalies (from TSFM agent) event, linking them to failure  
 1725 codes to support predictive maintenance recommendations. In the potential industrial applications,  
 1726 WO agent can complete to tasks of automating the interpretation of maintenance data, predicting  
 1727 future work orders, and bundling related tasks to reduce operational downtime.

1728

1729

1730

Table 8: Work Order Event Schema Definition

1731

1732

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"

1762

1763

1764

1765

1766

1767

Table 9: Alert Event Schema Definition

1768

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"

1770

1771

1772

1773

1774

1775

1776

1777

1778

1779

1780

1781

Table 10: Anomaly Event Schema Definition

Field Name	Type	Description
timestamp	DateTime	The date and time when the anomaly event was recorded. Example: "4/26/20 14:14"
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 11: 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.

Table 12: 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 13: 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 Mal-function	Human-readable explanation of the failure code.

1890 **D ASSETOPSBENCH SCENARIOS**  
18911892 **D.1 SCENARIOS CREATION PRINCIPLES**  
18931894 The scenarios in **AssetOpsBench** are designed to evaluate the capabilities required for autonomous  
1895 agents operating in real industrial environments. Although grounded in real operational data and  
1896 engineering practices, each scenario is intentionally framed to test a specific dimension of agent  
1897 reasoning, tool interaction, and decision-making relevant to asset management. The scenarios are  
1898 built around four core principles:1899 

- 1900 • **Reasoning and Tool Use:** Scenarios require agents to perform domain-specific reasoning  
1901 such as time-based logic, schema interpretation, and multi-step tool invocation. Common  
1902 failure cases include premature termination, incorrect parameter selection, or misuse of  
1903 diagnostic tools.
- 1904 • **Data Handling and Forecasting:** Agents must interpret telemetry, detect anomalies, and  
1905 configure appropriate models for forecasting or anomaly detection. Tasks emphasize the  
1906 translation of real-world engineering intuition into ML configuration steps (e.g., model  
1907 selection, training windows, thresholds).
- 1908 • **Agent Communication and Coordination:** Many scenarios simulate multi-agent work-  
1909 flows where the agent must ask clarifying questions, summarize findings, or coordinate  
1910 subtasks. This reflects how real engineering teams collaborate during diagnostics or plan-  
1911 ning.
- 1912 • **Workflow Orchestration and Decision-Making:** Scenarios measure the agent’s ability  
1913 to plan complex workflows, handle dependencies, reason under uncertainty, and determine  
1914 when to stop or escalate due to missing or conflicting information.

1915 These principles ensure that scenarios remain faithful to real asset-management workflows while  
1916 systematically probing the capabilities of autonomous agents.  
19171918 **D.2 SCENARIO GENERATION**  
19191920 The scenarios in **AssetOpsBench** were generated from real industrial operations and shaped through  
1921 an 18-month collaboration with reliability engineers, controls specialists, and domain experts re-  
1922 sponsible for large portfolios of mechanical assets (e.g., AHUs, chillers, boilers, compressors). Un-  
1923 like synthetic rule-based benchmarks, the scenarios are grounded in operational conditions, OEM  
1924 specifications, maintenance records, and engineering workflows used in practice.1925 The development process was iterative and domain-driven. Subject-matter experts first identified  
1926 high-impact failure modes and diagnostic tasks central to asset health, safety, and performance. For  
1927 each asset type, expert engineers drafted scenario templates that captured realistic fault signatures,  
1928 cross-sensor interactions, physical constraints, and contextual operating conditions. These templates  
1929 underwent multi-round reviews involving 3–7 experts to ensure that each scenario reflects plausible  
1930 field behavior and aligns with real diagnostic reasoning patterns.1931 Across the 18-month timeline, the scenario library evolved through 12 major iterations. Each it-  
1932 eration added new scenario types, refined diagnostic narratives, and updated failure descriptions  
1933 based on expert insights and validation against real data patterns. While generating a single sce-  
1934 nario is relatively quick, ensuring its realism, consistency, and clarity required significant expert  
1935 effort including documentation, cross-checking with historical data, and verification of operational  
1936 plausibility.1937 Overall, the **AssetOpsBench** scenarios form a rigorously curated, expert-validated collection of op-  
1938 erational situations that reflect how reliability engineers analyze equipment behavior in real-world  
1939 industrial settings.1940 **D.3 SCENARIO STATISTICS**  
19411942 As shown in Table 14, **AssetOpsBench** includes a total of 141 scenarios with 99 single-agent sce-  
1943 narios and 42 multi-agent scenarios. These scenarios are to be open source research community.

1944 Table 14: Examples of Scenario with their Subtypes (Aligned with Task Taxonomy - Figure 2(b))  
1945

1946 <b>Agent Group</b>	1947 <b>Subtype</b>	1948 <b>Task Descriptions</b>
1949 <b>TSFM Agent</b> 1950 # Scenarios: 23	1948 Forecasting	Predict future KPI trends over time windows
	1949 Model Tuning	Select or refine time series models for accuracy
	1950 Anomaly Detection	Identify deviations in operational behavior
	1951 Hybrid Tasks	Combine prediction with anomaly evaluation
	1952 Model Capabilities	Query TSFM model limits and configurations
1953 <b>Work Order Agent</b> 1954 # Scenarios: 36	1953 Retrieval & Filter	Filter work orders by asset, location, or time
	1954 Event Summary	Summarize logs or alerts over time windows
	1955 Scheduling	Recommend or optimize work order sequences
	1956 RCA & Alert Review	Perform root cause or alert logic review
	1957 KPI-based Reco.	Link alerts or KPI trends to work orders
1958 <b>Multi-Agent (End-to-End) Tasks</b> 1959 # Scenarios: 42	1958 Knowledge Query	Tasks involving anomaly detection or forecasting
	1959 Failure Reasoning	Uses degradation models and causal logic
	1960 Sensor Mapping	Maps failure modes to sensors
	1961 Sensor Inventory	Retrieves installed sensors on an asset
	1962 Other	Multi-step inference or decision-making

1963 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 14 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.

1976 

#### D.4 SCENARIO EXAMPLES

1977 We include two examples (Table 15 and Table 16) that showcase distinct behaviors of agent outputs. Readers can observe that the *characteristic form* varies even for problems that appear similar on the surface.

1981 Table 15: Example Knowledge Query: Energy Prediction for Chiller 9  
1982

1983 <b>Field</b>	1984 <b>Description</b>
1984 <b>ID</b>	1985 507
1985 <b>Type</b>	1986 Knowledge Query
1986 <b>Text</b>	1987 What is the predicted energy consumption for Chiller 9 in the week of 2020-04-27 based on data from the MAIN site?
1988 <b>Characteristic Form</b>	1989 The expected response should confirm the successful execution of all required actions, ensuring that the correct asset ( <b>Chiller 9</b> ), location ( <b>MAIN</b> ), and time range ( <b>week of 2020-04-27</b> ) were used for data retrieval and analysis. It should specify that the agent identified the sensor name ( <i>power input sensor</i> ) and retrieved the historical energy consumption data for Chiller 9 during the specified time period. 1990 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 16: Example Knowledge Query: Predicting Energy Usage for Chiller 9

Field	Description
<b>ID</b>	511
<b>Type</b>	Knowledge Query
<b>Text</b>	Can you predict Chiller 9’s energy usage for next week based on data from the week of 2020-04-27 at <b>MAIN</b> ?
<b>Characteristic Form</b>	<p>The expected response should confirm the successful execution of all required actions, ensuring that the correct asset (<b>Chiller 9</b>) and location (<b>MAIN site</b>) were used for data retrieval and analysis. It should specify that the agent first identified the sensors for Chiller 9, then selected the <i>Chiller 9 Power Input</i> sensor, and successfully retrieved the energy usage data for the specified time period.</p> <p>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 <b>Time Series Forecasting model</b> 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.</p>

## D.5 SCENARIO COMPARISON WITH OTHER BENCH

We prepare a table to compare with the literature in Table 17. AssetOpsBench extends prior benchmarks by incorporating temporal/dynamic queries, name disambiguation, and tool-output–driven operations. These capabilities not present in TaskBench or ITBench. Additionally, while earlier benchmarks rely on either complex tool graphs or simpler single-step tools, AssetOpsBench emphasizes multi-step tool reuse, aligning better with real industrial agent workflows.

Table 17: 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	×	×	✓

2052  
2053

## D.6 USER STUDY RELIABILITY ANALYSIS

2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062

To quantitatively assess the realism of AssetOpsBench scenarios, we conducted a human evaluation study. We randomly selected 25 representative scenarios covering four categories: IoT queries, time-series forecasting (TSFM), work orders/events, and failure mode reasoning (FMSR). Participants were domain experts, including reliability engineers, maintenance engineers, and data scientists familiar with condition-based monitoring and predictive maintenance. Each participant evaluated scenarios using a 3-point scale ( $1 = \text{Not Realistic}$ ,  $2 = \text{Realistic}$ ,  $3 = \text{High Realistic}$ ) and could optionally provide qualitative comments. Background questions captured participants' role, years of experience, and familiarity with predictive maintenance. Responses were collected via a Google Form as shown in Figure 16.

2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070

Scenario 216 (TSFM): Forecast 'Chiller 9 Condenser Water Flow' using data in 'chiller9\_annotated\_small\_test.csv'. Use parameter 'Timestamp' as a timestamp. \*

Use the following parameters as inputs: 'Chiller 9 Liquid Refrigerant Evaporator Temperature, Chiller 9 Return Temperature, Chiller 9 Tonnage, Chiller 9 Setpoint Temperature, Chiller 9 Supply Temperature, Chiller 9 Chiller % Loaded, Chiller 9 Condenser Water Supply To Chiller Temperature, Chiller 9 Power Input, Chiller 9 Chiller Efficiency'.

1                    2                    3

Not Realistic                High Realistic

Optional comment for scenario 216

Your answer

2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080

Figure 16: Representative example of Scenario for Collecting user feedback

2081  
2082  
2083  
2084  
2085

The following metrics were computed to assess internal consistency, inter-rater agreement, and the reliability of aggregated scores.

2086  
2087  
2088

## D.7 RELIABILITY METRICS

2089  
2090

Table 18: Summary of Reliability Metrics for User Study Ratings

2091  
2092  
2093  
2094  
2095  
2096  
2097

Metric	Value
Cronbach's Alpha	0.8871
ICC(1)	0.2334
ICC(2)	0.8817
Fleiss' Kappa	0.2093

2098  
2099

## D.8 INTERPRETATION

2100  
2101  
2102  
2103  
2104  
2105

The internal consistency of the 25 scenario ratings is excellent, as indicated by a Cronbach's alpha of 0.887. The ICC(1) value of 0.233 reflects moderate agreement at the individual-participant level, whereas ICC(2) of 0.882 demonstrates that the aggregated ratings across participants are highly reliable. Fleiss' kappa of 0.209 indicates slight-to-fair categorical agreement among participants, which is consistent with the subjective nature of realism judgments. Overall, while individual participants may vary in their ratings, the averaged scores per scenario provide a stable and trustworthy measure of perceived realism.

2106 E GROUND TRUTH PREPARATION FOR REFERENCE-BASED EVALUATION  
2107

2108 To ensure that each scenario can be objectively evaluated, we first construct a ground-truth specification  
2109 that precisely defines the expected reasoning steps and final answer. The example in Listing  
2110 3 illustrates FMSR agent’s task where the system must retrieve sensor names associated with a wind  
2111 turbine. The ground truth includes the task description, the required planning step, and the exact  
2112 sequence of execution actions that a correct agent should follow. By explicitly defining the operations—such as calling `get_available_sensor_information` with the asset name “Wind Turbine”, the  
2113 ground truth provides a verifiable reference trace. This structure allows us to compare an agent’s  
2114 generated actions and outputs against a deterministic set of expected behaviors, ensuring consistent  
2115 and reproducible evaluation across models.  
2116

2117 Listing 3: Example FMSR task specification.

```

2118 {
2119     "id": 105,
2120     "type": "FMSR",
2121     "deterministic": false,
2122     "characteristic_form": "the answer should contain a list of sensor
2123         names for asset wind turbine.",
2124     "text": "Provide some sensors of asset Wind Turbine.",
2125     "planning_steps": [
2126         "Provide some sensors of asset Wind Turbine."
2127     ],
2128     "execution_steps": [
2129         {
2130             "name": "get_available_sensor_information",
2131             "action": "Get Available Sensor Information",
2132             "arguments": "Wind Turbine",
2133             "outputs": "[a list of sensor names]"
2134         },
2135         {
2136             "name": "finish",
2137             "action": "Finish",
2138             "arguments": "",
2139             "outputs": ""
2140         }
2141     ],
2142     "execution_links": [
2143         {
2144             "source": "get_available_sensor_information",
2145             "target": "finish"
2146         }
2147     ]
2148 }
```

2149 E.1 PLAN-EXECUTE REFERENCE-BASED SCORING  
2150

2151 To assess the fidelity of generated outputs, we perform **reference-based scoring** using ROUGE met-  
2152 rics. This evaluation is limited to the **Plan-Execute** paradigm to maintain consistency and preserve  
2153 the experimental flow. ROUGE metrics used include:

- 2154 • rouge1: unigram (1-gram) overlap between generated and reference outputs.
- 2155 • rouge2: bigram (2-gram) overlap.
- 2156 • rougeL: longest common subsequence between generated and reference sequences.
- 2157 • rougeLsum: line-wise longest common subsequence for multi-line outputs.

2158 E.2 EXECUTION CHAIN EVALUATION  
2159

2160 To systematically evaluate agent task execution, we design a **chain-based execution scoring**  
2161 method. In many scenarios, an agent performs a sequence of steps corresponding to *Think-Act-*

2160     Observe cycles. Ground truth data provides the expected sequence of steps for each task. Each  
 2161     executed step contains a name (representing the action) and an arguments field.  
 2162

2163     Our scoring approach compares an agent’s executed sequence with the ground truth sequence using  
 2164     three criteria:

- 2165     1. **Step Matching:** The name of each executed step is matched to the corresponding ground  
 2166        truth step. Unlike exact matching, we allow fuzzy matching based on string similarity using  
 2167        a threshold to account for minor variations in step names.
- 2168     2. **Argument Similarity:** Step arguments are treated as strings and compared using a  
 2169        ROUGE-like similarity metric (via `difflib.SequenceMatcher`). This captures  
 2170        cases where the agent produces slightly different or paraphrased arguments.
- 2171     3. **Sequence Coverage and Order:**
  - 2172        • Coverage penalizes missing ground truth steps.
  - 2173        • Extra steps are penalized proportionally.
  - 2174        • Order preservation is evaluated: steps executed out-of-order incur a penalty.

2176     The final **Execution Chain Score** for a single trajectory is computed as:

$$2177 \quad \text{Score} = (\text{Average argument similarity over matched steps}) \times (1 - \text{extra step penalty}) \times (1 - \text{order penalty})$$

2179     This produces a single scalar in  $[0, 1]$  summarizing how closely an agent’s execution matches the  
 2180     ground truth. Algorithm 1 outline the entire process.

---

2181     **Algorithm 1:** Compute Chain Execution Score

---

2183     **Input** : Ground truth steps  $GT$ , agent steps  $AG$ , name threshold  $\theta$ , name weight  $w_n$ ,  
 2184        argument weight  $w_a$   
 2185     **Output:** Final chain execution score  $S \in [0, 1]$   
 $2186 \quad matched \leftarrow \emptyset;$   
 $2187 \quad step\_scores \leftarrow [];$   
 $2188 \quad \text{foreach } gt \in GT \text{ do}$   
 $2189 \quad \quad best\_score \leftarrow 0;$   
 $2190 \quad \quad best\_idx \leftarrow \text{None};$   
 $2191 \quad \quad \text{foreach } ag \in AG \text{ do}$   
 $2192 \quad \quad \quad \text{if } ag \in matched \text{ then}$   
 $2193 \quad \quad \quad \quad \quad \text{continue};$   
 $2194 \quad \quad \quad \quad \quad name\_sim \leftarrow similarity(gt.name, ag.name);$   
 $2195 \quad \quad \quad \quad \quad \text{if } name\_sim < \theta \text{ then}$   
 $2196 \quad \quad \quad \quad \quad \quad \text{continue};$   
 $2197 \quad \quad \quad \quad \quad arg\_sim \leftarrow similarity(gt.arguments, ag.arguments);$   
 $2198 \quad \quad \quad \quad \quad score \leftarrow w_n \cdot name\_sim + w_a \cdot arg\_sim;$   
 $2199 \quad \quad \quad \quad \quad \text{if } score > best\_score \text{ then}$   
 $2200 \quad \quad \quad \quad \quad \quad \quad best\_score \leftarrow score;$   
 $2201 \quad \quad \quad \quad \quad \quad \quad best\_idx \leftarrow \text{index of } ag;$   
 $2202 \quad \quad \quad \quad \quad \text{if } best\_idx \neq \text{None} \text{ then}$   
 $2203 \quad \quad \quad \quad \quad \quad \quad add best\_idx to matched;$   
 $2204 \quad \quad \quad \quad \quad append best\_score to step\_scores;$   
 $2205 \quad \quad step\_coverage \leftarrow average(step\_scores);$   
 $2206 \quad \quad extra\_penalty \leftarrow \frac{|AG| - |matched|}{|GT| + |AG| - |matched|};$   
 $2207 \quad \quad order\_penalty \leftarrow \text{fraction of inversions in } matched \text{ indices};$   
 $2208 \quad \quad S \leftarrow step\_coverage \cdot (1 - extra\_penalty) \cdot (1 - order\_penalty);$   
 $2209 \quad \quad \text{return } S;$   


---

2211     **F ADDITIONAL BENCHMARK EXPERIMENTS**

2212     This appendix section contain outcome of an extensive benchmark we conducted in this paper.

2214  
2215

## F.1 LLM-AS-A-JUDGE EVALUATION AGENT AND HUMAN VALIDATION

2216  
2217  
2218  
2219  
2220  
2221  
2222

In Listing 19, we provided the system prompt that we used for generating a rubric metric for the evaluation agent. Given LLM is used for generating the rubric metric, we also conducted human validation of these generated metric. The results shown in Section 5.1, `llama-4-maverick` is selected to be the LLM of evaluation agent. Table 19 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.

2223  
2224  
2225

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.

2253  
2254

Table 19: Prompt instruction for LLM-as-a-judge evaluation agent

2255  
2256  
2257  
2258  
2259  
2260  
2261

Human validation was performed using Google Forms. As illustrated in Figure 17, domain experts were presented with the original task description, the agent's reasoning and final answer, and a checklist covering six evaluation dimensions. Each dimension was evaluated using binary judgments (True/False), enabling structured comparison between human and model evaluations. Four forms, each containing 10 samples, were distributed, yielding a total of 240 data points. The aggregated results are reported in Section 5.1.

2262  
2263

## F.2 DEEP INVESTIGATION OF AGENT-AS-TOOL PERFORMANCE

2264  
2265  
2266  
2267

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 18) 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

2268  
 2269 AssetOpsBench Human Judge Test  
 2270  
 2271  
 2272 Group: 1, Index: 0  
 2273  
 2274 ## Task: ##  
 2275 Generate a machine learning recipe for detecting overheating failure for Chiller 6. Result  
 2276 should include feature sensors and target sensor.  
 2277  
 2278 ## Final answer: ##  
 2279 For detecting "Compressor Overheating" failure in Chiller 6, the relevant feature sensors are:  
 2280 'Chiller 6 Power Input', 'Chiller 6 Evaporator Water Outlet Temperature', 'Chiller 6 Condenser  
 2281 Water Inlet Temperature', 'Chiller 6 Condenser Water Outlet Temperature', and 'Chiller 6  
 2282 Compressor Discharge Temperature'. The target sensor is 'Chiller 6 Compressor Discharge  
 2283 Temperature'. A machine learning model can be trained using these features to predict the  
 2284 likelihood of compressor overheating. ## Step 1: Understand the Task The task is to  
 2285 generate a machine learning recipe for detecting overheating failure for Chiller 6, including  
 2286 feature sensors and target sensor. ## Step 2: Identify Failure Modes for Chiller 6 The failure  
 2287 modes for Chiller 6 are identified as: 'Compressor Overheating: Failed due to Normal wear,  
 2288 overheating', 'Heat Exchangers: Fans: Degraded motor or worn bearing due to Normal use',  
 2289 'Evaporator Water side fouling', 'Condenser Water side fouling', 'Condenser Improper water  
 2290 side flow rate', 'Purge Unit Excessive purge', 'Refrigerant Operated Control Valve Failed  
 2291 spring'. ## Step 3: Determine Relevant Sensors for "Compressor Overheating" The relevant  
 2292 sensors for monitoring "Compressor Overheating" are: 'Chiller 6 Power Input', 'Chiller 6  
 2293 Evaporator Water Outlet Temperature', 'Chiller 6 Condenser Water Inlet Temperature', 'Chiller  
 2294 6 Condenser Water Outlet Temperature', and 'Chiller 6 Compressor Discharge Temperature'.  
 2295 ## Step 4: Compile Machine Learning Recipe For detecting "Compressor Overheating"  
 2296 failure in Chiller 6, the relevant feature sensors are: 'Chiller 6 Power Input', 'Chiller 6  
 2297 Evaporator Water Outlet Temperature', 'Chiller 6 Condenser Water Inlet Temperature', 'Chiller  
 2298 6 Condenser Water Outlet Temperature'. The target sensor is 'Chiller 6 Compressor  
 2299 Discharge Temperature'. A machine learning model can be trained using these features to  
 2300 predict the likelihood of compressor overheating. ## Step 5: Provide Final Answer For  
 2301 detecting "Compressor Overheating" failure in Chiller 6, the relevant feature sensors are:  
 2302 'Chiller 6 Power Input', 'Chiller 6 Evaporator Water Outlet Temperature', 'Ch  
 2303  
 2304  
 2305 ## Thinking process: ##  
 2306 Thought: I need to generate a machine learning recipe for detecting overheating failure for  
 2307 Chiller 6, which includes identifying the relevant feature sensors and the target sensor. I will  
 2308 use FMSRAgent to achieve this.  
 2309  
 2310 Action: Tool Name: FMSRAgent , Tool Parameter: Action Input 1: request=Generate a  
 2311 machine learning recipe for detecting overheating failure for Chiller 6, including feature  
 2312

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

2313  
 2314  
 2315  
 2316 benchmarks, with higher values indicating stronger task alignment. Among the models com-  
 2317 pared, gpt-4.1-2025-04-14 demonstrates the most consistent and well-rounded performance,  
 2318 achieving near-saturation in FMSR (100%) and strong results in End-to-End integration. In contrast,  
 2319 granite-3-3-8b-instruct and 11lama-3-3-70b-instruct perform well in TSFM and  
 2320 FMSR but underperform in WO-related tasks, which are particularly challenging due to their de-  
 2321 pendence on structured document comprehension and task planning. The 11lama-4-maverick  
 model shows promising results in WO and End-to-End integration, indicating a potential optimiza-

tion 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, multi-modal industrial applications.

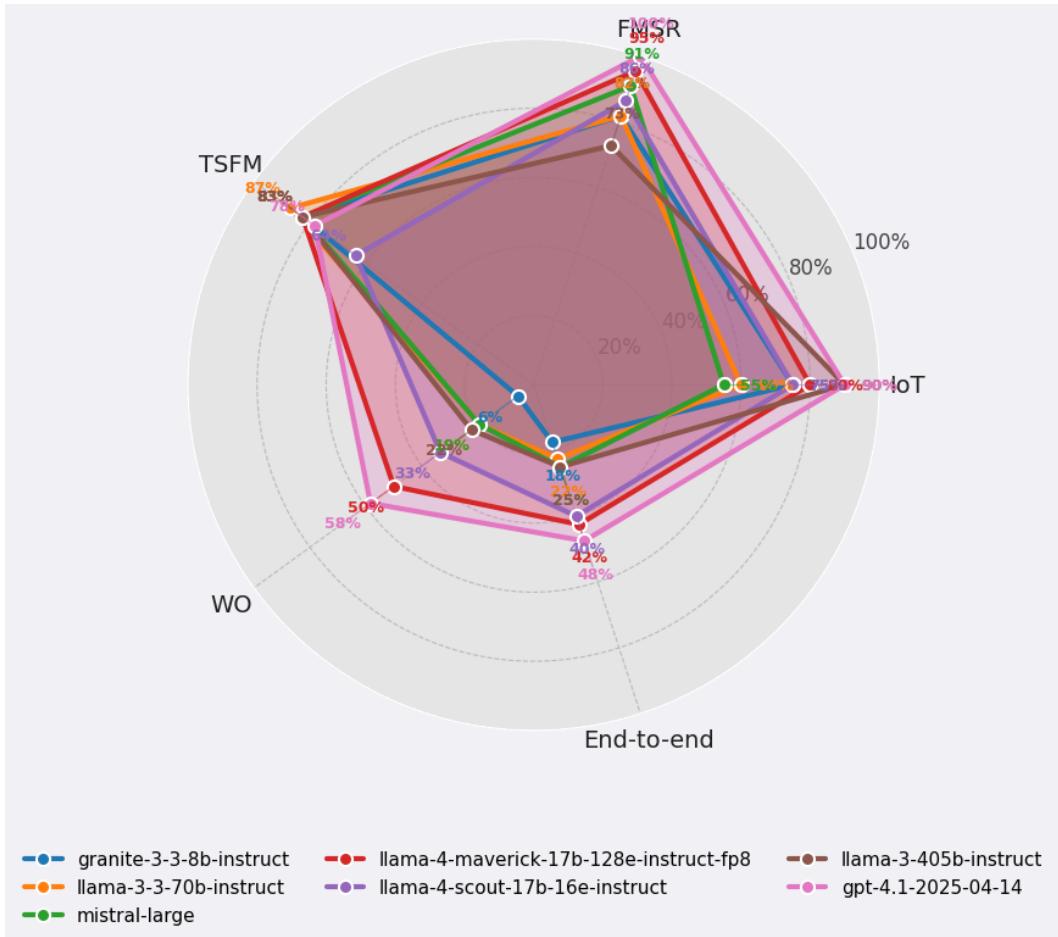


Figure 18: Task wise distribution of the Accomplished Tasks.

### Insight I

The radar chart highlights that while `gpt-4.1-2025-04-14` delivers the most balanced performance across all industrial task categories, other models exhibit strong specialization. For example, `granite-3-3-8b-instruct` and `llama-3-3-70b-instruct` excel in FMSR and TSFM but struggle with WO-related tasks. This reveals a clear trade-off between broad generalization and domain-specific strengths, suggesting that hybrid or task-specialized agent architectures may be most effective in practice.

### F.3 FAILURE ANALYSIS ON TOOL USE

As part of the AssetOpsBench trajectory analysis, we examine how agents interact with the available toolset and execution environment. We start this effort from the analyze (1) the complete list of all tools in Table 5 (containing the details of their name, usage, and parameters), and (2) those JSON snippets illustrating how agent actions are logged. Across **834 trajectories**, collected from multiple LLM models and multiple agent configurations, we store every agent step as a structured JSON record containing both the *action type* and its *execution state*. This representation allows us to distinguish between **Tool-oriented actions** (invocations of predefined data retrieval, analysis, and

2376 analytic functions) and **CodeReAct-oriented actions** (Python code generated and executed on the  
 2377 fly).

2378 Across **834 trajectories** collected from multiple LLM models and agent configurations, every agent  
 2379 step is logged as a structured JSON record containing both the *action type* (e.g., Tool-oriented or  
 2380 CodeReAct-oriented) and its associated *execution state*. This enables a unified analysis of failure  
 2381 cases across heterogeneous agent architectures.

2382 We distinguish between **Tool-oriented actions**, which invoke predefined data retrieval, analysis,  
 2383 or analytic functions, and **CodeReAct-oriented actions**, where the agent emits executable Python  
 2384 code. Each action record includes fields such as `action`, `action_input`, `observation`, and  
 2385 `state`. A failure is captured when the framework sets `state = "Invalid Action"` and  
 2386 records the underlying error in the `observation` field. These signals form the basis of our failure-  
 2387 mode statistics.

2388 **Example 1: Tool-Oriented Invalid Action (trimmed).**

```
2389
2390
2391 {
2392     "step": 1,
2393     "thought": "Use 'Read Sensors From File' to list sensors for Chiller 6.",
2394     "action": "Read Sensors From File",
2395     "action_input": "/path/to/chiller_6_sensors.txt",
2396     "observation": "Error: 'NoneType' object has no attribute 'replace'",
2397     "state": "Invalid Action"
2398 }
```

2399 This failure arises from malformed input to the `Read Sensors From File` tool. The associated  
 2400 error message is used directly in computing tool-level failure frequencies (see Figure 19).

2401 **Example 2: CodeReAct (Dynamic Python) Invalid Action (trimmed).**

```
2402
2403
2404 {
2405     "step": 2,
2406     "thought": "Write preventive work orders to JSON.",
2407     "action": "import json
2408     with open('pwo.json','w') as f:
2409         json.dump(pwo_list, f)",
2410     "observation": "Invalid action: Object of type Timestamp is not JSON serializable",
2411     "state": "Invalid Action"
2412 }
```

2412 This *CodeReAct* step is syntactically correct but fails during execution because the object being  
 2413 serialized contains a non-JSON-serializable `Timestamp`. These runtime failures contribute to the  
 2414 action-state distribution analysis.

2415 Figure 19 summarizes the distribution of valid and invalid executions for both action classes. Tool-  
 2416 oriented actions benefit from well-defined schemas, while CodeReAct-oriented actions incur more  
 2417 failures due to the variability of generated Python code.

2418 Figure 20 highlights the concentration of Tool-oriented failures within a handful of tools such as  
 2419 `jsonreader`, `tsfm_integrated_tsad`, and `Read Sensors From File`. These patterns  
 2420 reflect the structural complexity of their inputs and outputs. The JSON fragments above illustrate  
 2421 the types of failures contributing to these distributions.

2422 **Execution-State Insight**

2423 Analysis of 834 agent trajectories reveals that **Tool-oriented actions** achieve higher valid-  
 2424 execution rates due to well-defined input/output schemas, whereas **CodeReAct-oriented**  
 2425 **actions** suffer more runtime failures from dynamic Python generation. Failures in Tool-  
 2426 oriented actions are concentrated in a few complex tools, highlighting that structural com-  
 2427 plexity and input validation are critical factors in agent reliability.

2430

2431

2432

2433

2434

2435

2436

2437

2438

2439

2440

2441

2442

2443

2444

2445

2446

2447

2448

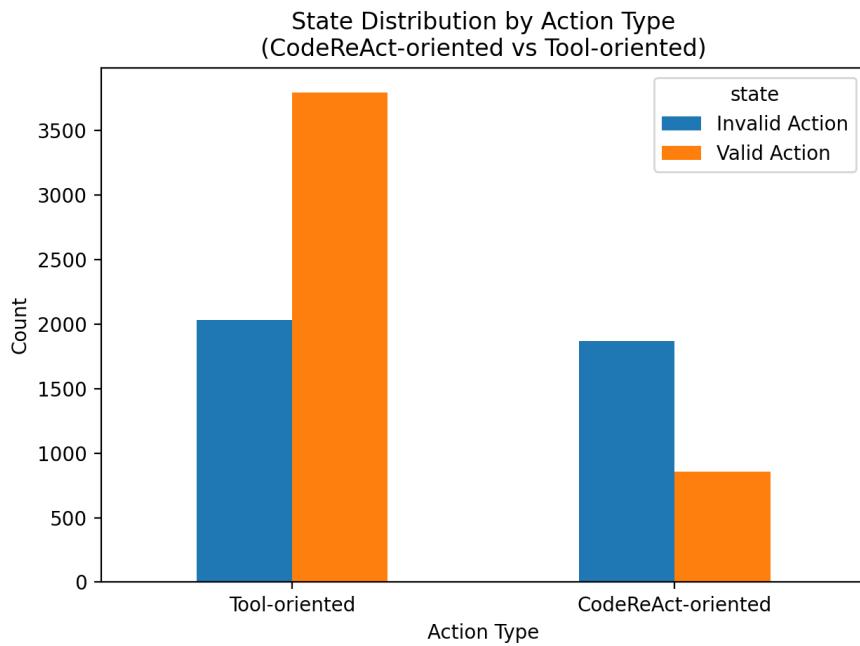
2449

2450

2451

2452

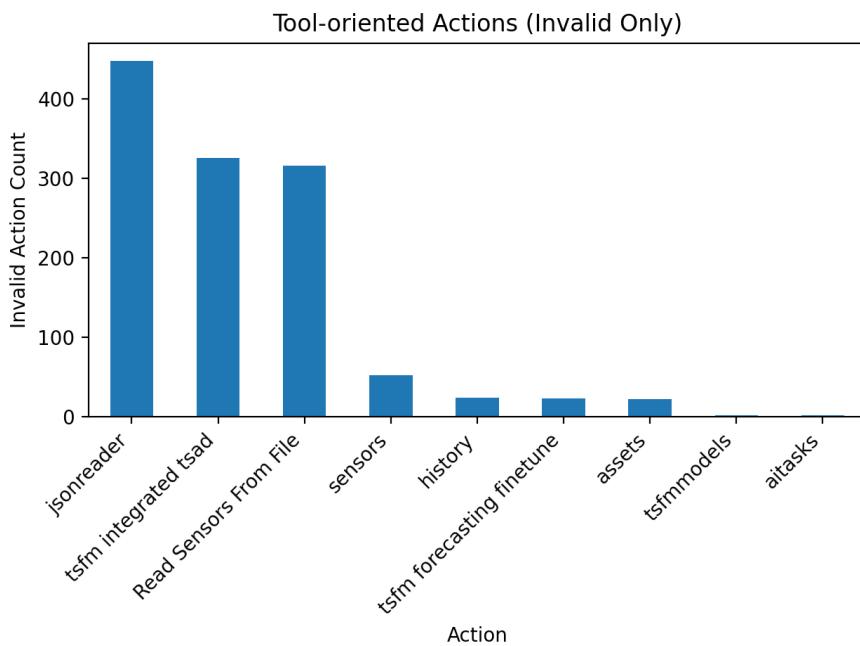
Figure 19: Distribution of execution states across action types. Tool-oriented actions exhibit higher valid-execution rates, whereas dynamically generated Python produces more execution failures.



2454

2455

2456



2475

2476

Figure 20: Invalid-only failures for Tool-oriented actions. Errors are concentrated in a small number of frequently used tools with complex parsing or I/O behavior.

2479

2480

2481

#### F.4 EXECUTION EFFICIENCY

2482

2483

In this section, we analyze AssetOpsBench execution efficiency of 7 LLMs, complementing the Leaderboard results in Section 5.1. Tables 20 and 21 present results from two multi-agent imple-

2484     gements. Metrics include the average number of steps taken per task and the average runtime (in  
2485     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.

Table 20: 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	$6.0 \pm 2.4$	$104 \pm 178$	$6.4 \pm 2.5$	$218 \pm 371$
mistral-large	$4.9 \pm 2.6$	$347 \pm 19871$	$5.2 \pm 2.2$	$289 \pm 443$
llama-3-405b-instruct	$4.8 \pm 2.5$	$250 \pm 773$	$5.6 \pm 2.2$	$255 \pm 248$
llama-3-70b-instruct	$3.9 \pm 1.6$	$101 \pm 107$	$4.3 \pm 2.1$	$151 \pm 220$
llama-4-maverick-17b-128e	$4.3 \pm 1.5$	$120 \pm 258$	$4.5 \pm 1.7$	$137 \pm 175$
llama-4-scout-17b-16e-instruct	$4.4 \pm 2.0$	$101 \pm 87$	$5.8 \pm 2.9$	$178 \pm 157$
granite-3-3-8b	$5.3 \pm 3.1$	$197 \pm 240$	$6.6 \pm 3.6$	$228 \pm 256$

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.

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

Table 21: 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-

2538 Tool setting, compared to only 38–44% on most metrics in Plan-Execute. Similarly,  
 2539 `llama-4-maverick-17b-128e-instruct` excels in both setups but scores notably higher  
 2540 in Agent-As-Tool, achieving 59–78% on core performance metrics versus 45–57% in Plan-Execute.  
 2541

2542 This pattern is consistent across most models: **Agent-As-Tool** incur higher execution costs but  
 2543 deliver better reasoning fidelity. Conversely, **Plan-Execute** agents—while faster and more struc-  
 2544 tured—often struggle with complex retrieval, verification, and consistency tasks. These findings  
 2545 suggest a fundamental trade-off: Plan-Execute offers process efficiency, while Agent-As-Tool yield  
 2546 higher end-task quality—a crucial insight for selecting agent architectures based on application goals  
 2547 such as throughput vs. correctness.

### 2548 **Insight II**

2549 Across both execution modes, our results reveal a fundamental trade-off: *Plan-Execute*  
 2550 agents are faster and more step-efficient, yet consistently underperform in reasoning-  
 2551 intensive tasks, whereas *Agent-As-Tool* agents incur higher execution costs but deliver sub-  
 2552 stantially stronger task accuracy and reliability. This shows that efficiency-oriented archi-  
 2553 tectures do not automatically yield better end-task quality : an important consideration for  
 2554 industrial LLM systems where correctness often outweighs runtime.

### 2556 **Insight III**

2557 Our analysis reveals a subtle behavioral difference across execution modes: although *Single-  
 2558 Agent* tasks are intended to be completed by one agent in a single round, *Plan-Execute* agents  
 2559 tend to over-plan, potentially reusing the same agent multiple times or invoking additional  
 2560 agents. Conversely, *Agent-As-Tool* agents, despite being reactive and theoretically able to  
 2561 finish in one round, often continue executing, indicating the presence of tasks that exceed  
 2562 an individual agent’s capability and lead to repeated attempts. This underscores hidden  
 2563 task difficulty and model limitations, which are critical for interpreting execution efficiency  
 2564 alongside task performance. A common observation arising from this behavior is the impor-  
 2565 tance of enabling parent agents to ask questions to other agents within the system to resolve  
 2566 tasks more efficiently.

## 2569 **F.5 TASK COMPLETION COMPARISON: AGENT-AS-TOOL VS PLAN-EXECUTE**

2570 We evaluate the performance of multiple LLM models on the Task Completion metric under two  
 2571 execution paradigms: **Agent-As-Tool** and **Plan-Execute**. Agent-As-Tool agents execute tasks in-  
 2572 crementally by invoking tools as needed, whereas Plan-Execute agents first plan a full sequence of  
 2573 steps before execution. Figure 21 visualizes the leaderboard across seven representative models.  
 2574

### 2576 **Observations.**

- 2578 • `gpt-4.1-2025-04-14` and `llama-4-maverick-17b-128e-instruct-fp8`  
 2579 achieve the highest task completion under the Agent-As-Tool paradigm, reaching 65% and  
 2580 59% respectively.
- 2581 • Plan-Execute results show mixed trends: some models, such as `mistral-large`, im-  
 2582 prove slightly (40% → 46.5%), while others, notably `gpt-4.1`, experience a drop (65%  
 2583 → 38.38%).
- 2584 • Smaller or older models (`granite-3-3-8b-instruct`,  
 2585 `llama-3-3-70b-instruct`) exhibit lower task completion across both paradigms,  
 2586 highlighting limitations in multi-step reasoning and tool integration.
- 2587 • Differences between the paradigms suggest that pre-planning can sometimes limit adapt-  
 2588 ability, whereas reactive Agent-As-Tool strategies better leverage incremental reasoning  
 2589 and tool invocations for complex tasks.

2591 **Insights.** The comparison highlights several key points:

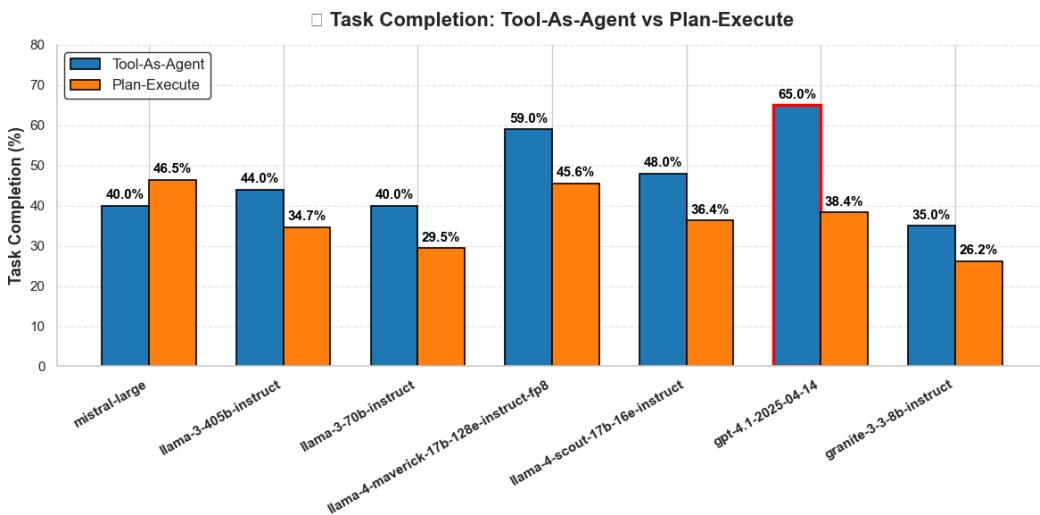


Figure 21: Task Completion scores (%) for Agent-As-Tool and Plan-Execute paradigms. Values above bars indicate the actual completion percentage for each model.

- Model capabilities and adaptability:** High-performing models demonstrate both accurate reasoning and effective tool utilization. GPT-4.1 and LLaMA-4-Maverick excel when allowed to execute incrementally in the Agent-As-Tool paradigm.
- Impact of execution paradigm:** Agent-As-Tool allows flexible, context-aware reasoning, particularly for models capable of dynamically selecting the next best action. Plan-Execute may underperform if initial plans are suboptimal or if unexpected states arise during execution.
- Guidance for benchmark design:** Reporting Task Completion under both reactive (Agent-As-Tool) and planned (Plan-Execute) paradigms is crucial to capture model strengths and weaknesses in real-world agentic tool-use.
- Future directions:** Incorporating dynamic feedback loops in Plan-Execute or hybridizing with Agent-As-Tool strategies may further improve task completion, especially for larger multi-step tasks.

### Insight III

The Task Completion comparison highlights that execution paradigm significantly interacts with model capability. High-performing models such as GPT-4.1 and LLaMA-4-Maverick achieve their best results under the *Agent-As-Tool* paradigm, leveraging incremental reasoning and dynamic tool selection. In contrast, Plan-Execute sometimes constrains adaptability, leading to lower performance for models with strong reasoning potential. This suggests that reactive, step-wise strategies can better exploit advanced model reasoning, whereas pre-planned sequences may underutilize model strengths, particularly on multi-step or complex tasks.

## F.6 RUNTIME AND COST ANALYSIS

Table 22 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.

2646 Table 22: Runtime and estimated cost for executing 140+ utterance tasks using the Agents-as-Tools  
 2647 paradigm.

2649 <b>LLM</b>	2650 <b>Provider</b>	2651 <b>Avg Tokens per Task</b>	2652 <b>Total Cost (USD)</b>
2653 gpt-4.1	2654 OpenAI	2655 $\approx 3,664$	2656 \$300.00
2657 llama-4-maverick	2658 Watsonx	2659 $\approx 3,730$	2660 \$130.00

## 2655 F.7 UNCERTAINTY ANALYSIS

2656 As discussed in Section 5, the evaluation agent was run five times to produce reliable performance  
 2657 metrics. Table 23 shows the inter-rater agreement across these five evaluation runs, along with the  
 2658 derived uncertainty (computed as  $1 - \text{agreement}$ ). The average agreement and uncertainty across all  
 2659 metrics are also reported.

2661 Table 23: Inter-rater agreement and derived uncertainty across five evaluation runs.

2663 <b>Metric</b>	2664 <b>Agreement</b>	2665 <b>Uncertainty</b>
2666 Task Completion	2667 0.9731	2668 2.69%
2669 Data Retrieval Accuracy	2670 0.9697	2671 3.03%
2672 Generalized Result Verification	2673 0.9681	2674 3.19%

## 2675 F.8 ABLATION EXPERIMENTS

2676 In this section, we present the detailed report of the ablation study. We fixed the Agent-As-Tool  
 2677 paradigm and conducted both sets of experiments.

### 2678 F.8.1 DISTRACTOR AGENTS

2679 We have introduced 10 distractor agents to intentionally increase the complexity and ambiguity for  
 2680 global agents. Table 24 categorizes these agents based on their respective domains and functional  
 2681 roles. The set includes both general-purpose agents, such as those for echoing inputs or handling  
 2682 off-topic queries, and domain-specific agents focused on tasks like predictive maintenance, sensor  
 2683 data summarization, and edge ML deployment. This taxonomy enhances the realism of multi-agent  
 2684 environments by supporting modular integration and introducing controlled confusion.

2685 Across 99 scenarios, we compare language models with and without distractor agents to evaluate  
 2686 their robustness in agentic tool-use settings. Table 25 shows the result. GPT-4.1 remains  
 2687 the strongest model overall, achieving the highest scores in task completion, data accuracy, and  
 2688 result verification across both settings. Llama-4-Maverick emerges as the best-performing  
 2689 open-weights model, showing not only high accuracy but also improved performance when dis-  
 2690 tractors are introduced. In contrast, models such as Mistral-Large and Llama-4-Scout  
 2691 experience moderate degradation under distractors, indicating sensitivity to noisy action spaces.  
 2692 Granite-3-3-8B remains stable across conditions but at a lower overall accuracy level, showing  
 2693 reliability but limited reasoning depth.

2694 The introduction of distractor agents reveals interesting behavioral differences. While most models  
 2695 suffer performance drops, Llama-3-405B and Llama-4-Maverick improve across all three  
 2696 evaluation dimensions, suggesting strong corrective reasoning and robustness to tool-selection noise.  
 2697 These results highlight a tiered landscape of model reliability: GPT-4.1 at the top, followed by  
 2698 mid-tier models with varying sensitivities, and smaller models offering stability at reduced capabili-  
 2699 ty. Overall, the findings underscore the importance of evaluating both accuracy and robustness, as  
 real-world agentic systems often face ambiguous or misleading tool/agent choices.

Table 24: 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.

#### Insight 4: Robustness to Distractor Agents

Across 99 scenarios, GPT-4.1 consistently demonstrates top performance, while Llama-4-Maverick shows strong open-weight results with robustness to distractor agents. Mid-tier models exhibit varying sensitivity to noisy action spaces, and smaller models like Granite-3-3-8B maintain stability but with lower overall accuracy. These findings reveal a tiered landscape of model reliability, emphasizing that both task accuracy and resilience to distractors are critical for practical agentic tool-use systems.

#### F.8.2 IMPACT OF IN-CONTEXT EXAMPLES

Table 26 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.

**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 F.8). These results reinforce the conclusion that in-context exam-

2754 Table 25: Comparison of Model Performance With and Without Distractor Agents (99 Scenarios).  
2755

2756 <b>Model</b>	2757 <b>Setting</b>	2758 <b>Task</b>	2759 <b>Data</b>	2760 <b>Result</b>
		2761 <b>Completion</b>	2762 <b>Accuracy</b>	2763 <b>Verification</b>
2764 gpt-4.1-2025-04-14	Without Distractors	52	57	55
	With Distractors	48	56	54
2765 granite-3-3-8b-instruct	Without Distractors	40	44	41
	With Distractors	40	44	41
2766 mistral-large	Without Distractors	42	46	43
	With Distractors	40	44	41
2767 llama-3-405b-instruct	Without Distractors	41	41	38
	With Distractors	44	44	44
2768 llama-3-3-70b-instruct	Without Distractors	38	43	34
	With Distractors	41	43	36
2769 llama-4-maverick	Without Distractors	46	49	46
	With Distractors	48	49	49
2770 llama-4-scout	Without Distractors	45	44	46
	With Distractors	40	40	40

2773 Table 26: Comparison of gpt-4.1 and granite-3-3-8b With/Without In-Context Examples  
2774 (# of Tasks = 65)  
2775

2776 <b>Model</b>	2777 <b>In-Context</b>	2778 <b>Task</b>	2779 <b>Data Retrieval</b>	2780 <b>Generalized Result</b>
		2781 <b>Completion</b>	2782 <b>Accuracy</b>	2783 <b>Verification</b>
2784 gpt-4.1	Yes	52	57	55
	Yes	40	44	41
2785 granite-3-3-8b	No	22	21	24
	No	2	3	3

2785 ples are essential for ReAct-style reasoning in LLM-based agents. We did not select tasks from WO  
2786 and E2E since their performance is already poor.

#### 2787 Insight: Impact of In-Context Examples

2788 The presence of in-context examples dramatically improves performance: gpt-4.1  
2789 achieves 80% average accuracy with examples versus 33% without, while  
2790 granite-3-3-8b drops from 60% to 3%. This highlights that effective ReAct-style  
2791 reasoning critically depends on relevant contextual guidance.

## 2794 F.9 PLAN-EXECUTE REFERENCE-BASED SCORING

2795 **Evaluation Setup.** To assess the fidelity of generated outputs, we perform **reference-based scoring**  
2796 using ROUGE metrics. This evaluation is limited to the **Plan-Execute** paradigm to maintain  
2797 consistency and preserve the experimental flow.

2798 ROUGE metrics used include:

- 2799 • rouge1: unigram (1-gram) overlap between generated and reference outputs.
- 2800 • rouge2: bigram (2-gram) overlap.
- 2801 • rougeL: longest common subsequence between generated and reference sequences.
- 2802 • rougeLsum: line-wise longest common subsequence for multi-line outputs.

2803 **Results Summary.** ROUGE scores highlight model differences in n-gram and sequence-level fi-  
2804 delity. Table 27 presents sample scores for representative models across Plan-Execute outputs.

Table 27: 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	<b>0.343</b>	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	<b>0.420</b>	<b>0.251</b>	<b>0.343</b>	<b>0.404</b>
llama-4-maverick	0.403	0.240	0.325	0.383

We conduct a more in-depth analysis:

- Top-performing models such as llama-3-405b-instruct and mixtral-8x7b-instruct-v01 achieve  $\text{rouge1} \approx 0.42$ ,  $\text{rouge2} \approx 0.26$ , and  $\text{rougeL} \approx 0.34$ , 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.

### Insight

Although gpt-4.1 excels in task reasoning and completion, its lower ROUGE scores compared to open-weight models indicate that strong reasoning does not always correspond to higher lexical alignment with reference outputs.

## F.10 AGENT-AS-TOOLREFERENCE-BASED SCORING

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.

**Results.** Table 28 reports ROUGE-1/2/L scores along with generation lengths. mistral-large achieves the highest performance with ROUGE-1  $\approx 0.37$ , ROUGE-2  $\approx 0.19$ , and ROUGE-L  $\approx 0.30$ , 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.

2862 Table 28: ROUGE-based comparison for the **Agent-As-Tool** setting. Scores are computed on the  
 2863 extracted *thinking* segments of each trajectory. Longer generations reduce lexical overlap with con-  
 2864 cise references, lowering ROUGE despite potentially richer content.

2866 Model	2867 ROUGE-1	2868 ROUGE-2	2869 ROUGE-L	2870 ROUGE-Lsum	#Samples	Pred. Avg. Words	GT Avg. Words
<b>mistral-large</b>	<b>0.3691</b>	<b>0.1933</b>	<b>0.2971</b>	<b>0.3124</b>	40	<b>83.0</b>	29.85
llama-3-3-70b-instruct	0.3661	<b>0.1963</b>	0.2971	<b>0.3177</b>	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
<b>gpt-4.1-2025-04-14</b>	<b>0.1628</b>	<b>0.0816</b>	<b>0.1332</b>	<b>0.1389</b>	40	<b>277.12</b>	29.85

2871

2872

2873 **Insight**

2874

2875 In the Agent-As-Tool setting, models generating reasoning traces closer in length to ref-  
 2876 erence answers (e.g., `mistral-large`, `llama-3-3-70b-instruct`) achieve higher  
 2877 ROUGE scores, whereas longer outputs from models like `gpt-4.1` reduce lexical overlap  
 2878 despite potentially valid and rich reasoning, highlighting that ROUGE penalizes verbosity  
 2879 rather than reasoning quality.

2880

2881 Next, Table 29 reports the **average execution scores** per model:

2882

2883 Model	2884 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
<code>mistralai/mistral-large</code>	0.113
openai-azure/gpt-4.1-2025-04-14	0.117

2891

2892 Table 29: Average Execution Chain Scores for different LLM models. Scores reflect alignment with  
 2893 ground truth sequences in terms of step name, argument similarity, and sequence coverage.

2894

2895 The results indicate that:

2896

- 2897 • `meta-llama/llama-3-405b-instruct`, `mistral-large`, and  
 2898 `gpt-4.1-2025-04-14` achieve the highest alignment with ground truth steps,  
 2899 demonstrating better handling of multi-step task execution in the Agent-As-Tool setting.
- 2900 • Larger models such as `llama-4-maverick` and `llama-4-scout` have moderate  
 2901 scores, suggesting that complexity alone does not guarantee faithful execution.
- 2902 • Smaller or older models, including `granite-3-3-8b` and `llama-3-3-70b`, exhibit  
 2903 lower scores, primarily due to missing steps, extra steps, or argument discrepancies.

2905

2906 Overall, this evaluation provides a quantitative, interpretable measure of how closely an agent's  
 2907 executed actions match the intended ground truth, complementing other performance metrics such  
 2908 as reference-based scoring (ROUGE) or semantic verification.

2909

2910

2911 **Execution Insight**

2912

2913

2914

2915

2916 Top-performing models (`llama-3-405b-instruct`, `mistral-large`, `gpt-4.1`)  
 2917 achieve the highest average execution scores, indicating superior alignment with ground-  
 2918 truth multi-step sequences. In contrast, larger models like `llama-4-maverick` show  
 2919 moderate alignment, highlighting that model size alone does not guarantee faithful task ex-  
 2920 ecution.

## 2916 G GENERALITY: NEW DATASETS AND SCENARIOS

2918 This section complements the generality discussion presented in Section 5.3. In total 166 scenarios  
 2919 are generated using 4 different datasets to support the generality study:  
 2920

- 2921 • We use two public datasets for condition monitoring of industrial assets, hosted on UCI,  
 2922 which provide programmatic access to descriptions and metadata:
  - 2924 – **Metro Train MetroPT-3 (15 scenarios)**: Created scenarios based on dataset descrip-  
 2925 tion and failure locations to test failure detection and reasoning.
  - 2926 – **Hydraulic System (17 scenarios)**: Generated scenarios for hydraulic pumps focusing  
 2927 on early fault identification and operational anomalies.
- 2928 • We also used internal datasets useful for condition monitoring of an industrial assets and  
 2929 ISO documents for testing an agent that is deployed in production system.
  - 2931 – **FailureSensorIQ (88 scenarios)**: Identify responsible sensor for early detection of  
 2932 failures.
  - 2933 – **Asset Health (42 scenarios)**: Assess the condition of an industrial asset based on its  
 2934 recent history.

### 2935 G.1 SCENARIO USING METROPT-3 DATASET

2936 The MetroPT dataset<sup>1</sup> is a real-world multivariate time-series dataset collected from the Air Production  
 2937 Unit (APU) of metro trains in Porto, Portugal. It contains readings from pressure, temperature,  
 2938 motor current, and air intake valves were collected from a compressor’s Air Production Unit (APU).  
 2939 The dataset includes documented failure events such as air and oil leaks, providing ground truth  
 2940 for predictive maintenance and anomaly detection tasks. MetroPT enables evaluation of IoT agent,  
 2941 FMSR agent and TSFM agent as this dataset is particularly suitable for temporal modeling, early  
 2942 fault detection, and remaining useful life estimation. Its high-resolution, real-world nature makes  
 2943 it a challenging benchmark for testing model robustness, interpretability, and real-time prediction  
 2944 capabilities. With the help of our internal SME, we created 15 complex scenarios and two examples  
 2945 are given in Table 30. We can see the reachness in type of analysis an end user is interested.  
 2946

2947 Table 30: Sample predictive maintenance scenarios for MetroPT-3 Dataset.  
 2948

2949 ID	2950 Scenario Description
2951 1	2952 Consider asset mp_1. After the maintenance performed on May 30, 2020, how has the 2953 compressor’s condition evolved during May 31 to June 6? Are there any indications that 2954 further repair or monitoring is needed?
2955 2	2956 Consider asset mp_1. From the compressor sensor data collected between May 29 and 2957 June 4, 2020, can we assess the likelihood of an air leak failure occurring within the 2958 subsequent week starting June 5? Is preventive maintenance advisable?

### 2959 G.2 SCENARIO USING HYDROLCIC SYSTEM DATASET

2960 The UCI Hydraulic Systems dataset was collected from a lab-scale hydraulic test rig equipped with  
 2961 multiple sensors reporting pressures, volume flows, temperatures, motor power, vibration, and cooling  
 2962 metrics. The rig cycles through constant 60-second loads, while four component fault types  
 2963 (cooler, valve, pump leakage, and accumulator) are varied across severity levels. With 2,205 in-  
 2964 stances and 43,680 features, the dataset is multivariate and structured for both classification and  
 2965 regression tasks. The condition of each component is annotated per cycle, enabling fault diagnosis  
 2966 and predictive maintenance modeling. With the help of our internal SME, we created 17 complex  
 2967 scenarios and two examples are given in Table 35.  
 2968

2969 <sup>1</sup><https://archive.ics.uci.edu/dataset/791/metropt+3+dataset>

2970 Table 31: Sample predictive maintenance scenarios for Hydrolic System Dataset.  
2971

2972 <b>ID</b>	2973 <b>Scenario Description</b>
2974 1	2975 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?
2976 2	2977 Consider asset hp_1. At 2024-01-22, can the hydraulic accumulator close to total failure be detected by analyzing sensor data spanning previous days? What sensor signatures confirm this state?",

2980  
2981 **G.3 ASSET HEALTH SCENARIO USING INTERNAL DATASET**  
29822983 Based on business unit requirements and in collaboration with domain experts, we created 42 sce-  
2984 narios for detecting asset health to conduct the benchmark study, primarily using work order data.  
2985 Each scenario follows the prescribed format described in the main paper. A representative example  
2986 is shown below:2987 

```
{  
2988     "id": 1000,  
2989     "file": "Air Handling Unit_615152AC_insights_prompt.txt",  
2990     "text": "You are an expert in Air Handling Unit maintenance and  
2991         reliability analysis. Your task is to analyze provided  
2992         asset_details_facts and workorder_facts...",  
2993     "type": "System Health",  
2994     "category": "Asset Analysis",  
2995     "deterministic": true,  
2996     "characteristic_form": "The expected condition of the asset is  
2997         'Not enough data' because only 4 work orders are available."  
2998 }
```

2999 One of the task types is **System Health**, aimed at evaluating the condition of an asset based on recent  
3000 system records (typically work orders, alerts, etc.) and raising flags such as *good* or *needs attention*.  
3001 Table 32 summarizes the coverage of the 42 scenarios across asset classes.3002  
3003 Table 32: Distribution of scenarios across asset types/classes.  
3004

3005 <b>Asset Type/Class</b>	3006 <b>Number of Unique Instances</b>
3006 Air Handling Unit	9
3007 CRAC	10
3008 Chiller	10
3009 Pump	8
3010 Boiler	5

3011 This diversity spans both horizontal coverage (different asset classes) and vertical variation (multiple  
3012 instances within each class), providing a robust testbed for evaluating agent generalization and per-  
3013 formance across operational conditions. All 42 scenarios fall under the **Asset Health** category and  
3014 primarily rely on work order information. Each scenario captures distinct aspects of asset behavior,  
3015 reflecting operational variability. Token count analysis provides insight into scenario complexity.3016 Over 60% of scenarios (26/42) fall in the 767–2,841 token range, reflecting mostly concise formats.  
3017 A long-tailed distribution exists to ensure LLMs handle both compact and extended input contexts.  
30183020 **G.4 FAILURESENSORQA DATASET USING ISO DOCUMENT**  
30213022 The FailureSensorQA dataset is designed to support predictive maintenance reasoning using struc-  
3023 tured knowledge from ISO standards and industrial asset documentation. Each scenario in the  
dataset presents a realistic diagnostic question, prompting the agent to identify relevant failure modes

3024 Table 33: Token count statistics for 42 Asset Health scenarios.  
3025

---

3026 <b>Statistic</b>	3027 <b>Value</b>
3028 Total scenarios	3029 42
3029 Median	3,277 tokens
3030 Mean	3,695 tokens
3031 Standard Deviation	3,125 tokens
3032 Minimum–Maximum	777–11,098 tokens
3033 Mode	1,316 tokens

---

3034 Table 34: Token count distribution across scenarios.  
3035

---

3037 <b>Token Range</b>	3038 <b># Scenarios</b>
3038 (767 – 2,841]	3039 26
3039 (2,841 – 4,905]	3040 6
3040 (4,905 – 6,970]	3041 1
3041 (6,970 – 9,034]	3042 3
3042 (9,034 – 11,098]	3043 6

---

3045 and determine the most informative sensors for early detection. Table 35 shows representative ex-  
3046 amples, including scenarios for aero gas turbines and compressors, where the task requires mapping  
3047 sensor readings such as vibration, temperature, or fuel flow to potential failure events. By leverag-  
3048 ing ISO-standardized descriptions, this dataset ensures that reasoning aligns with industry practices,  
3049 enabling evaluation of agent capabilities in sensor-failure attribution and condition monitoring. The  
3050 dataset emphasizes multi-step reasoning, sensor selection, and domain-specific knowledge integra-  
3051 tion, providing a challenging benchmark for testing predictive maintenance agents. We generated  
3052 total 88 scenarios with the help of our reliability engineers.

3053 Table 35: Representative scenarios from the FailureSensorQA dataset.  
3054

---

3056 <b>ID</b>	3057 <b>Scenario Description</b>
3058 1	3059 For aero gas turbine, list all the failure modes that can be detected or indicated by abnormal 3060 readings from vibration, speed, or fuel pressure/fuel flow.?
3060 2	3061 Which sensors are most effective for detecting failure modes related to vibration and tem- 3062 perature anomalies in a compressor?

---

3063 **SCENARIO EXECUTION AND EVALUATION**  
3064

3065 The 42 scenarios were executed across three models, resulting in 126 executions. Each execution  
3066 generates an output trajectory, which is subsequently analyzed by the Evaluation Agent across five  
3067 runs, yielding 630 evaluation instances. The Evaluation Agent compares outputs against the char-  
3068 acteristic form described in the scenario examples to calculate automated metrics. Manual review  
3069 was used to validate the final column of results, identifying only one case (Granite) where the model  
3070 overconfidently claimed task completion.

3071 **PERFORMANCE INSIGHTS**  
3072

3073 The scenarios primarily assess LLMs’ analytical skill—the ability to interpret provided information  
3074 and generate appropriate conclusions. Agents such as FMSR, which excel in skill-based reasoning  
3075 tasks, demonstrate strong performance, particularly in single-agent communication settings.

## 3078 H EMERGING FAILURE MODE DISCOVERY AND AGENT DEVELOPMENT 3079

3080 To support adaptive evaluation of multi-agent LLM systems, this appendix outlines the implemen-  
3081 tation details behind the failure discovery process. While the main text presents the empirical dis-  
3082 tribution of failure types—including emergent patterns—this appendix focuses on the structured  
3083 methodology used to extract and cluster novel failure behaviors beyond the MAST (Multi-Agent  
3084 System Failure Taxonomy) Cemri et al. (2025). The evaluation spanned 881 multi-agent trajec-  
3085 tories, drawn from diverse language model configurations. Trajectory distribution by model is as  
3086 follows:

- 3087 • mistral-large: 145 trajectories
- 3088 • llama-3-405b-instruct: 145 trajectories
- 3089 • llama-3-3-70b-instruct: 145 trajectories
- 3090 • llama-4-maverick-17b-128e-instruct-fp8: 125 trajectories
- 3091 • llama-4-scout-17b-16e-instruct: 111 trajectories
- 3092 • gpt-4.1-2025-04-14: 105 trajectories
- 3093 • granite-3-3-8b-instruct: 105 trajectories
- 3094
- 3095
- 3096

3097 Among the 881 utterance execution trajectories analyzed using an LLM-as-a-judge framework (se-  
3098 lected LLM judge model - *openai-azure/gpt-4.1-2025-04-14* as the LLM judge) to identify the  
3099 causes of multi-agent AI failures, we found that—beyond the existing MAST categories—185 tra-  
3100 jectories exhibited one additional failure reason, while 164 trajectories contained two distinct ad-  
3101 dditional failure reasons. This highlights the empirical necessity of taxonomy expansion to capture  
3102 compound and emergent failure patterns in real-world deployments. To extend the original MAST  
3103 taxonomy, we conducted a structured analysis of novel multi-agent system failures observed in re-  
3104 cent interaction traces. This subsection details our identification methodology and explains how the  
3105 resulting failure modes align with the MAST framework.

### 3106 H.0.1 ALGORITHM FOR EMERGING FAILURE MODES CLUSTERING 3107

3108 To systematically identify and normalize *emerging failure modes* observed in multi-agent LLM sys-  
3109 tem interactions, we introduce a structured algorithmic framework based on semantic embedding  
3110 and unsupervised clustering. This process abstracts unanticipated failure patterns into representa-  
3111 tive categories that either align with or extend the predefined MAST taxonomy.

3112 **Definitions and Notation.** Let:

- 3114 •  $T = \{t_1, \dots, t_n\}$ : Set of multi-agent execution trajectories.
- 3115 •  $\mathcal{M}$ : The predefined MAST taxonomy of failure types.
- 3116 •  $F = \{f_1, f_2, \dots, f_m\}$ : Set of *emerging failure mode* descriptions not covered by  $\mathcal{M}$ ,  
3117 extracted from LLM-as-a-judge evaluations.
- 3118 •  $\phi : \mathcal{S} \rightarrow \mathbb{R}^d$ : Sentence embedding function (e.g., Sentence-BERT).
- 3119 •  $\mathbf{E} = [\phi(f_1), \dots, \phi(f_m)]^\top \in \mathbb{R}^{m \times d}$ : Matrix of embedded failure descriptions.
- 3120 •  $\mathcal{C} = \{C_1, \dots, C_k\}$ : Partition of  $F$  into  $k$  clusters, each with centroid  $\mu_j$ .
- 3121
- 3122

3123 **Step 1: Emerging Failure Mode Extraction.** Each trajectory  $t_i \in T$  is evaluated by an LLM-as-  
3124 a-judge to identify:

- 3125 • Labeled failure types from the MAST taxonomy  $\mathcal{M}$ .
- 3126 • Up to two *emerging* failure descriptions  $f_{i1}, f_{i2} \notin \mathcal{M}$ .
- 3127
- 3128

3129 The full set of novel descriptions is aggregated as:

$$3130 F = \bigcup_{i=1}^n \{f_{i1}, f_{i2}\} \setminus \text{NULL}$$

3132  
 3133 **Step 2: Semantic Embedding.** Each emerging failure mode  $f_i \in F$  is transformed into a  $d$ -  
 3134 dimensional vector:

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

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

3141  
 3142 **Step 3: Optimal Clustering via K-Means.** To discover latent groups of semantically similar  
 3143 failure descriptions, we apply K-Means clustering over the embeddings  $\mathbf{E}$ . The silhouette score for  
 3144 a given point  $i$  is:

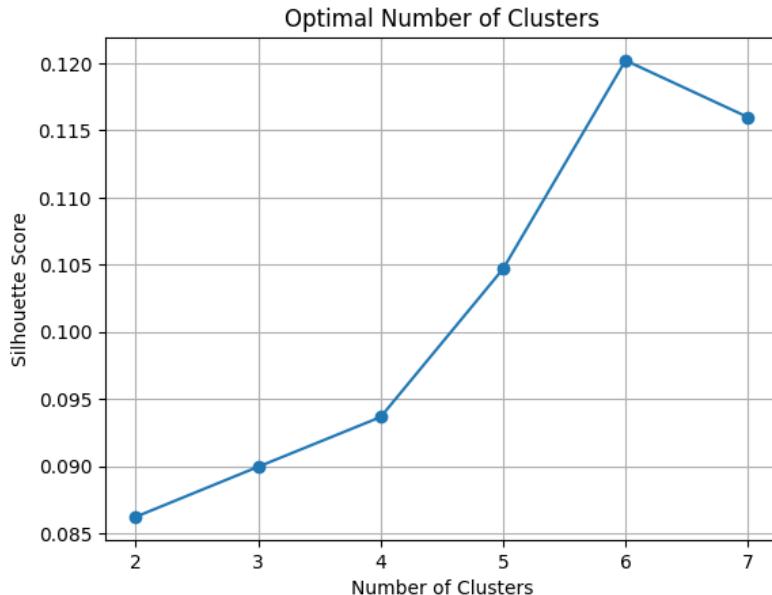
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

3145 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.

3151 The optimal number of clusters is selected as:

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



3175  
 3176 Figure 22: Silhouette analysis showing optimal number of clusters  $k^* = 6$ .  
 3177

3178 **Step 4: Cluster Center Selection.** For interpretability, we select a representative  $f_j^*$  from each  
 3179 cluster  $C_j$  as the most centrally located failure mode:

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

3183 **Step 5: Taxonomy Alignment.** Each representative failure mode  $f_j^*$  is reviewed and mapped to  
 3184 one or more MAST categories:

- Specification Failures

3186     • **Inter-Agent Failures**  
 3187     • **Task Verification Failures**  
 3188

3189 Failures that exhibit characteristics of multiple categories are marked as *compound* or *intersectional*,  
 3190 suggesting the need for extensions to the base taxonomy.  
 3191

3192 **Outputs.** The algorithm yields:

3193     • A clustered taxonomy  $\mathcal{C} = \{C_1, \dots, C_{k^*}\}$  of emerging failure modes.  
 3194     • Canonical representatives  $\{f_1^*, \dots, f_{k^*}^*\}$  for each cluster.  
 3195     • Category mappings for taxonomy refinement or extension.  
 3196     • Frequency statistics per failure type for prioritization.  
 3197

#### 3199 H.0.2 METHODOLOGY: SEMANTIC CLUSTERING OF EMERGENT FAILURES

3200 Building on the formal clustering algorithm outlined above, we implemented a practical instantiation  
 3201 of the pipeline to organize the large volume of emerging failure mode descriptions identified by  
 3202 the LLM-as-a-judge. We found lots of new and different behaviors when we first looked. But a  
 3203 closer look showed that many of them were either just repeating the same idea or were only slightly  
 3204 different versions of the same core problems. To distill these into interpretable categories, we applied  
 3205 a semantic clustering methodology grounded in high-dimensional language representations.  
 3206

3207 Each emerging failure description was manually or programmatically summarized into a concise  
 3208 label and explanatory text. These summaries were then embedded into a semantic vector space  
 3209 using the all-MiniLM-L6-v2 model from the SentenceTransformer library, yielding a set of  
 3210 dense, comparable embeddings suitable for clustering.  
 3211

3212 We applied the KMeans algorithm to group these embeddings into semantically coherent clusters.  
 3213 To determine the optimal number of clusters, we computed silhouette scores for values of  $k$  ranging  
 3214 from 2 to 7 and selected the value that maximized mean silhouette score (see Figure 22). This  
 3215 analysis yielded an optimal configuration of  $k^* = 6$  clusters.  
 3216

3217 For interpretability, each cluster was assigned a canonical label derived from the failure mode de-  
 3218 scription closest to the cluster centroid. This process produced six representative categories of  
 3219 emerging failure modes, summarized below:  
 3220

- *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.

3221 These cluster-derived failure modes serve as canonical extensions to the base MAST taxonomy,  
 3222 revealing previously unclassified behaviors that frequently arise in multi-agent LLM interactions.  
 3223 Their emergence underscores the value of inductive, embedding-based clustering for scalable failure  
 3224 mode discovery and taxonomy refinement.  
 3225

#### 3226 H.0.3 TAXONOMIC ALIGNMENT WITH MAST OF EMERGENT FAILURES

3227 These emergent failure modes reveal both alignment and tension with the original MAST taxonomy.  
 3228 Each cluster can be mapped to one or more of MAST’s three core failure categories, but many  
 3229 straddle boundaries or reveal overlapping failure dynamics:  
 3230

- **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.

## H.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 36 summarizes the results.

Table 36: Benchmark performance with and without agent communication enabled.

Model	<code>enable_agent_ask=True</code>	<code>enable_agent_ask=False</code>
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

3294 differences in agent behavior and encourages transparent reporting of configuration parameters for  
3295 reproducibility.  
3296

#### 3297 Agent Communication Insight

3298 Enabling `enable_agent_task` significantly boosts performance for models capable  
3299 of multi-turn reasoning (e.g., `llama-4-maverick`, `llama-3-405b-instruct`),  
3300 demonstrating that agent communication can resolve ambiguities and improve task execu-  
3301 tion. Other models show minimal sensitivity, highlighting differences in internal reasoning  
3302 and multi-step handling capabilities.  
3303

3304  
3305  
3306  
3307  
3308  
3309  
3310  
3311  
3312  
3313  
3314  
3315  
3316  
3317  
3318  
3319  
3320  
3321  
3322  
3323  
3324  
3325  
3326  
3327  
3328  
3329  
3330  
3331  
3332  
3333  
3334  
3335  
3336  
3337  
3338  
3339  
3340  
3341  
3342  
3343  
3344  
3345  
3346  
3347

## 3348 I WORKFLOW ILLUSTRATION

3349  
 3350 This subsection will later be merged into Appendix Section A. We are keeping it here temporarily so  
 3351 that references to figure numbers in the rebuttal do not need revision. The complexity of real-world  
 3352 problem-solving often exceeds the capabilities of a single, monolithic single LLM based AI Agent.  
 3353 To address multi-faceted challenges, especially those requiring interaction with external systems,  
 3354 specific knowledge retrieval, or sequential decision-making, advanced agentic architectures have  
 3355 been developed. This section formalizes two leading paradigms: the **Agent-As-Tool** approach,  
 3356 which focuses on dynamic, iterative reasoning and acting by calling specialized components; and  
 3357 the **Plan-Execute** framework, which emphasizes structured planning, dependency management, and  
 3358 context-aware execution to ensure traceable and reliable task completion.

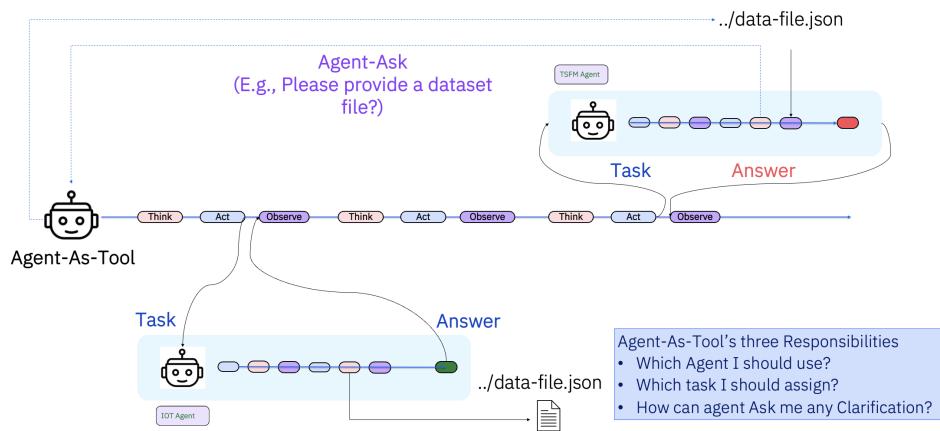


Figure 23: Typical Workflow: Agent-As-Tool using Agent-Ask

### 3376 I.1 AGENT-AS-TOOL WORKFLOW

3378 The **Agent-As-Tool** paradigm, coupled with the Agent-Ask mechanism, offers a robust framework  
 3379 for complex task execution by decomposing a problem and routing sub-tasks to specialized AI  
 3380 agents. As illustrated in the workflow Figure 23, the primary Agent-As-Tool receive a user query.  
 3381 The Agent-As-Tool has to fulfill three core responsibilities: (1) determining the appropriate special-  
 3382 ized agent (e.g., the TSFM Agent or IoT Agent) for a given sub-task, (2) correctly assigning the task,  
 3383 and (3) facilitating clarification from the specialized agent if needed. Once the Agent-As-Tool re-  
 3384 ceives an input query, it enters a standard Think-Act-Observe loop, and it decides on the appropriate  
 3385 specialized agent in think step. The specialized agent executes its designated task and provides an  
 3386 Answer. Answer is embedded inside to the orchestration agent's observation for next set of action.  
 3387 In this case, an artifact like a data-file.json, back to the main orchestrator, allowing the overall system  
 3388 to complete complex, multi-faceted operations that exceed the capability of any single agent. This  
 3389 architecture highlights the benefits of modularity and specialized expertise in large-scale language  
 3390 model systems.

3391 Algorithm 2 details an iterative, multi-turn execution framework akin to the ReAct (Reasoning and  
 3392 Acting) paradigm. This framework enables a central agent to solve a complex User Task ( $\mathcal{Q}$ ) by  
 3393 strategically engaging specialized agents ( $\mathcal{A}_{spec}$ ) within a bounded number of steps  $T_{max}$ . The  
 3394 process begins with the initialization of the agent's internal state  $M$  and an empty execution history  $\mathcal{H}$ . The  
 3395 agent then enters an iterative loop where a thinking policy  $\Pi_{think}(M)$  determines the  
 3396 next action (either THINK-ACT or FINISH) and the continuation signal. When the action is  
 3397 THINK-ACT, the agent selects the best agent.id via  $\Pi_{select}$  and formulates a precise sub\_task via  
 3398  $\Pi_{formulate}$  for execution. Conversely, if the action is FINISH, the agent summarizes the full  $\Pi$ ,  
 3399 updates the memory, and terminates the loop. Following any action, the resulting output is compressed  
 3400 into a concise observation ( $\Pi_{compress}$ ) to manage context length. This observation is logged to  $\mathcal{H}$ ,  
 3401 and critically, the agent's internal memory  $M$  is updated with this new context, driving the decision-  
 making in the subsequent round. Finally, once the loop terminates, a policy  $\Pi_{final}$  generates the  
 complete Final Output ( $O$ ) by synthesizing the entire execution history  $\mathcal{H}$ .

---

3402 **Algorithm 2:** Agent-As-Tool (Simplified as ReAct)

---

3403 **Input** : User Query  $Q \in \mathcal{Q}$ , Maximum Steps  $T_{max}$ , Set of Agents  $\mathcal{A}_{spec} \subseteq \mathcal{A}$

3404 **Output:** Final Output  $O \in \mathcal{O}$ , Execution Plan  $\Pi$

3405

3406  $M \leftarrow \text{InitializeMemory}(Q)$  // Initialize the global Memory System  $M$

3407  $\Pi \leftarrow \emptyset$ ;

3408  $\mathcal{H} \leftarrow \emptyset$ ;

3409 **for**  $t = 1$  to  $T_{max}$  **do**

3410      $(\text{continue}, \text{action}) \leftarrow \Pi_{think}(M)$  // Agent decides next step using current Memory  $M$

3411     **if**  $\text{action} = \text{THINK-ACT}$  **then**

3412          $\text{agent\_id} \leftarrow \Pi_{select}(\text{action}, M, \mathcal{A}_{spec})$  // Select best Agent  $A_i \in \mathcal{A}_{spec}$

3413          $\tau \leftarrow \Pi_{formulate}(\text{action}, M)$  // Formulate sub-task  $\tau \in \mathcal{T}$  based on current Memory  $M$

3414          $\text{output} \leftarrow \text{ExecuteAgent}(\text{agent\_id}, \tau)$  // Agent executes task and returns structured output  $o \in \mathcal{O}$

3415          $\Pi \leftarrow \Pi \cup \{(\tau, \text{agent\_id})\}$  // Update the execution plan  $\Pi$  with task-agent assignment

3416     **else if**  $\text{action} = \text{FINISH}$  **then**

3417          $\text{final\_output} \leftarrow \text{Summarize}(\Pi)$ ;

3418          $M \leftarrow \text{UpdateMemory}(M, \text{final\_output})$ ;

3419         **break**;

3420      $\text{observation} \leftarrow \Pi_{compress}(\text{output})$  // Generate concise observation summary (optional)

3421      $\mathcal{H} \leftarrow \mathcal{H} \cup \{(t, \text{action}, \text{observation})\}$  // Log step to history

3422      $M \leftarrow \text{UpdateMemory}(M, \text{observation})$  // Update Memory  $M$  with new context from observation

3423     **if**  $\text{continue} = \text{False}$  **then**

3424         **break** // Stop if agent decides termination

3425      $O \leftarrow \Pi_{final}(\mathcal{H})$  // Produce final answer from the complete history

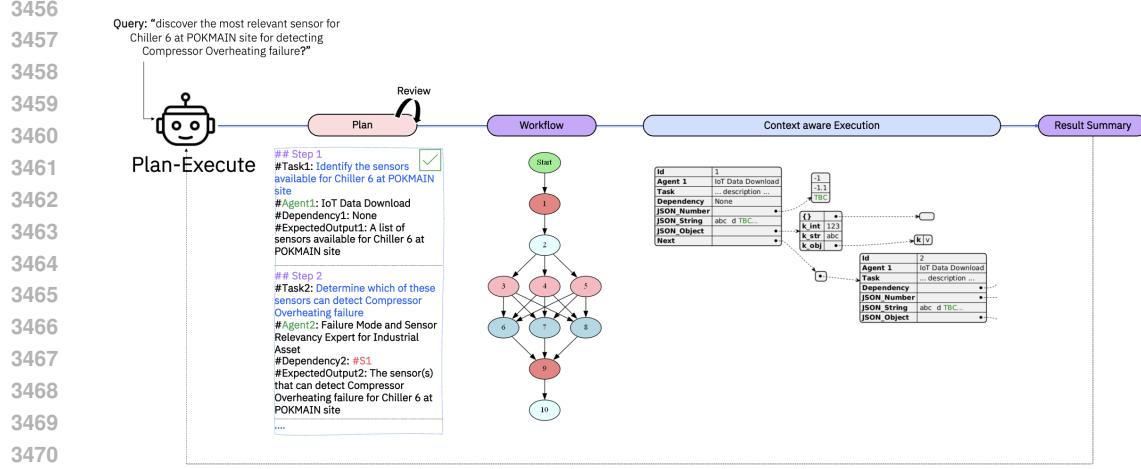
3426 **return**  $(O, \Pi, \mathcal{H})$

---

## 1.2 PLAN-EXECUTE

A diagram in Figure 9 illustrates a **Plan-Execute** approach to addressing a complex industrial query, such as “discover the most relevant sensor for Chiller 6 at POKMAIN site for detecting Compressor Overheating failure?”. The process begins with the main agent receiving the Query and formulating a detailed Plan. This plan is meticulously broken down into sequential steps. For instance, Step 1 involves a Task to “Identify the sensors available for Chiller 6 at POKMAIN site” and specifies Agent 1 (e.g., an IoT Data Download Agent) to execute this task, with an Expected Output of “A list of sensors available for Chiller 6 at POKMAIN site.” Following this, Step 2 takes this output as a Dependency (#S1) to execute the Task: “Determine which of these sensors can detect Compressor Overheating failure,” assigning it to a specialized Agent 2 (e.g., a FMSR Agent).

After the Plan is reviewed, it is translated into a dynamic, dependency-aware Workflow represented as a directed graph. This graph outlines the logical flow and potential parallel execution paths (e.g., tasks 3, 4, and 5 running concurrently) based on the sequential nature of the task dependencies. The Context-aware Execution phase then involves a specialized execution engine that manages these tasks, tracking their state, inputs (like the JSON objects and strings containing the intermediate results), and dependencies between agents. For example, the output of the first stage (ID 1) becomes a structured input for subsequent tasks (ID 2), ensuring that information is seamlessly and accurately passed between the specialized agents. The entire process culminates in a Result Summary that provides the final, actionable answer to the initial complex query. This methodology ensures traceability, modularity, and the effective integration of multiple specialized AI agents for industrial problem-solving.



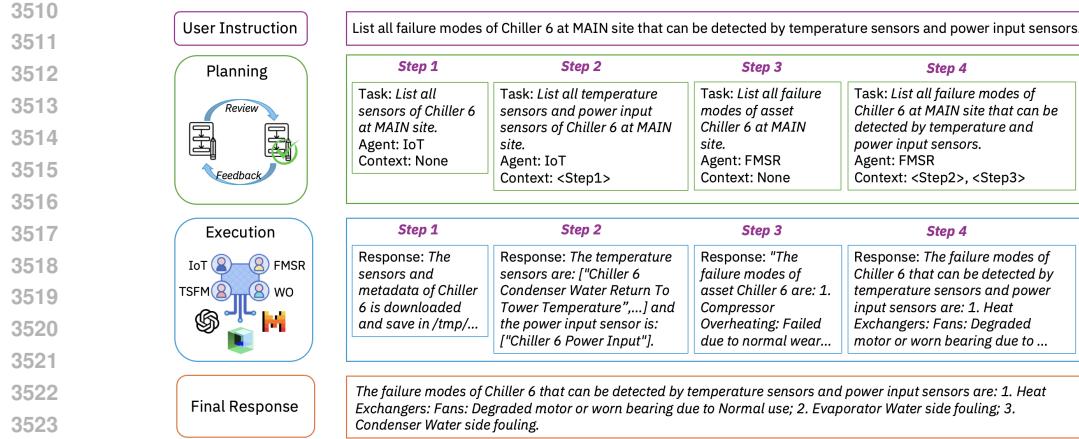


Figure 25: Plan-Execution Workflow Concrete Example.

Algorithm 3 describes a structured, three-phase approach for complex queries, prioritizing explicit planning and efficient data flow management. This approach ensures a systematic resolution of the User Query ( $\mathcal{Q}$ ). The first phase, **Planning and Decomposition**, transforms  $\mathcal{Q}$  into an execution structure. A planning policy  $\Pi_{plan}$  generates a sequential Plan, which is then compiled into a Directed Acyclic Graph (Workflow) defining task dependencies. The Context is initialized, and a TaskSequence is generated by topologically sorting the Workflow, which guarantees that tasks are processed only after their dependencies have been met. The second phase, **Context-Aware Execution**, manages the task flow based on the TaskSequence. For each task in the sequence, the algorithm retrieves *\*only\** the necessary context. This process involves iterating through all task dependencies to collect the required data from the global Context. This collected data is then formatted and combined with the task.description to form the FullQuery, which is executed by the assigned agent. The resulting RawOutput is immediately standardized via FormatToJSON and stored back into the global Context (e.g., as #S1), making it available for subsequent tasks. The final phase, **Summarization**, occurs upon completion of all tasks. A final summarization policy ( $\Pi_{summary}$ ) synthesizes the definitive Final Output ( $O$ ) from the original  $\mathcal{Q}$  and the comprehensive Context.

### I.3 DETAILED WORKFLOW EXAMPLES

Figure 25 presents a concrete example of the Plan-Execute workflow for the user query: “List all failure modes of Chiller 6 at the MAIN site that can be detected by temperature sensors and power-input sensors.” The planning stage produces four steps, illustrated in the middle row, each outlining a specific sub-task derived from the original query. The execution stage then follows these steps in sequence, generating intermediate outputs and ultimately producing the final answer. This example highlights how the Plan-Execute approach breaks down a complex request into structured actions and systematically retrieves the required information.

## J MODEL PERFORMANCE AND PLANNING ANALYSIS

We extended our evaluation to include models that were not part of the original benchmark, specifically Anthropic Claude variants (claude-3-7-sonnet, claude-4-sonnet) and GCP Gemini (gemini-2.5-pro). Model performance was evaluated using planning accuracy metrics (BERTScore, ROUGE, and alignment with ground-truth plans), consistent with our original execution-accuracy leaderboard. We also analyzed planned step statistics to understand model behavior in generating task plans.

3564 J.1 COMBINED SCORE (BERTSCORE + ROUGE-L)  
35653566 For each model, we compute a *combined planning score*  $S_{\text{combined}}$  that integrates BERTScore ( $B$ )  
3567 and ROUGE-L ( $R$ ):

3568 
$$S_{\text{combined}} = \frac{B + R}{2},$$
  
3569

3570 where  $B \in [0, 1]$  is the average BERTScore between the model-generated plan and the ground-truth  
3571 plan, and  $R \in [0, 1]$  is the average ROUGE-L F1 score. This provides a balanced measure of both  
3572 semantic similarity (via BERTScore) and sequence-level overlap (via ROUGE-L).3573 J.2 PLANNING ACCURACY  
3574

Model Name	Avg $\pm$ Std	Questions
mistral-medium-2505	$0.620 \pm 0.063$	141
gemini-2.5-pro	$0.615 \pm 0.068$	141
gpt-oss-120b	$0.606 \pm 0.077$	141
mistral-small-3-1-24b	$0.604 \pm 0.062$	141
claude-4-sonnet	$0.595 \pm 0.068$	141
llama-3-405b-instruct	$0.588 \pm 0.074$	141
claude-3-7-sonnet	$0.571 \pm 0.071$	141
llama-4-maverick-17b	$0.558 \pm 0.071$	141
gpt-5-2025-08-07	$0.544 \pm 0.092$	141
granite-3-3-8b-instruct	$0.529 \pm 0.067$	141
llama-3-3-70b-instruct	$0.522 \pm 0.068$	141

3587 Table 37: Planning accuracy (Avg  $\pm$  Std) for all evaluated models.  
35883589 J.3 PLANNED STEP STATISTICS  
3590

Model Name	Avg Steps $\pm$ Std	Min	Max	Zero Steps
llama-3-405b-instruct	$3.14 \pm 1.84$	1	9	0
llama-3-3-70b-instruct	$6.55 \pm 1.55$	3	12	0
llama-4-maverick-17b	$4.34 \pm 1.80$	1	9	0
granite-3-3-8b-instruct	$5.56 \pm 2.44$	2	30	0
gpt-oss-120b	$1.91 \pm 1.21$	1	10	0
mistral-medium-2505	$2.38 \pm 1.04$	1	5	0
mistral-small-3-1-24b	$2.77 \pm 1.33$	1	6	0
claude-3-7-sonnet	$3.10 \pm 1.15$	1	5	0
gpt-5-2025-08-07	$2.33 \pm 1.16$	0	5	1
gemini-2.5-pro	$1.87 \pm 1.01$	1	5	0
claude-4-sonnet	$2.45 \pm 1.34$	1	5	0

3604 Table 38: Planned step statistics for all evaluated models.  
36053606 J.4 KEY INSIGHTS  
3607

- **Top-performing models:** `mistral-medium-2505` achieves the highest planning accuracy ( $0.620 \pm 0.063$ ) and produces concise, low-variance plans ( $2.38 \pm 1.04$  steps), combining high performance with efficiency. `gemini-2.5-pro` is also highly competitive ( $0.615 \pm 0.068$ ).
- **Anthropic Claude models:** Both `claude-3-7-sonnet` and `claude-4-sonnet` show solid planning accuracy with moderate plan lengths (2.5–3 steps) and low variance, indicating reliable reasoning and execution alignment.
- **Instruction tuning matters:** Medium-sized instruction-tuned models (`mistral-medium`, `mistral-small`) consistently produce efficient plans with low variance, outperforming larger models with longer, more variable plans (`llama-3-3-70b`, `granite-3-3-8b`).

3618  
3619  
3620  
3621  
3622  
3623  
3624  
3625  
3626  
3627  
3628  
3629

- **Step length vs. performance:** Shorter plans with low variance generally correlate with higher planning accuracy, while overly long plans may introduce redundancy without improving alignment with ground-truth executions.

- **Consistency vs. variability:** High-variance models (gpt-5-2025-08-07, granite-3-3-8b) occasionally generate very long or empty plans, which may reduce reliability despite moderate average scores.

3630  
3631  
3632  
3633  
3634  
3635  
3636  
3637  
3638  
3639  
3640  
3641  
3642  
3643  
3644  
3645  
3646  
3647  
3648  
3649  
3650  
3651  
3652  
3653  
3654  
3655  
3656  
3657  
3658  
3659  
3660  
3661  
3662  
3663  
3664  
3665  
3666  
3667  
3668  
3669  
3670  
3671

**Conclusion:** The unified analysis demonstrates that medium-sized, instruction-tuned models offer the best balance of planning accuracy and step efficiency, while the inclusion of Claude and Gemini models extends benchmark coverage and validates performance trends. These results are consistent with our original execution-accuracy leaderboard, confirming the robustness of the benchmark for evaluating reasoning-capable language models.