Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Multi-Agentic Digital Twin Framework for Human-in-the-loop Workflows

Maqbool Hussain

m.hussain@derby.ac.uk

University of Derby

Gayan Dihantha Kuruppu Kuruppu Appuhamilage

University of Derby

Wajahat Ali Khan

University of Derby

Farid Meziane

University of Derby

Ben Anderson

AddQual Ltd

Article

Keywords:

Posted Date: November 2nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10386014/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: There is **NO** Competing Interest.

A Multi-Agentic Digital Twin Framework for Human-in-the-loop Workflows

Gayan Dihantha Kuruppu Kuruppu Appuhamilage[1],
Maqbool Hussain[1,*], Wajahat Ali Khan[1], Farid Meziane[1], Ben Anderson[2]

[1] School of Computing, College of Science and Engineering, University of Derby, Kedleston Road, Derby, DE22 1GB, United Kingdom
[2] AddQual Ltd, No. 27, Dunstall Park, Dunstall Park Road, Derby DE24 8HJ, United Kingdom

* Corresponding Author: Maqbool Hussain (m.hussain@derby.ac.uk)

Abstract

Traditional robot task planning struggles to recover from failures and changing requirements because it relies on predefined rules, labelled datasets, and limited state awareness. Digital twins provide real-time virtual representations of physical systems, while multi-agent systems support collaborative, stepwise reasoning. However, their integration for failure evaluation, requirement adaptation, and state-aware robot planning remains underexplored, particularly at the architectural level. This study proposes a multi-agentic digital twin framework for human-in-the-loop workflows, enabling bidirectional communication between machines, digital twins, and multi-agents. The framework is validated through robot task planning for an aerospace product-qualification metrology process. Results show that digital-twin integration adds 885-1,149 milliseconds to communication between robots and agents because continuous state updates are required. Multi-agent workflows need more human feedback than single-agent workflows to align outputs with predefined goals and mitigate hallucinations, highlighting the importance of human-in-the-loop orchestration. Datasets, logs, code, videos, results, and deployment instructions are available on GitHub[1].

Main

A digital twin is a virtual model that uses real-time data and simulation tools to enable data-driven governance and informed decision-making for assets or processes[2]. The digital twin concept originally emerged from aerospace and manufacturing sectors, where it is used as a virtual counterpart that mirrored behaviour of a physical system in real time[3,4]. Later, the concept expanded into non-mechanical domains such as medicine[5], city planning[6], and industry[7] . The growing popularity of digital twins has led researchers to explore a multitude of technical approaches for simulating physical systems. However, this has also brought attention to methodologies that are more accurately described as digital shadows or digital models rather than true digital twins. Many of these are merely closed-loop systems that lack feedback mechanisms, adaptive learning capabilities or continuous real-time bidirectional data streams and control structures capable of influencing behaviour of their associated physical twins[8]. The key proposition that distinguishes a digital twin from traditional modelling or simulation is its real-time

bidirectional data stream, which both reflects and influences behaviour of the physical and virtual counterparts[2,9,10].

In recent years, digital twins have advanced as a coordinating approach for demonstrating autonomous systems that integrate data, models, and human interactions[8]. This coupling of continuous learning cycles involving data, models, and intelligence enhances predictive capabilities and adaptive control across physical and virtual systems[8]. A growing trend is for digital twins to become agentic with cognitive abilities and extended autonomous interactions with their environment[11,12]. Large language models are increasingly being employed to play a key role in enabling perceptual and cognitive capabilities, along with improved contextual understanding and interoperability between interconnected systems[8]. In addition, the ability of large language models to comprehend human language and interpret non-uniform data in ways that translate human intent into control objectives enhances operability without requiring extensive operational proficiency[8].

Recently, large language models have begun shifting toward a multi-agent approach, in which multiple AI agents handle stepwise problem-solving through tool use and collaboration[13]. For instance, scientists have discovered new antibodies for SARS-CoV-22 virus strain by using a team of AI agents[14]. Despite their potential to contribute to scientific discovery, agents are often susceptible to prompt fragility, where subtle changes in input can produce heterogeneous outputs. They can also hallucinate, generating mistakes or fabricated facts, which makes human involvement in decision-making process necessary[13]. Research on agentic systems that collaborate with humans on organisational tasks such as planning remains limited.[15]. Therefore, an approach is needed to integrate digital twins with AI agents in order to ground reasoning in real-world factors, leverage precise data streams, and apply physical constraints where necessary[16].

The proposed framework evaluated multi-agent aspects of digital twins using a robot task-planning use case in aerospace product qualification. Task planning with multi-agent systems is a growing area of research interest because, reasoning and collaboration capabilities of multiple agents can enable more adaptive and efficient solutions to task-planning problems[17]. At the same time, executing tasks efficiently while integrating external domain-specific knowledge remains a key challenge for multi-agent systems[17]. The progression toward agentic digital twins has opened up innovative possibilities by unifying cognitive capabilities of AI agents and enabling adaptation through digital twins in energy infrastructure[18]. The authors focus on a multi-agent digital twin approach for robot task planning that incorporates external data sources and semi-autonomous adoption to changing factors in physical environment during execution.

Traditional robot task-planning methodologies often struggle because robots cannot adapt to unpredictable situations in the real world [19]. In dynamic settings robots need to adapt to changing task specifications despite having only an incomplete set of preconfigured instructions[20]. It is generally necessary for robots to understand states of their physical twins in order to achieve successful outcomes. Existing symbolic STRIPS-based or learning-based methodologies are limited because they rely on carefully predefined values and labelled datasets which are often ineffective in dynamic settings where robots must continuously transform plans to adapt to evolving environmental conditions[21]. However, without proper feedback mechanisms to determine whether a current action has succeeded or failed outcome of subsequent actions may be compromised[22]. Because large language models are trained on vast corpora of real-world knowledge, they have potential to be used in robot task planning to formulate stepwise task sequences in natural language. However it is reasonable to assume that this approach may generate actions that are infeasible in a given context[23]. The incorporation of automated robot task

planning therefore poses a considerable research challenge, as robot task plans must be grounded in contextual awareness, adaptive behaviour, and uncertainty-aware reasoning in dynamic settings[20].

For these reasons, recent literature suggests a growing trend toward integrating large language models with robotic systems to map natural language instructions into robot action sequences[24], enable advanced reasoning using prior experiences[25], and extend reasoning capabilities beyond task planning and interactions[26]. Although large language models help improve generalisation and natural human-robot interaction, they still face limitations in practicality and reliability[27]. Hallucinations in large language models can cause misalignment with real-world constraints or inconsistencies with the given inputs in robot task planning[28]. Their applicability to robot task-planning tasks without incorporating human assessment of outcomes is also limited, as they often misinterpret simulated data or real-world inputs[29]. For these reasons, incorporating a human-in-the-loop into robot task-planning process is crucial for achieving practical outcomes.

Another significant challenge is grounding large language models to enable robots to interact with their environment and continuously learn from updated state changes during execution[30,31]. Although large language models are trained on vast corpora of human textual knowledge, they still remain insufficiently grounded in physical contexts[32]. The challenge of robotic grounding lies in supporting generated robot tasks with real-world operational context[33]. Previous research has experimented with embedding external tools for reasoning and interaction within physical contexts but it still lacks a scalable and unified framework[31]. Beyond physical context alone, a framework's ability to identify and reason about robot failures is also a vital factor in achieving robust and explainable robotic behaviours[34]. In contrast existing large language model-based approaches often fail to address the root causes of failure and instead repeatedly attempt same action without making inferences needed to recover from execution errors[35]. Moreover, generating plans under assumption that all proposed steps will be executed successfully limits robustness to intermediate failures that arise in dynamic environments or from unreliable low-level control policies[26]. Therefore, The proposed framework examined these failures and application of human-in-the-loop approach to incorporate human inference for mitigating them within our framework.

Beyond the grounding aspects of large language models, a key challenge for intelligent robot systems lies in assessing prospects for extracting and applying human knowledge and reasoning to robot task planning[35,36]. Because large language models are trained on extensive corpora of human knowledge, they can capture and interpret semantic aspects of natural language instructions. However, they often fail to perform context-aware reasoning for effective task planning because they lack experience in targeted contexts[37,38]. Large language models also do not consistently outperform specialised models or tools on their respective downstream tasks[29]. Designing suitable interfaces that can effectively collect on-robot data to augment reasoning capabilities also remains a challenge[39]. Nonetheless, large language models often face limitations in integrating manipulation capabilities through external tools[24]. Existing research has increasingly incorporated specialised modules alongside large language models to improve reliability by guiding outputs through modules for arithmetic reasoning[40], explicit task planning[29], feedback from the operational environment[26], and API calls to external tools[40]. It is therefore important to recognise that human inference is crucial for reducing negative effects of these reasoning limitations and for maximising the reliability of external tools.

To the best of our knowledge research on AI-driven multi-agentic digital twin approaches for robot task planning remains limited. Recent literature in the construction[41–43], manufacturing[44,45] mining[46] and energy[16] domains has introduced agent-based digital twin approaches. Table 1 summarises the existing

agentic digital twin studies, their key limitations, and how the present study addresses these research gaps.

Table 1: Summary of related agentic digital twin studies

Study	Domain	Proposed Approach	Key Limitations	How the proposed framework addresses the limitations?
Hasan and Nguyen et al[16]	Energy	Integration of agentic AI and digital twins for cognitive reasoning, continuous learning, and electrical-grid balancing.	Identified challenges in secure agent digital twin communication and effective human-AI collaboration.	Uses a secure Model Context Protocol for connectivity and HITL workflows to mitigate hallucinations and reduce the risk of execution faults.
Li et al[41]		Building digital twin combined with knowledge-engineering methods for model development, diagnosis, and lifecycle calibration.	Focused mainly on bias faults. Validated using a limited number of HVAC variables. Scalability and generalisability of AI agents were left as future work.	Integrates the required telemetry and command features of physical twins to represent the real-time state of an aerospace product-qualification process. Uses a layered architecture to support scalable and generalisable AI multi-agent systems.
Shen et al[42]		Agent-based architecture for continuously managing the digital twin self-model of a three-storey building through human-agent collaboration.	Offline-focused implementation. No real-time integration with physical-twin sensors. Limited description of action execution. Sequential workflows only partially transfer information between stages.	Enables real-time telemetry ingestion and command execution through digital twins. Uses HITL workflows to preserve context and reduce the limitations of sequential processing.
Lee et al[43]		Multi-agent approach for autonomously creating and calibrating virtual building models to extend digital twins.	Limited to virtual HVAC modelling. Lacks an adequate mechanism for integrating heterogeneous data sources.	Focuses on robot task planning and uses model context protocol to integrate structured information from heterogeneous sources and coordinate interactions with digital twins.
Liu et al[44]	Manufacturing	Digital twin-assisted human-robot collaborative assembly framework using vision-language-model-based agents.	Primarily addresses object recognition, grasping strategies, object handling, and human-robot collaboration rather than comprehensive robot task planning.	Provides comprehensive robot task planning grounded in real-time physical-twin states and coordinated through a multi-agent digital twin framework.
Marah & Challenger[45]		Integrates multi-agent systems with digital twins to enable intelligent, autonomous, and cooperative behaviour among multiple robots.	Presents a JADE-based agent architecture, metamodel, methodology and ontology, using predefined behaviours, limited advanced reasoning, no central HITL orchestration, and proof-of-concept validation.	Presents a layered LLM-driven multi-agent digital twin with explicit HITL orchestration, adaptive planning, failure handling, and quantitative evaluation of latency and workflow failures.
Collao et al[46]	Mining	Multi-agent and digital twin approach for anomalous-behaviour detection, predictive maintenance, and quality assurance in a ball mill.	Domain-specific implementation. Limited system integration and real-time response capabilities.	Supports integrated, real-time robot task planning and responses based on continuously updated real-world states.

Existing literature on knowledge acquisition for task planning has explored approaches such as human-robot interaction[47], responsive knowledge bases[48], and incorporation of commonsense reasoning[37,38,49] through natural language[19]. Iterative prompting strategies that incorporate context based on large

language model outputs have demonstrated substantial effectiveness by leveraging feedback from repeated action synthesis[38], environment-mediated responses[26], environment representations[50], perception-based object recognition[26,51], invalid actions[37,38,52], unsatisfied preconditions[29,35], execution-time code failures[50], correction of syntactic inconsistencies in output[24,53], and interaction with environment[54]. Mitigating latency is essential for highly dynamic tasks, given that large language models are typically hosted in cloud[39].

Agentic digital twins have their own intricacies including the extent to which a digital twin is aware of its underlying data inputs as well as issues related to computational sustainability, interoperability, trust assessment, and cybersecurity threats[8]. Agentic digital twins still exhibit significant gaps in the following areas: (a) architectural representation, particularly in how agentic AI systems interact with digital twin frameworks; (b) cognitive reasoning, including adaptive reasoning, agent-to-agent communication, and real-time learning capabilities; and (c) convergence of limited adaptive reasoning in digital twins with lack of physical grounding and real-time synchronisation with physical twins in AI agents[16,55]. Furthermore (d) integration of multi-agent systems with digital twins remains an active area of research as infusion of cognitive capabilities through large language models to enhance task execution continues to pose a significant challenge[18]. Also (e) establishing a coherent link between robots and large language models also remains an unresolved research challenge[39].

To address these gaps, this study proposes a multi-agentic digital twin framework. The architectural representation gap (a) is resolved through a cloud-based approach that enables seamless integration between digital twins and multi-agent systems avoiding edge-based solutions to reduce latency between physical twins and multi-agents. The convergence and synchronisation challenges (c) are addressed by establishing a two-way communication mechanism between physical twins and multi-agents via the Internet of Things allowing multi-agents to learn and interpret the physical state before executing commands through digital twins. Furthermore, the integration of multi-agent systems with digital twins (d) is enhanced through use of Model Context Protocol (MCP) which provides a natural language interface to frequently read states and write states execute commands on digital twins. Finally, the challenge of linking robots with large language models (e) is explored through a layered architecture where robots and sensors are interfaced with multi-agents via digital twins enabling more coherent interaction across physical and cognitive systems.

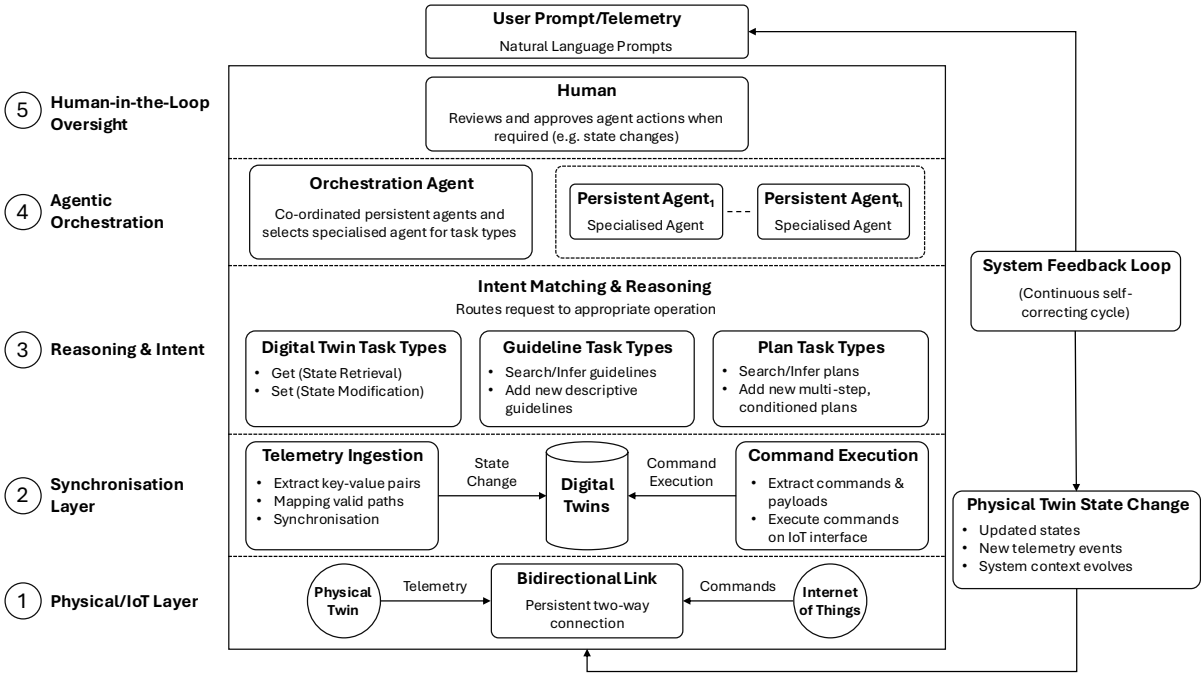


Figure 1: Conceptual architecture.

This figure illustrates conceptual architecture for an agentic digital twin system. At the base, the Physical Twin / Internet of Things, and Bidirectional Link enable *real-world sensing*, *connectivity*, and *persistent two-way communication*. The second layer handles *telemetry ingestion*, *digital twin representation*, *command execution*, and *physical state changes*. The third layer defines task categories, including *state retrieval/modification*, *guideline search and generation*, and *plan inference or creation*. The fourth layer introduces an orchestration agent that coordinates persistent specialized agents and assigns tasks. At the top, human oversight reviews and approves actions when needed, while user prompts and telemetry provide natural language input. A separate system feedback loop supports continuous self-correction as the system context evolves.

The proposed framework proposes a two-way communication between machines and multi-agents using digital twins to enable semi-autonomous robot task planning within human-in-the-loop workflows. The Figure 1 illustrates a conceptual framework in which human experts such as domain specialists and operators collaborate with AI agents to enable effective monitoring, analysis, and decision-making. The framework supports real-time interaction with dynamic real-world environments allowing systems to remain grounded and context aware. Through continuous feedback and adaptation, system optimises performance by aligning with predefined goals, mitigating hallucinations, and correcting errors. Human expertise is integrated into the loop to guide AI behaviour, enhance reliability, and ensure informed decision-making. This collaborative approach facilitates development of intelligent, adaptive, and trustworthy smart systems. Present work then implemented this generalised conceptual framework with consideration for architecture, security, scalability, reusability, reliability, networking, DevOps, monitoring, and cost to assess in robot task planning aerospace product qualification use case.

The methodology integrated actuators such as 6 sensors, 1 Universal Robot's Cobots, 1 GOM Metrology devices and 1 Robotiq's Mechanical Gripper devices to describe 14 physical twins. The sensors are connected to a cobot, reducing number of IoT devices to 3. The digital twins (dt = 14) not only represent 9 physical twins but also include 2 loading and 2 unloading positions as well as entire scenario which is modelled with 1 factory-level digital twin. Additionally, 30 REST API endpoints provided connectivity to databases and digital twins. Along with 32 MCP endpoints to enable a natural language interface for interacting with physical systems. The framework retrieves states from 48 telemetry features and sets states through 16 command features in physical twins while establishing cloud-to-device communication using Internet of Things. These states are then continuously updated,

and command requests are continuously monitored in a digital twin instance. Finally, this research evaluated preconfigured prompts to interact with the physical twins. The source code and setup instructions for this end-to-end framework are available in our GitHub repository[1] as noted in the Code availability section. This framework implemented three single agents: (a) *digital twin agent*, (b) *guideline agent*, and (c) *plan agent*. Each agent is connected to its own MCP tools which provides necessary natural language interfaces to perform specific tasks. The "*digital twin agent*" has capability to read and write states to digital twins. The "*guideline agent*" can read and write guidelines which are a structured data type containing a title and content that *plan agent* uses during planning. The "*plan agent*" can read and write plans which are a structured data type containing a conditional stepwise array of required states based on digital twin states. Individually these agents cannot interact with one another. To enable multi-agent collaboration, the framework utilised one orchestration agent that uses single agents as connected agents. The orchestration agent can perform multi-agent tasks such as reading a guideline and creating a plan or executing a plan using digital twins. Current work experimented with each single agent individually with a single agent combined with the orchestration agent and with all single agents coordinated through orchestration agent.

The authors validated the proposed framework using product qualification scenarios based on a metrology use case in the aerospace domain. This study evaluated the framework in both simulated and real-world settings. However, due to experimental nature of the framework authors conducted multi-agentic experiments extensively in simulation. Then authors evaluated state retrieval capabilities of the digital twin aspects of the framework using real collaborative robots and sensors. This research conducted experiments in three categories: (a) agent-to-system experiments which used only a single AI agent to communicate with the system; (b) orchestration-agent-to-system experiments which used an orchestration AI agent to communicate with its connected agents within a specific system; and (c) human-in-the-workflow experiments which used an orchestration AI agent to coordinate communication among multiple AI agents with the support of human inference. This work designed and evaluated these experiments to compare the performance of single AI agents and multi-agent workflows, assess computational costs and limitations involved, and determine the extent to which human inference could affect latency and accuracy of automated workflows[13].

Main contributions by this study are: (a) a novel architectural representation of a digital twin-in-the-middle framework that enables two-way communication between machines and multi-agents, and vice versa; (b) physical grounding and real-time harmonisation of AI models with physical twins; and (c) a comparison of automated and human-in-the-loop approaches using success rate, human-in-the-loop rate, latency, token usage, request usage, and system availability as evaluation metrics.

Results

The proposed framework evaluated across multi-agent, MCP, digital twin, Internet of Things, and physical twin aspects using a number of evaluation metrics including success rate, human-in-the-loop rate, latency, token usage, request usage, and system availability for each experimental category based on agent-to-system, orchestration-agent-to-system, and human-in-the-loop workflows.

Multi-Agent Aspects

The experiments are carried out using predefined prompts designed to perform product qualification through metrology in the aerospace domain. These predefined prompts are designed to evaluate context window consumption and organised contextual instructions. Prompts designed to evaluate context

window consumption describes technical aspects such as 120,00 available tokens per context window and maximum number of 50,000 tokens per minute. On the other hand prompts designed using organised contextual instructions describes prompts contain relevant information including targeted agents and associated MCP calls, frequency of use of agents and associated MCP calls within a single prompt, expected outcomes or state changes, possible existing inputs or states to be validated, and availability of such inputs or states. Prompts are then structured according to three aspects: format, length, and rule.

Format aspect included structured and unstructured formats. Structured formats explicitly described how to perform a task using the exact nomenclature required to identify agents, telemetry, or command features along with the necessary parameters. Authors adopted this structured format to reduce hallucination and help framework identify required functionality from associated agent that provides specific MCP tools needed either to determine its state or to modify it using required parameters. In contrast, unstructured format did not explicitly use nomenclature to identify agents, telemetry, command features, or required parameters. Authors used these unstructured prompts as a stress test to evaluate how system determines which agent is responsible for managing a specific type of task in multi-agent experiments.

Length aspect included short and long lengths. Length is determined based on total number of prompts used in each experiment. This does not include the length of a single prompt. An experiment with seven or more prompts are considered a long experiment or otherwise a short experiment. As mentioned in Figure 2, the proposed framework frequently observed token limit issues in orchestration-agent-to-system experiments, as *orchestration agent* consumed more tokens while processing prompts. For this reason, this work applied length aspect to observe this exception patterns. To address this, the framework should incorporate model-agnostic token budgeting, prompt-rate throttling, and validation mechanisms to reduce token-limit failures and hallucinations while making clear that no multi-agent framework is completely free from technical constraints.

Rule aspects included conditional and unconditional rules. Conditional rules require system to make a decision based on outputs from a previous agent or from a system state, and to evaluate that condition within a single prompt. These conditions could involve checking whether a state is true or false, or whether a value is greater than or less than an expected threshold. On the contrary, unconditional rules did not include any conditions and each prompt is designed to perform a specific isolated task. Authors used conditional rule as a stress test to evaluate whether it could handle longer more token-intensive prompts. the proposed framework observed conditional rule based prompts consumed more tokens than unconditional rule based prompts.

As shown in Table 2, authors designed combinations of prompt structures to evaluate context windows consumption and organised contextual instruction aspects. Authors then determined number of prompts per experiment, organisation of prompt structures within each experiment, and number of experiments required to validate agent-to-system, orchestration-agent-to-system, and human-in-the-loop workflow behaviours within framework.

Table 2: Prompt design.

#	Prompt Structure			Evaluation	
	Format	**Length**	**Rule**	**Context Window Consumption**	**Organised & Context-rich Instructions**

	Structured Prompt	Unstructured Prompt	Short Prompt	Long Prompt	Conditional Prompt	Unconditional Prompt	Small	Large	Poor	Rich
SSC	✓		✓		✓			✓		✓
SSU	✓		✓			✓	✓			✓
SLC	✓			✓	✓			✓		✓
SLU	✓			✓		✓		✓		✓
USC		✓	✓		✓			✓		✓
USU		✓	✓			✓	✓		✓	
ULC		✓		✓	✓			✓		✓
ULU		✓		✓		✓	✓		✓	

During evaluation, authors designed each prompt to target specific tasks in single-agent and multi-agent evaluation. The proposed framework categorised these tasks into *digital twinning tasks*, *guideline tasks*, *planning tasks*, and *not existing tasks* task types. *Digital twinning tasks* task type involved reading and modifying digital twin states on digital twins. *Guideline tasks* task type involved creating a structured data store to maintain guidelines, including a title and description, for agents to follow when creating plans. *Planning tasks* task type targeted creation, updating, and deletion of plans, as well as steps, conditions, and their parameters. *Not existing tasks* task type described a stock exchange portfolio management scenario, which is not included in this framework. This is used to observe possible hallucinations where an agent might attempt to associate task type with an available agent. Supplementary Table 1 shows conceptual data types for guideline and plan data structures.

In single-agent evaluations, the proposed framework experimented each task type individually using specialised agents. For example, this study evaluated *digital twin tasks* task type with *digital twin agent*. The proposed framework did not use *orchestration agent* as an intermediary for these experiments in order to compare performance between single-agent and multi-agent scenarios. This work only covered 75% of the scenarios, analysing 12 out of the 16 possible total scenarios due to overwhelming number of experimental variations. Authors recognise this as a limitation in our study and identify it as an area for future work. Supplementary Table 2 below shows example prompts. The proposed framework did not include length aspect in prompts shown below as it describes number of prompts in an experiment rather than an individual prompt.

For multi-agent evaluations, the proposed framework experimented each task type using *orchestration agent* as an intermediary. These prompts are modified to identify the specific agent at the beginning of structured prompts, but not in unstructured prompts. Authors used this set of prompts to analyse both orchestration-agent-to-system experiments and human-in-the-loop workflows. The proposed framework did not cover *not existing tasks task* type and only covered 56% of possible combinations, analysing 9 out of 16 scenarios. Due to overwhelming number of experimental variations mentioned previously, this work recognise this as a limitation of in our study and identify it as an area for future work. Supplementary Table 3 below does not discuss length, as length aspect only affects number of prompts in an experiment, not length of prompt in terms of number of tokens.

For 153 agent-to-system experiments, authors evaluated 3831 prompts on each agent to assess its capabilities. For an example, using *digital twin agent* authors evaluated digital twinning tasks as well as other tasks such as guideline tasks or planning tasks using *digital twin agent* and handling requests involving non-existent agents such as *stock market agent*. The proposed framework designed these

prompts so that they could later serve as templates for other experimental categories to evaluate outcomes based on multi-agent interactions. Initially, authors observed that each agent responded only according to its own capabilities and not to other functions. When a capability is unavailable, each agent specifically stated that it did not have access to relevant MCP definitions. For an example, when the proposed framework evaluated *digital twin agent* with not existing tasks and *digital twin agent* responded with feedback such as: "*It seems there might be a misunderstanding. This agent is specifically designed to work with digital twin management tools and does not have capabilities related to stock exchange operations, such as creating a portfolio or managing trading strategies. If you have any questions or tasks related to digital twins, feel free to ask!*" or similar messages. This research then analysed each digital twin to identify significant deviations from expected outcomes when using structured and unstructured formatted prompts. Structured format and conditional rule prompts, significantly reduced hallucinations. This work did not observe any effect of conditional rule prompts in agent-to-system experiments on evaluating organised context instructions, as single agents consumed fewer tokens than multi-agents. The framework has limited capability to count number of tokens used at every step as well as identify which MCP tools are used and how much time is consumed during prompt life cycle. Table 3Table 4 below shows distribution of prompts targeting digital twinning tasks, guideline tasks, planning tasks and not existing tasks across single agents. Present work covered only 31% which is 30 out of 96 possible combinations due to overwhelming number of experimental variations. Authors identified this as a limitation of the current experiment and recognise it as an area for future work.

Table 3: Distribution of prompts in single-agent experiments.

#	Task Type	Prompt Structure						Single Agent			Total Prompts
		Format		Length		Rule					
		Structured Prompt	Unstructured Prompt	Short Prompt	Long Prompt	Conditional Prompt	Unconditional Prompt	Digital Twin Agent	Guideline Agent	Plan Agent	
1	**Digital Twinning Tasks**	✓		✓		✓		83	0	0	**83**
2		✓		✓			✓	0	0	0	**0**
3		✓			✓	✓		169	14	29	**212**
4		✓			✓		✓	0	0	0	**0**
5			✓	✓		✓		0	0	0	**0**
6			✓	✓			✓	0	0	0	**0**
7			✓		✓	✓		125	15	0	**140**
8			✓		✓		✓	0	0	0	**0**
9	**Guideline Tasks**	✓		✓		✓		0	132	0	**132**
10		✓		✓			✓	0	111	0	**111**
11		✓			✓	✓		22	197	22	**241**
12		✓			✓		✓	23	235	23	**281**
13			✓	✓		✓		0	0	0	**0**
14			✓	✓			✓	0	134	0	**134**
15			✓		✓	✓		0	0	0	**0**
16			✓		✓		✓	0	226	0	**226**
17	**Planning**	✓		✓		✓		0	0	0	**0**
18		✓		✓			✓	0	0	27	**27**

19		✓			✓	✓		28	18	1561	**1607**
20		✓			✓		✓	0	19	0	**19**
21			✓	✓		✓		0	0	0	**0**
22			✓	✓			✓	0	0	197	**197**
23			✓		✓	✓		0	0	223	**223**
24			✓		✓		✓	29	0	0	**29**
25	**Not Existing Tasks**	✓		✓		✓		0	0	0	**0**
26		✓		✓			✓	0	0	0	**0**
27		✓			✓	✓		26	26	26	**78**
28		✓			✓		✓	0	0	0	**0**
29			✓	✓		✓		0	0	0	**0**
30			✓	✓			✓	0	0	0	**0**
31			✓		✓	✓		0	0	0	**0**
32			✓		✓		✓	27	27	27	**81**
Total Prompts								**532**	**1154**	**2135**	***3821***

Within orchestration-agent-to-system 120 experiments, the authors designed 1200 prompts to study a limited multi-agent approach using an orchestration agent as an intermediary to perform digital twinning tasks, planning tasks, and guideline tasks using their respective agents. The proposed framework organised these experiments involving 54 experiments on *orchestration agent with digital twin agent*, 36 experiments on *orchestration agent with plan agent*, and 30 experiments on *orchestration agent with guideline agent*. This research did not include interrelated prompting such as "*obtaining a plan from plan agent and executing it through digital twin agent*", because the proposed framework first needed to determine whether a sequential workflow could be established as a reliable use case for evaluating. During experimentation, the proposed framework observed an increased number of limits including limited tokens per minute, and hallucinations. Table 4 below shows distribution of targeted tasks across agents using *orchestration agent* as an intermediary. The proposed framework also did not include *not existing tasks* task type and *stock exchange agent* for *orchestration agent* as such an agent did not exist and the authors found that the framework responded appropriately to requests involving non-existent agents. Table 5 below shows prompt structures used to target digital twinning tasks, guideline tasks, planning tasks, and non-existing tasks by agents using *orchestration agent* as an intermediary. Analysis is difficult, because current framework only recorded orchestration agent's token count and not the total number of tokens used throughout prompt life cycle. In addition, current framework is limited in managing token restrictions directly and could not identify which MCP definitions are used as *orchestration agent* only returned final outcome. This led to many token restriction errors as well as unknown errors which are likely caused by token limitations in underlying agents. The framework could not identify the type of error resulting in needing human inference for managing these situations and guide *orchestration agent* toward expected outcomes. Based on orchestration-agent-to-system results, the proposed framework removed sequential workflows and replaced them with human-in-the-loop workflows as multi-agent setup required human oversight to obtain confirmations because of hallucinations and technical limitations.

Table 4: Distribution of targeted tasks in multi-agent experiments.

#	Target Task	Orchestration Agent			Total Prompts
		Digital Twin Agent	Guideline Agent	Plan Agent	
1	**Digital Twinning Tasks**	351	0	0	**351**

2	**Guideline Tasks**	0	351	0	**351**
3	**Planning Tasks**	0	0	498	**498**
4	**Not Existing Tasks**	0	0	0	**0**
Total Prompts		**351**	**351**	**498**	***1200***

Table 5: Distribution of prompt structures in multi-agent experiments.

#	Prompt Structure						Orchestration Agent			Total Prompts
	Format		Length		Rule					
	Structured Prompt	Unstructured Prompt	Short Prompt	Long Prompt	Conditional Prompt	Unconditional Prompt	Digital Twin Agent	Guideline Agent	Plan Agent	
1	✓		✓		✓		59	34	0	**93**
2	✓		✓			✓	0	7	0	**7**
3	✓			✓	✓		263	73	331	**667**
4	✓			✓		✓	0	110	31	**141**
5		✓	✓		✓		18	0	0	**18**
6		✓	✓			✓	0	8	0	**8**
7		✓		✓	✓		11	0	136	**147**
8		✓		✓		✓	0	119	0	**119**
Total Prompts							**351**	**351**	**498**	***1200***

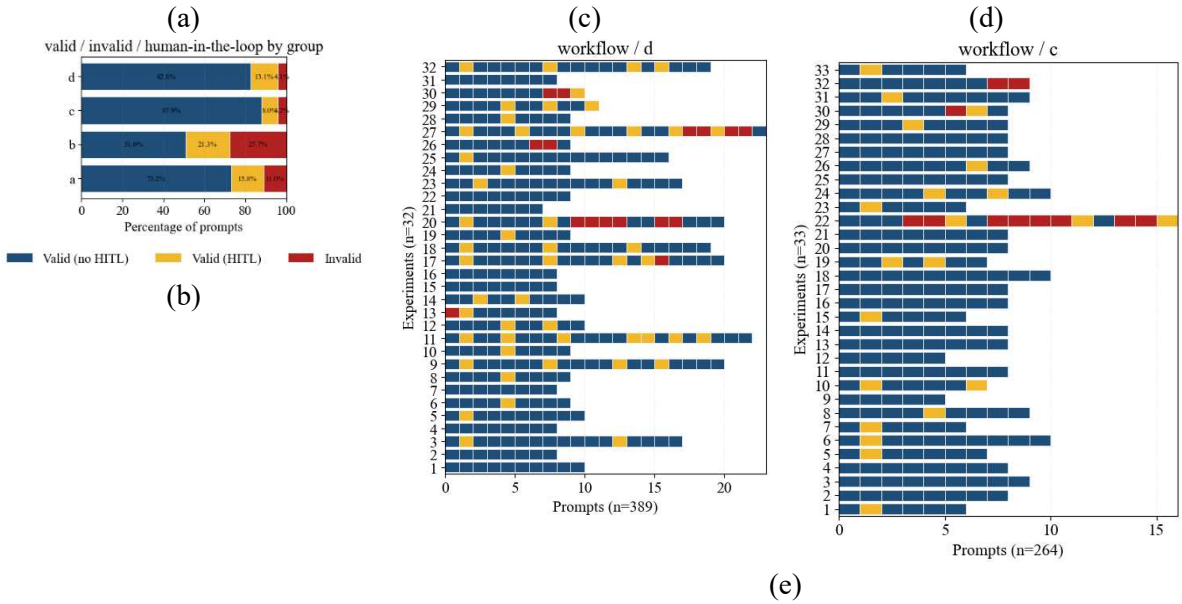
During 131 human-in-the-loop workflows, the authors evaluated 1407 prompts to study multi-agentic approach using *orchestration agent* together with human inference to perform digital twinning tasks, planning tasks, and guideline tasks. Sequential workflow approach is replaced by human-in-the-loop approach because multi-agent experiments struggled to infer due to frequent context window consumption which required continuous human inference to guide multi-agents toward expected outcomes. This is an important criterion observed during orchestration-agent-to-system experiments. Based on previous observations, the proposed framework created subcategories with an additional parameter, namely short and long lengths as token limits continuously affected speed and accuracy. This work created experiments based on combinations of prompt structures targeting individual tasks and combination of tasks as shown in Table 6 given below. Due to overwhelming number of experimental variations of the experimental design, this work covered 13 of the 24 possible combinations, representing 54% of the total. This research recognised this as a limitation of the current experiment and identify it as an area for future work. This research did not evaluate other possible combinations, because regardless of prompt design, multi-agent setup is prone to errors due to token limitations and hallucinations. In addition, this work included multi-agent interactions involving retrieving data from guidelines to generating plans and then applying changes to digital twins.

Table 6: Distribution of prompts in human-in-the-loop multi-agent experiments.

#	Prompt Structure			Target Task										Total Prompts
	Format	Length	Rule	Digital Twinning Tasks	Guideline Tasks	Planning Tasks	Not Existing Tasks	Digital Twinning Tasks	Digital Twinning Tasks	Digital Twinning Tasks	Guideline tasks	Guideline tasks	Planning Tasks	

	Structured Prompt	Unstructured Prompt	Short Prompt	Long Prompt	Conditional Prompt	Unconditional Prompt					Guideline tasks	Planning Tasks	Not Existing Tasks	Planning Tasks	Not Existing Tasks	Not Existing Tasks	
1	✓		✓		✓		0	0	0	0	0	84	0	0	0	0	**84**
2	✓		✓			✓	0	0	0	0	0	0	0	0	0	0	**0**
3	✓			✓	✓		0	182	29	0	0	170	0	316	0	0	**697**
4	✓			✓		✓	0	47	0	0	0	0	0	0	0	0	**47**
5		✓	✓		✓		0	0	0	0	0	54	0	0	0	0	**54**
6		✓	✓			✓	0	0	0	0	0	0	0	0	0	0	**0**
7		✓		✓	✓		0	137	0	0	0	170	0	90	0	0	**367**
8		✓		✓		✓	0	94	34	0	0	0	0	0	0	0	**128**
Total Prompts							**0**	**460**	**63**	**0**	**0**	**478**	**0**	**406**	**0**	**0**	***1407***

Beyond our prompt strategy, the present study gathered insights from each prompt life cycle from every experiment. This research recorded start and end times to identify experiment durations and narrow down digital twins, Internet of Things, and physical twin related results by querying using built-in time intervals. In addition, the present study recorded insights for each prompt life cycle for example run identifier, prompt input and output, and validity which is a flag set to false if the run contained exceptions. This research also recorded start and end times of each prompt, list of MCP calls which included time elapsed for authorisation approval as well as name, list of arguments, and other identifiers along with token details such as number of completion tokens, prompt tokens, and total tokens. During each experiment, the present study recorded this information to generate diagrams shown in Figure 2 and Figure 3.



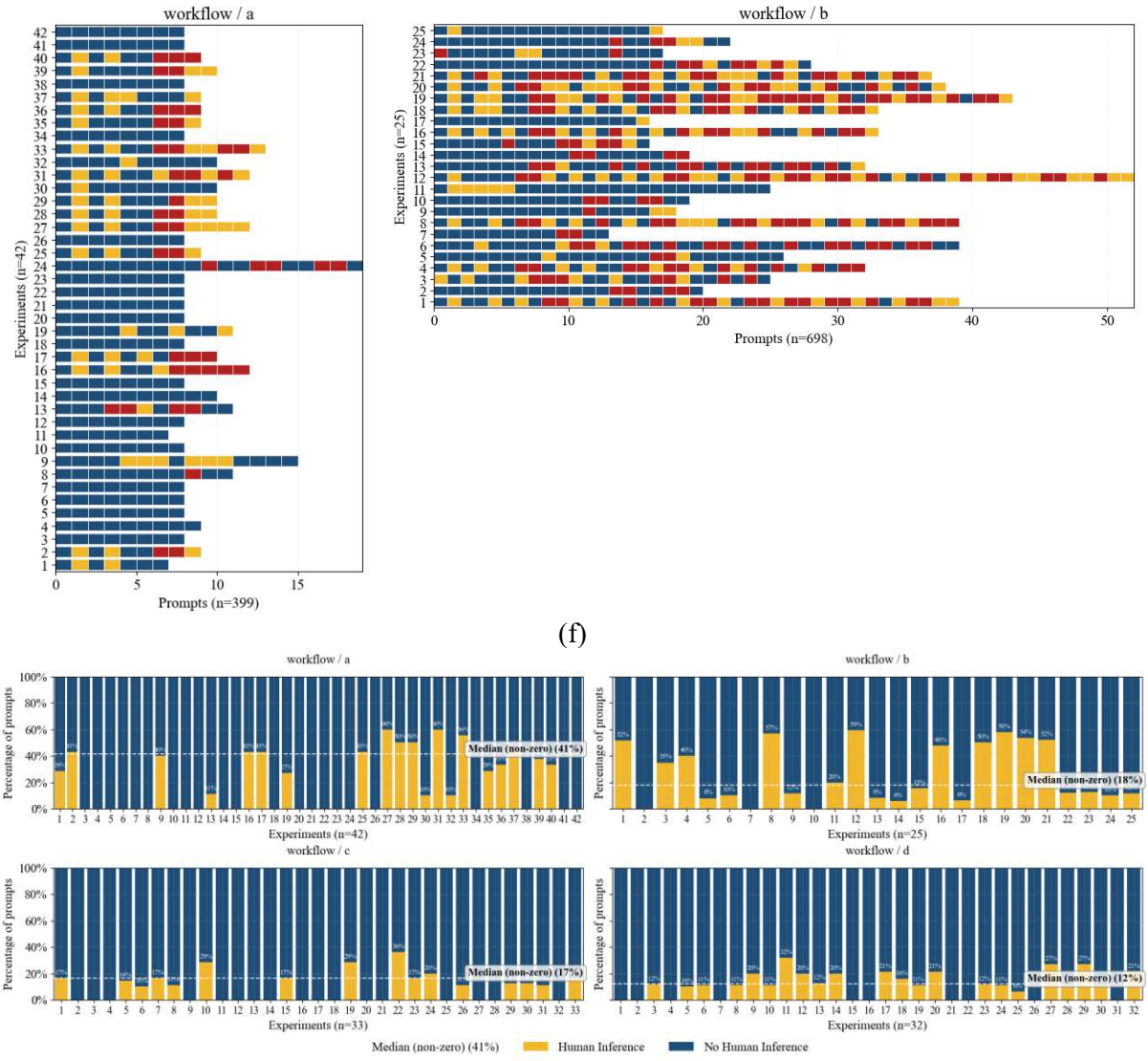


Figure 2. Multi-Agent Aspects: Human-in-the-loop Workflows Metrics.

a. ***Prompt validity in Human-in-the-loop (HITL) Workflows*:** Prompt structured nature, experiment lengths, and number of prompts in an experiment could cause invalid prompts due token limitations along with hallucinations causes need of frequent human inference for multi-agents to achieve goals. On average experiment set of A (9), B (27), C (8), and D (12) prompts each and achieved predefined states without inference by A (73.2%), B (51%), C (87.9%), D (82.8%). Experiment set A which had only 9 prompts per experiment caused more invalid and human inference prompts to reduce prompt validity rate to 73.2% considerably high human inference (15.8%) and invalid prompts due to errors (11%) compared to 8 prompts with less human inference (8%) and invalid prompts (4.2%). On the other hand, C and D had unstructured prompts above 85.5% validity with less human inference 10.5% and invalid prompts 4.5%.

b. ***Experiment Set A*:** Median 9 prompts which are short, structured, unconditional in design for each experiment with 73.2% out of 399 total prompts which are 292 valid prompts, 63 human inferenced prompts, and 44 invalid prompts.

c. ***Experiment Set D*:** Median 12 prompts which are long, unstructured, unconditional in design for each experiment with 82.8% out of 389 total prompts which are 322 valid prompts, 51 human inferenced prompts, and 16 invalid prompts.

d. ***Experiment Set C*:** Median 8 prompts which are short, unstructured, unconditional in design for each experiment with 87.9% out of 264 total prompts which are 232 valid prompts, 21 human inferenced prompts, and 11 invalid prompts.

e. ***Experiment Set B*:** Median 27 prompts which are long, structured, unconditional in design for each experiment with 51% out of 698 total prompts which are 356 valid prompts, 149 human inferenced prompts, and 193 invalid prompts.

f. ***Percentage of Prompts*:** During Human-in-the-workflow experiments the present study designed 132 experiments in to 4 sets of experiments. In set A had total 42 experiments in which 20 recorded at least one human inference and 22 recorded no instances of human inference. As for set B out of 25 experiments 22 recorded at least one instance of human inference and only 3 recorded zero human inference. Then set C out of 33 experiments had 16 prompts with at least one human

inference and 17 recorded no human inference. Finally, set D had 22 human inferred experiments and 10 with 0 human inference prompts out of 32 total experiments.

Based on outcomes of human-in-the-loop workflow experiments, the present research observed that the availability of task-relevant information in each prompt and number of prompts consumed in an experiment could lead to greater human interference. The yellow bars indicate frequent hallucinations, where human judgment is required to determine whether the outcome matched expected goal. This issue forms part of our future work, particularly in exploring how multi-agent systems can be guided to follow an expected goal based on changing states. The red bars may also be reduced by experimenting with novel large language models and by applying necessary implementation improvements such as more robust handling of bad requests, retry methods, and control over frequency of prompts. The time elapsed between the current prompt, and the next prompt may also help to avoid token-related issues. Based on our observations, tracking total token consumption is helpful for identifying token limit errors such as maximum number of tokens per minute and context window limitations for each experiment. In Sets A and B, the present research observed these issues clearly as most experiments began to produce invalid prompts after six prompts. This occurred because during experimentation, the present research did not include a time-wait guardrail before sending next prompt and the framework is unable to record the total number of tokens consumed in multi-agent workflows. This research plan to add these two aspects in future work to increase number of valid prompts.

In addition to the human-in-the-loop workflows, the current work observed that 7% of the agent-to-system experiments and 39% of orchestration-agent-to-system workflows had at least one failure. More than one-third of multi-agent workflows recorded at least one failure due to MCP errors, token limit errors, Azure Foundry errors, and unknown errors arising from framework limitations. For this reason, the current work did not evaluate sequential workflows because based on our observations, this framework could not handle unexpected technical scenarios that could significantly reduce reliability in later prompts. Figure 2 (a) shows success rate of each experiment based on number of prompts that responded successfully with the expected results and number of failed prompts.

The proposed framework encountered four types of errors: Foundry errors, MCP errors, rate-limit errors, and unknown errors. Foundry errors are defined as errors originating from Azure Foundry Classic framework such as 4 *run step not found errors*, which this framework did not handle during experimentation. The run step not found error occurred in only 0.00006% of cases which is very low, but it resulted in a 404-error code for the step identifier. In human-in-the-loop experiments, the proposed framework used human inference to retry the same prompt and continue experiment. This research frequently experienced rate-limit errors (0.04%) originating from cloud, when number of tokens per minute exceeded limit for the current AI model GPT-4.0. This research encountered 318 rate-limit errors in total. To address this, the proposed framework suggest adding manual time-elapsed guardrails by measuring total token consumption to minimise likelihood of experiments reaching the limit as well as applying different language models or other frameworks that support a higher number of tokens per minute. Next, 96 *model context protocol errors* are caused by MCP tools where agent responded with 18 *internal server errors*, 65 *timeout errors*, and 13 *bad request errors*. These errors are observed in 0.014% of all prompts and occurred in 72 agent-to-system experiments, 3 orchestration-agent-to-system experiments, and 21 human-in-the-loop workflow experiments. Finally, 104 *unknown errors* consisted of 103 *run did not complete errors* and 1 *cannot assist the request error*. These unknown errors (0.0151%) did not indicate their exact cause and therefore could not be clearly identified but they are encountered only in multi-agent workflows. The current framework did not have the capability to trace unknown errors, and viewed this as an area for future work. Along with the other results, statistical

probability metrics for each prompt over its life cycle describe the time elapsed from making a request to retrieving a response from cloud. On median the current framework took 7 to 8 seconds to retrieve a result during the prompt life cycle while conservatively it took 11.916 to 14 seconds. However, in some cases human-in-the-loop workflow consumed 115,000 milliseconds to complete a prompt. This could indicate that *digital twin agent* required significant time to execute a command and return a response to multi-agent system. This metric suggests that agents require time for two-way communication between machines and multi-agent systems.

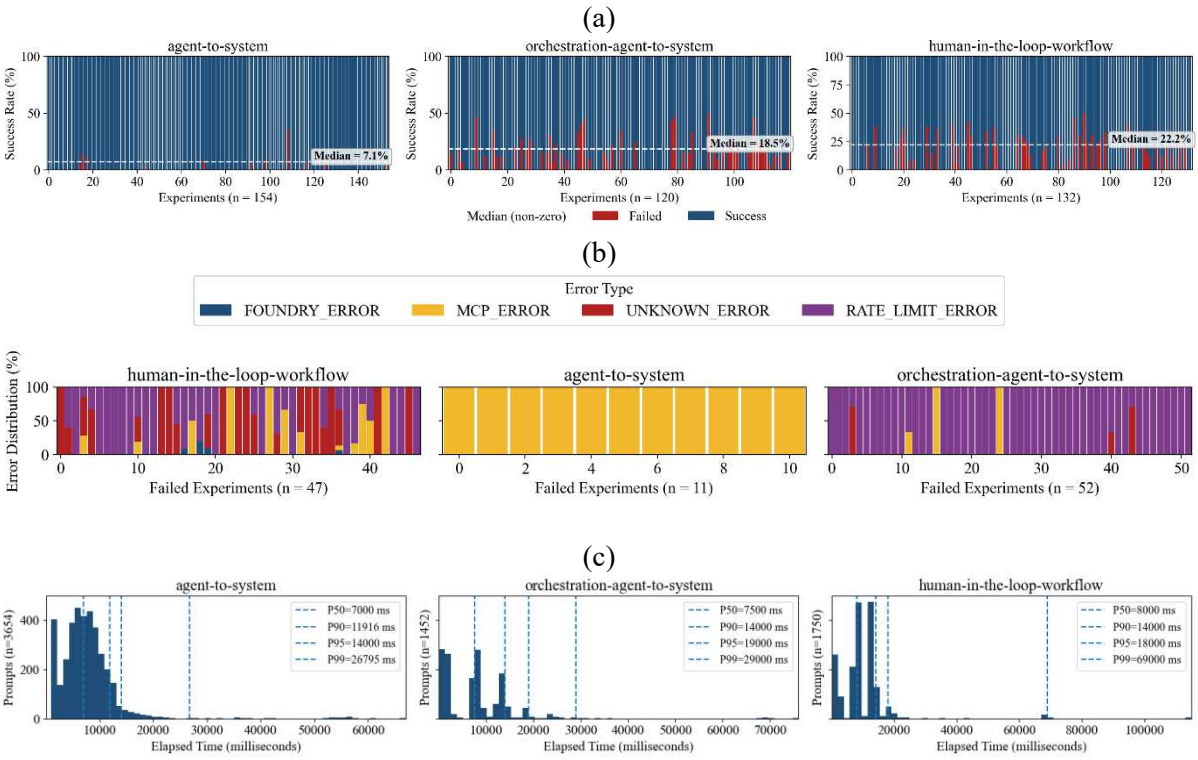


Figure 3. Multi-Agent Aspects: Overall Agentic Chats metrics.

a. ***Success Rates*: S**uccess rate of agent-to-system, orchestration-agent-to-system, and human-in-the-loop experiments are percentages of successful outcomes that retrieved expected output and failed outcomes that did not retrieve the expected output. Single agent experiments only observed failures at least one prompt (7%). But multi-agent experiments namely orchestration-agent-to-system (39%) and human-in-the-loop (35%) recorded at least one difficulty retrieving expected output.

b. ***Error Distribution*:** Types of errors recorded by prompts during multi-agent experiments. This work frequently experienced rate-limit errors (0.04%) originated from cloud when the number of tokens per minute exceeded limit for current AI model. This work then encountered unknown errors (0.0151%) for which this framework could not record origin or reason. Identifying this error is one area of future work. Next, Foundry errors (0.00006%) are infrequent, but 4 instances are identified as having occurred due to concurrency limits. Finally, MCP errors (0.014%) occurred once due to a failure in connecting to a MCP server.

c. ***Prompts*:** Statistical probability metrics of each prompt over its life cycle describe time elapsed from making a request to retrieving a response from cloud. On median, current framework takes 7 to 8 seconds to retrieve a result during prompt life cycle while, conservatively it takes 11.916 to 14 seconds.

Overall agent-to-system workflows performed better than multi-agent workflows. A single agent could interact with digital twins more effectively than multi-agent systems. This work also considered applying multiple MCP definitions to manage digital twins, plans, and guidelines and comparing outcomes with those of agent-to-system and orchestration-agent-to-system workflows. But this is beyond the scope of this proposed framework as authors experimented with multi-agent systems using separate model control protocols.

Next, the proposed framework discusses AI model performance during each experiment. Figure 4 shows number of requests made per minute by AI model, number of input tokens per minute, number of output tokens per minute, and tokens per second. The framework made a median of 7 to 12 requests per minute in each experiment. This metric is heavily dependent on number of prompts requested by the agent in a given minute. However, in an orchestration-agent-to-system scenario, the proposed framework observed 29 requests per minute, which is the highest value recorded during experiments. The input token metric measured total number of tokens processed by AI model. During agent-to-system experiments, each prompt processed 5,572 tokens fewer compared with 7,098 tokens in orchestration-agent-to-system experiments and 9,779 tokens in human-in-the-loop workflows. The multi-agent workflows processed more tokens than single-agent workflows with an increase of 51.43%. The output token metric measured number of completion tokens generated by AI model. The results suggested completion token usage per prompt of 91 tokens in agent-to-system, 189 tokens in orchestration-agent-to-system, and 228 tokens in human-in-the-loop workflows. Based on these observations, multi-agent workflows consumed more output tokens than single-agent workflows with an increase of 128%. Finally, tokens-per-second metric defined number of tokens generated per second calculated by dividing the total number of generated tokens by elapsed time in seconds. This value ranged from 165 to 185 across experiments.

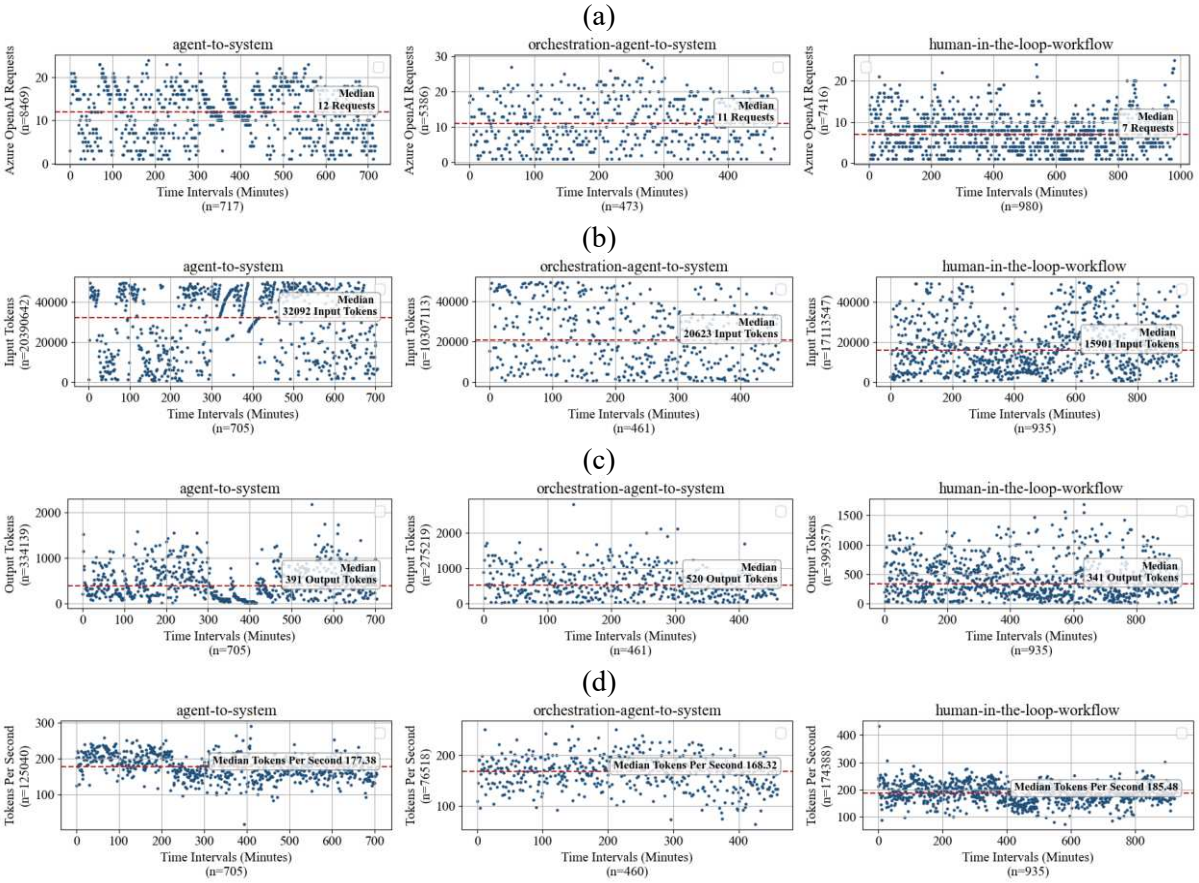


Figure 4. Multi-Agent Aspects: AI metrics.

a. ***Azure OpenAI Requests*:** Number of calls made to Azure OpenAI API over lifetime of each experiment. On median, each prompt made three requests as 21,271 calls depended on total number of interactions made through 6,856 prompts. The maximum number of calls made in a minute is 29 during an orchestration-agent-to-system experiment.

b. ***Input Tokens*:** Total number of prompt tokens processed by AI model in each experiment. Approximately 47.81 million tokens are processed with median of 6,973 tokens processed per prompt. The maximum number of input tokens generated in one minute is 49,998 during an agent-to-system experiment which is below maximum 50,000 tokens per minute allowed by cloud.

c. ***Output Tokens*:** Total number of tokens generated by AI model in each experiment. Slightly more than 1 million tokens are generated as output tokens with median of 147 completion tokens per prompt. The maximum number of completion tokens generated in one minute is 2,800 tokens during an orchestration-agent-to-system experiment.

d. ***Tokens Per Second*:** Token generation speed for prompt responses which is measured by dividing the total number of tokens by token generation time. This number varied from 168 to 185 tokens per second during experiments.

Overall based on results multi-agent workflows consumed more tokens per prompt than single-agent workflows. This paper observed communication between multiple agents in orchestration-agent workflows to support decision-making. Token consumption could increase incrementally based on number of agents and model protocol definitions available ultimately exhausting token usage per minute and causing token limitations. The AI model currently consumed 168 to 185 tokens per second which would require substantial computing resources for deployment in a local environment.

MCP Aspects

The Model Context Protocol (MCP) aspects describe agent's requests to external systems. The proposed framework used this protocol to manage digital twins, guidelines, and planning. This paper implemented the MCP in cloud and included metrics to measure time consumed by MCP layer. Supplementary Figure 1(a) shows time consumed by an agent to authenticate requests, while Supplementary Figure 1(b), Supplementary Figure 1(c), and Supplementary Figure 1(d) show time consumed to process digital twin, guideline, and plan MCP tool requests respectively. Based on results, approving a MCP request by an agent took between 418 and 432 milliseconds. The maximum time recorded to authorise is 10,552 milliseconds which is a significant delay that had to be handled by the proposed framework. Another important point is that this metric worked only for agent-to-system experiments. This is because current framework is limited in capturing step-level insights from single-agent workflows and this work could not duplicate the same step-tracking process for orchestration-agent multi-agent workflows.

To measure time required to complete MCP request, this work added response time metric into target task types. Based on results, guideline and planning required between 120 and 356 milliseconds to perform database-related requests. In contrast digital twinning required more time between 885 and 1,149 milliseconds because they are used to set and retrieve states of digital twins. It is also noteworthy that each request recorded maximum response times ranging from 32,191 to 56,053 milliseconds as cloud serverless containers required cold start to transition from a sleep state to an active state during operations.

Overall agents consumed time to authenticate requests using the client secret associated with agent, to enable access to its MCP definitions. Additionally MCP layer consumed time to perform operations with external systems such as databases or digital twins. Cold starts in cloud also affect performance as significant time is required to wake up before processing the request.

Digital Twin Aspects

This section evaluates the digital twin instance used in the proposed framework. The authors used 14 digital twins to simulate physical states within the instance. Supplementary Figure 2(a) shows the latency required to get or set states in digital twins, Supplementary Figure 2(b) shows number of state changes per minute, Supplementary Figure 2(c) shows the number of telemetry and command events recorded by digital twins, and Supplementary Figure 2(d) shows time consumed to execute command features in digital twins. Cloud services record this information at per-minute intervals, and it is not

possible to extract it as individual events. This paper did not include telemetry update event metrics in this section, because that is associated with Internet of Things aspects.

Firstly, Supplementary Figure 2(a) outlines latency involved in reading or writing states in digital twins. On median, current framework took 47 to 53 milliseconds to get or set states recorded in each experiment. This did not specifically describe time required to perform a command feature in a digital twin but rather latency involved in reading or writing a property in a digital twin. The maximum recorded time is 198 milliseconds. Secondly, Supplementary Figure 2(b) presents total number of states reads and writes per minute in digital twins. On median this framework performed 10 to 32 state reads and writes per minute as this number depended on how frequently telemetry events from Internet of Things are updated or how often command requests are issued by multi-agents. This paper added a 30 second to 1 minute delay guardrail to manage limited number of 8,000 telemetry messages in Azure subscription during experiments. However, the authors occasionally checked how frequently states in digital twins could be read or written. For example, between 0 and 25 minutes in agent-to-system experiments, the framework recorded 100 to 150 state reads and writes per minute with a maximum of 253 state changes per minute. This shows that with an appropriate subscription in place our framework had capability to read and write states of a single digital twin 8 to 18 times per minute which corresponds to every 3 to 7.5 seconds. Thirdly, Supplementary Figure 2(c) defines the total number of states writes per minute. In this case message routed metric recorded number of times per minute that messages are routed when framework changed existing states in digital twins based on telemetry events or command requests. The maximum recorded value is 126 messages per minute and median ranged between 9 and 28 messages per minute. Finally, Supplementary Figure 2(d) describes time consumed in processing command features through Internet of Things layer and updating changes to current states in digital twins. On median, command response time varied between 2,063 and 2,459 milliseconds per request although it could increase to 31,197 milliseconds depending on type of command feature requested. This metric included time consumed in reading current state, updating new state from multi-agents, execution time, and updating new state using execution response. This time is directly related to amount of time required for the command to be executed by its associated physical twins.

Overall, this proposed framework could handle digital twin state changes within 3 to 7 seconds but not in real time at one-second intervals. This paper analysed high-frequency state writes only in a limited number of cases because our current subscription had a daily limit on total number of messages consumed restricted capability. Without a time-delay guardrail framework could update digital twins every 3 seconds from physical twins layer to digital twins layer. From perspective of command features, command event updates could take several seconds because a response had to be sent back from physical twins layer until requested command is successfully executed in physical twins.

Internet of Things Aspects

The proposed framework used three Internet of Things devices to establish a two-way connection between physical twins and digital twins. Supplementary Figure 3(a) shows connected device availability, Supplementary Figure 3(b) shows total number of telemetry and command deliveries, Supplementary Figure 3(c) shows average latency required to publish an event, Supplementary Figure 3(d) shows total number of messages used in each experiment, not specific to digital twins, and Supplementary Figure 3(e) shows amount of time consumed to process and update corresponding digital twin.

Firstly, Supplementary Figure 3(a) outlines availability of devices during experiments. This metric is essential for identifying the framework's ability to establish and maintain connectivity between physical

twins and Internet of Things layer. The framework is able to maintain availability between 95.92% and 98.74% during experimentation. This paper also observed that total disconnection from physical twins occurred only 0.006% to 0.02% of the time. As part of future work this work aim to achieve a higher availability rate and implement appropriate methods to identify lost telemetry events as well as compensate for missing telemetry which may be safety-critical in time-sensitive domains. Secondly, Supplementary Figure 3(b) defines total number of telemetry and command deliveries to digital twins. On median our framework is able to deliver 3 to 10 updates per minute but without guardrails and able to deliver as many as 42 events per minute which is approximately 1,428 milliseconds apart. Thirdly, Supplementary Figure 3(c) explains latency in milliseconds required to publish. Based on results, latency ranged from 148 to 151 milliseconds with deviations from 58.5 to 278 milliseconds. This latency directly increased response times in framework. Fourthly, Supplementary Figure 3(d) describes total message consumption per day. Our current subscription had a limited number of messages. So, this work added a time-wait guardrail to reduce the number of telemetry update events that consumed messages between physical twins and Internet of Things. During experiments framework consumed 3,641 messages in agent-to-system experiments, 3,211 messages in orchestration-agent-to-system experiments, and 4,570 messages in human-in-the-loop workflows. As this metric is not reset daily and these experiments are conducted alongside other experiments such as those involving guidelines and plans with durations overlapping each other. For example, this count overlapped in Supplementary Figure 3(d) as a single total number of messages. Finally, Supplementary Figure 3(e) details response times required to ingest telemetry events Internet of Things into digital twins. On median, this framework took 158 to 188 milliseconds with values ranging from 35 to 1,999 milliseconds based on 14,493 telemetry update events recorded by framework.

In summary it is important to maintain a stable connection between physical twins and digital twins to increase availability rates. Proper safeguards for reconnection are required when connectivity is lost to avoid prolonged disconnections. The total number of telemetry ingestion responses differed because this framework updated sensor digital twins using a single telemetry event from UR Cobot. This incremental growth resulted from two-way communication between physical twins and multi-agent systems..

Physical Twin Aspects

The physical twin aspects depict telemetry extraction and command execution metrics implemented in local hub application. These metrics are divided into two categories: the time elapsed to perform method on device and time elapsed to send the event to Internet of Things. This paper used these metrics to analyse amount of time required by this framework to process telemetry and command features based on physical twin devices. Supplementary Figure 4(a) and Supplementary Figure 4(b) show time elapsed to extract a telemetry event and send that telemetry event to Internet of Things. Supplementary Figure 4(c) and Supplementary Figure 4(d) show time elapsed to execute a command event and send a success or failure response to Internet of Things.

Firstly, Supplementary Figure 4(a) outlines that simulated services GOM Inspect and Robotiq Gripper took 22 to 26 milliseconds as they used simulated values during experimentation. In contrast, UR Cobot used UR Sim a simulation software consumed a median of 2,445 milliseconds to extract a telemetry event and in some cases between 681 and 6,299 milliseconds. This delay represents lowest time framework could achieve at 2,445 milliseconds to simulate this device without cloud services. Secondly, Supplementary Figure 4(b) portrays time elapsed to extract and send a telemetry event to Internet of Things. These values are a combination of both the extract and send functions but this work observed that framework took 1,391 milliseconds to send a telemetry event. The framework recorded a maximum

of 14,394 milliseconds and in some cases it took more than 6,000 milliseconds. Thirdly, Supplementary Figure 4(c) describes amount of time required to execute a command event in physical twins. During experimentation, this paper performed 172 command requests through multi-agents: 24 commands on GOM Inspect, 28 commands on Robotiq Gripper, and 120 commands on UR Cobot. The UR Cobot required only 293 milliseconds for some commands such as enabling and disabling free drive. However, some commands such as move joints took 39,130 milliseconds to complete. For simulated variables this paper applied a 1,000-millisecond delay in GOM Inspect and Robotiq Gripper and their median delays are 1,011 milliseconds for GOM Inspect and 1,022 milliseconds for Robotiq Gripper. This value depended entirely on amount of time physical twins required to complete a command. Finally, Supplementary Figure 4(d) details total time elapsed to perform a command and send response to Internet of Things. It took 920 milliseconds to send a response back to Internet of Things.

Overall, this research used simulated values for GOM Inspect and Robotiq Gripper and UR Sim virtual simulator for the UR Cobot during experiments. This framework consumed time in extracting telemetry events and executing commands along with sending these payloads back to Internet of Things. As future work this research suggest comparing these values with those of real physical twins as well as with other Internet of Things platforms to reduce delays in state changes for multi-agent systems.

Discussion

The proposed framework looked at how to use natural language to perform robot task planning while maintaining a two-way communication link between machines and AI multi-agents via digital twins. The proposed framework presented an online cloud-compatible AI-based multi-agentic digital twin architecture. for cognitive reasoning aspects in robot task planning using multi-agents such as ability to use natural language guidelines to generate plans and execute plan steps on real-time states in digital twins and convergence of physical grounding through real-time harmonisation with established adaptive reasoning using digital twins and AI multi-agents. The primary goal of this study is to establish a consistent link between robots and large language models using digital twins.

The findings from proposed framework indicate that single agents are more accurate than multi-agents in agent-to-system communication. The multi-agent approach caused limitations such as hallucination and token limitations along with requirement for human inference. The human-in-the-loop workflows helped to overcome these limits in multi-agents increasing efficiency and reliability of the framework. As future work, this research recommend implementing an inference layer for multi-agents to automate robot task planning process and to experiment with other workflows such as sequential and group chat.

We encountered a few restrictions during experimentation. First this framework only allows fewer named primitive parameters for control primitives that can be modified without overwhelming prompts. This is a significant challenge identified in existing research[56], but this research used MCP to interface in agent-to-system communication. This protocol provided a standardised approach to communicate with endpoints with or without parameters to perform tasks and return appropriate responses in natural language. But as a limitation this framework struggled to interpret longer and more complex tasks. For example moving a robot arm took considerable amount of time which our framework caused failures, and planning endpoints are complex in design and caused hallucinations in large language model when determining how to use parameters in these MCP definitions. To overcome this command complexity issue[24,56], this research suggest to use robust technical handling such as retrying and human inferencing methods.

Our framework is encountered with large language model API limits as are identified in previous studies[22]. API rate limits are well known and plans are often not completed due to this technical limit. This paper are able to calculate number of tokens used in single-agent mode but this is not possible in multi-model mode. But for multi-agent mode, this research used human inference to determine that this framework is able to meet expected goals.

We retried query based on original prompt whenever this research encountered errors, such as MCP errors, API errors, token limit errors, and hallucinations in human-in-the-loop workflows. These types of errors often compromise reliability of output output[26,33,39,57,58]. For sequential workflows, this research did not implement an inference module to handle similar cases. This remains part of our future work. Mitigating latency is critical for highly dynamic tasks because LLMs are typically cloud-hosted[39]. and although this research deployed framework in cloud a comparison with a locally hosted version is still needed. Current LLM-based and multi-agent frameworks also face inherent limitations including small context windows[59], text-only input requirements, and risk that repeated runtime prompting increases time cost overruns[24], computational scalability burden[60], and latency, reducing suitability for real-time robotic decision-making[61,62]. From a state perspective, receiving feedback from diverse physical sources is more efficient than relying on action feedback[37]. But state awareness remains challenging because of complex state distributions found in open-world environments[63]. Real-time performance in language-model-based multi-agent systems can be improved by increasing computational efficiency and strengthening communication mechanisms[62]. Although current systems still face limitations such as limited computational capacity, suboptimal communication protocols, and difficulties integrating varied sources of data in dynamic environments[62].

We used online cloud ecosystem, instead of edge based deployment to reduce latency for requests from large language model. This way this research could alleviate latency in large language models hosted in cloud services for responses as it can affect for performance in dynamic tasks[39]. Not only can this research address latency issues by integrating framework into cloud architecture, but this research also used various integrated services when establishing an end-to-end machine-to-multi-agentic digital twins framework. Also, built-in features such as metrics and application logs along with a template-based modular architecture have helped locate and fix technical failures in their individual components. But this research identified a few technical limitations such as that researchers must have prior knowledge of integrating cloud services and basic knowledge of authentication and permissions in order to set up this system as there is a lack of documentation or tutorials in multiple languages or for given use cases.

Furthermore, this research experienced that Azure cloud has limitations as some services are not ready for deployment. For example, this research are unable to orchestrate multi-agents build human-in-the-loop workflows use alternative agent-to-agent communication protocols and authorise secured MCP tools from agents using newer versions of *Azure Agent Framework* and *Azure Foundry*. Also, *Azure Digital Twins* service does not support commands so this research had to create a custom function to flag and execute commands from digital twins. The main reason this research observed is that these services are still under development or in preview and require knowledge of identifying and implementing workarounds to get these services up and running without affecting efficiency of the framework. During implementation stage this research used many alternative approaches to establish a reliable solution but sometimes it is hard to overcome hard limits set by Azure cloud such as token limit per minute for large language models for agents, or delays incurred by Azure cloud when exchanging event payloads across cloud services. The most significant hurdle that limited our framework is that *Microsoft Foundry Classic Framework* only allowed a maximum depth of two agents when orchestrating multi-agent aspects. This work hope to overcome these constraints by conducting more experiments on other agent-to-agent communication mechanisms using alternative platforms. In the

future, this research hope to compare our framework with other solutions from other cloud platforms along with in-house implementations to tackle real-time communication problems build in additional data sources such as knowledge graph recommendations and explore alternative cognitive reasoning mechanisms in other scenarios across domains.

Our goal is to build a semi-autonomous robot task planning framework using multi-agentic digital twins to accommodate advantages of real-time mirroring of physical states and digital twins, as well as cognitive reasoning capabilities of large language models. But this research recognised several key aspects that remain open for future enhancement of this framework. Currently the proposed framework only supports text-based numerical telemetry inputs from physical twins. From a multimodal perspective, this framework can be used to gather visual data types such as images, audio, video, etc. But it is required to converge these limitations in process such data by and coalesce these multi model inputs with text based inputs for large language models[64]. As an extension, use of visual language models or multimodal language models to extract semantic information using visual inputs from the environment can be used to enhance cognitive reasoning process of multi-agents. The current framework only supports text as input in human-in-the-loop workflows. However, leveraging multimodal feedback can provide more unbiased feedback, which can enhance reasoning capabilities. Apart from visual data, incorporation of three-dimensional information can enhance spatial reasoning by understanding surrounding environmental context. This integration of spatial data into the cognitive reasoning process can be critical to effectively minimise collisions in dynamic settings. Subsequently, our cloud-oriented framework uses large language models for multi-agent orchestration. This work used GPT-4.0 model for robot task planning. However, task-specific fine-tuning or distillation methods can be used to improve task planning process in multi-agents. Also, further investigations are needed to compare smaller language models and newer large language models to resolve token limitation issues and hallucinations.

Also, by adopting external modules such as graphs and data structures, algorithmic functions, and deep learning methods such as reinforcement learning can be used to investigate their effectiveness in enhancing cognitive reasoning robot task planning process. Dynamic Planning Domain Definition Language (PDDL), Behaviour Trees (BT), and programming language representations for robotic planning methods can also be integrated to enhance effective task representation and execution of planning logic. The application of temporal information can also be utilised to find hidden patterns as digital twins produce many update events for telemetry and command events.

The proposed framework did not focused on multi-robot scenarios at this this stage. This framework can be enhanced to support multi-robot scenarios in collaborative task planning. Due to scope limitations the proposed framework did not consider using ALFRED or OPEN WORLD simulators as they are complex to involve dynamic robot task planning. Finally, replication of functions in this framework on other similar commercial and open-source software vendor platforms such as Amazon Web Services, Eclipse, etc must be investigated to compare their performance with current Azure cloud-based framework.

Methods

The proposed framework materialised bidirectional context translation with task-planning to real-world context by bridging physical twins with cloud-based cognitive multi-agents using digital twins using generalised layers. The Layers subsection discusses the conceptual aspects of this study, while the Use

Case subsection presents its practical application of robot-task planning in product qualification at AddQual Ltd[65], a Derby-based aerospace company.

Layers

This framework consists of five components including physical twins, digital twins, data sources, and multi-agent systems with human-in-the-loop layers as shown in Figure 5. Table 7 summarises the primary notation used in the proposed framework by combining conventional uppercase and lowercase symbols with calligraphic forms, Greek letters, and subscripted or superscripted variants to characterise sets, parameters, and functions.

Table 7: Primary notation summary.

Symbol Definitions	
P = { p | p is a physical twins entity }	I = {i | i is an Internet of things entity}
cmd = Command	tel = Telemetry
req = Request	res = Response
C = {c | c is a connection to I}	L_{dt} = {l_{dt} | l_{dt} is a list of DT}
TA = {ta | ta is a time await}	L_{dtm} = {l_{dtm} | l_{dtm} is a list of DT models}
OP = {op | op *is a object payload*}	K_{dt} = {K_{dt} | K_{dt} is a list of keys in a DT}
T: Π × CTX → T	A: $T \times \Pi \times CTX \rightarrow A$
DS = Data Source	*record* = Data Record
$STEP$ = {*step* | *step* is a data model to store plans}	
DT = { dt | dt is a digital twins entity }	CTX = { ctx | ctx is a language model context }

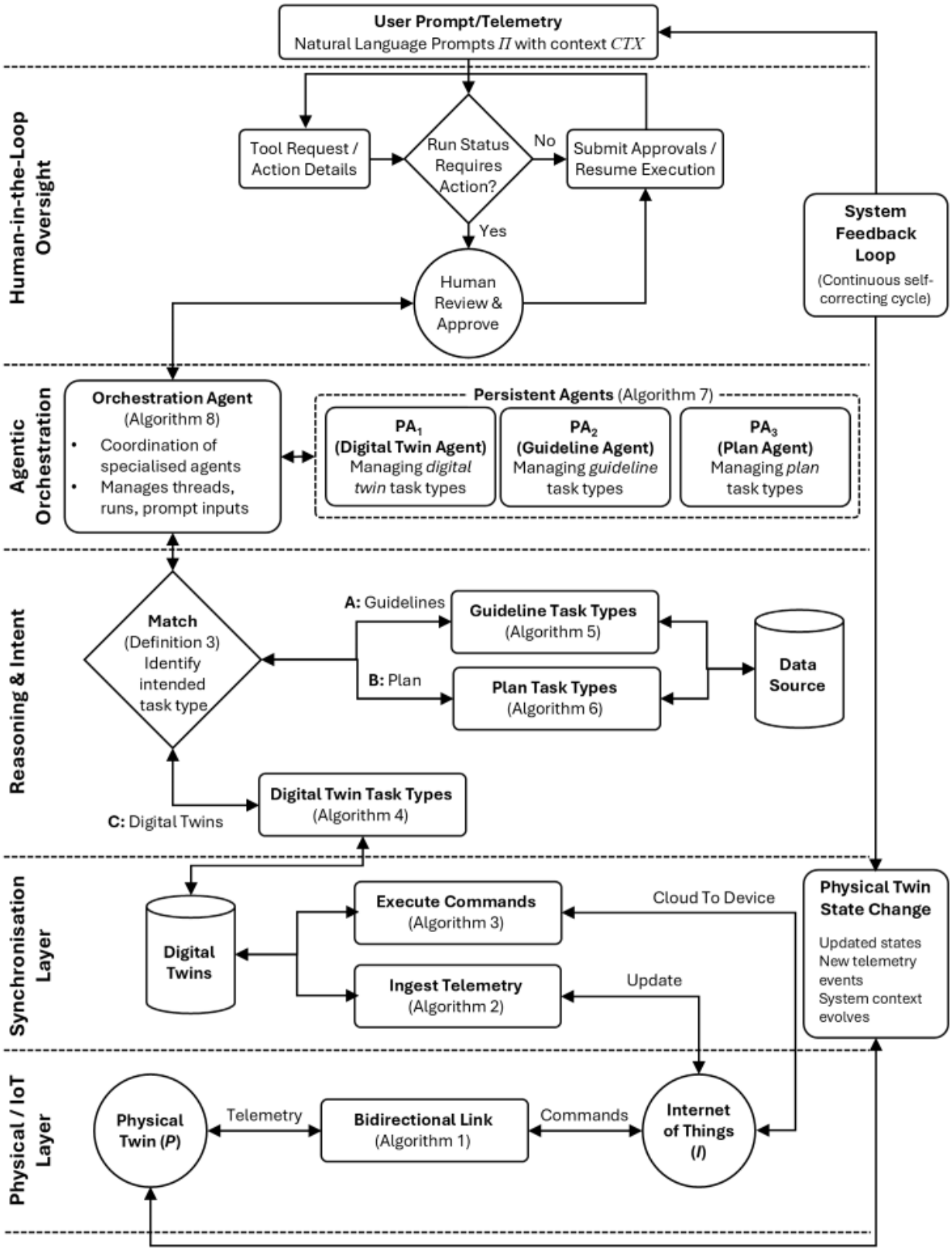


Figure 5: Layers.

The proposed framework links physical twins, IoT connectivity, digital twins, data sources, persistent agents, an orchestration agent, and human-in-the-loop oversight within a continuous feedback cycle. Telemetry from the physical layer is ingested and synchronised with the digital twin while approved commands are proxied back to the IoT interface to update physical states. User prompts and system telemetry are interpreted through intent matching which routes requests to digital twin operations,

guideline retrieval or creation, and plan retrieval or inference. The orchestration agent coordinates specialised persistent agents, while human approval is introduced when state-changing or externally executed actions require oversight. The resulting agent response and physical state changes feed back into the system, enabling continuous context refinement and self-correcting operation.

Definition 1 (Event). An event describes command or telemetry features originating from an entity $\boldsymbol{I}$, $\boldsymbol{P}$, or $\boldsymbol{DT}$:

$$E^{x,y} = \{\, e^{x,y} \mid x \in \{\, I, P, DT \,\}, y \in \{\, cmd, tel \,\} \,\} \quad (1)$$

where $\boldsymbol{e^{x,y}}$ denotes an event, $\boldsymbol{x}$ denotes the event source, and $\boldsymbol{y}$ denotes the event type. Example: $\boldsymbol{e^{P,tel}}$ denotes a telemetry event originating from a physical twin, where $\boldsymbol{P}$ represents physical twins and $\boldsymbol{tel}$ represents telemetry.

Definition 2 (Interaction). An interaction is feedback in the form of a request or response to a command or telemetry feature originating from an entity $\boldsymbol{I}$, $\boldsymbol{P}$, or $\boldsymbol{DT}$:

$$E_z^{x,y} = \{\, e_z^{x,y} \mid x \in \{\, I, P, DT \,\}, y \in \{\, cmd, tel \,\}, z \in \{\, req, res \,\} \,\} \quad (2)$$

where $e_z^{x,y}$ represents an event, x specifies the event source, y indicates the event type, and z refers to the feedback feature. Example: $e_{req}^{P,cmd}$ denotes a command request event originating from physical twin, where P represents physical twins, cmd represents command, and req represents request.

Physical Twins Layer

The physical twins layer describes physical elements such as machines, systems, and processes. An important aspect of this layer is that, it is responsible for establishing and maintaining two-way communication via events and interactions to send telemetry events, execute commands, and send appropriate responses to digital twins. Any state changes in the environment, faults, context, or workflows must be captured accurately and communicated to digital twins as soon as possible. This layer primarily interfaces with physical entities or simulated programs to provide functionality to the system without any direct human oversight. Its ability to change states in a given physical twin as well as communicate existing states requires stronger security measures to prevent unauthorised users from gaining access to these sensitive high-value systems which could in turn physically harm users.

This layer consists of events and interactions. Events are generated from two sources, represented as physical twins (P) in an Internet of Things system (I). Interactions are feedbacks of these events as communications between sources used for requesting and responding with telemetry data or command features.

Events

Derived from Definition 1 (Event) denotes a set of telemetry events generated by P can be defined as:

$$E^{P,tel} = \{\, e \mid e \text{ is a telemetry event from } P \,\} \quad (3)$$

where $e \in E^{P,tel}$ represents an individual telemetry event from P.

Derived from Definition 1 (Event) Similarly, set of command events generated by I can be defined as:

$$E^{I,cmd} = \{\, e \mid e \text{ is a command event from } I \,\} \quad (4)$$

where $e \in E^{I,cmd}$ represents an individual command event from I.

Interaction

Derived from Definition 2 (Interaction), $E_{req}^{P,tel}$ denotes a telemetry request event to an entity P. It represents an event in which P requests telemetry related information such as sensor values, state information, or operational measurements.

$$E_{req}^{P,tel} = \{\, e \mid e \text{ is a telemetry request event to } P \,\} \tag{5}$$

where $e \in E_{req}^{P,tel}$ represents an individual telemetry request event to P.

Derived from Definition 2 (Interaction), $E_{res}^{P,tel}$ denotes the set of telemetry response events originating from an entity P. It represents events in which P provides telemetry-related information such as sensor readings, state information, or operational measurements, in response to a telemetry request.

$$E_{res}^{P,tel} = \{\, e \mid e \text{ is a telemetry response event from } P \,\} \tag{6}$$

where $e \in E_{res}^{P,tel}$ represents an individual telemetry response event from P.

Derived from Definition 2 (Interaction), $E_{req}^{P,cmd}$ denotes the set of command request events originating from an entity P. It represents events in which P requests the execution of a command, control action, or operational instruction.

$$E_{req}^{P,cmd} = \{\, e \mid e \text{ is a command request event to } P \,\} \tag{7}$$

where $e \in E_{req}^{P,cmd}$ represents an individual command request event to P.

Derived from Definition 2 (Interaction), $E_{res}^{P,cmd}$ denotes the set of command response events originating from an entity P. It represents events in which P provides a response to a command request, such as an acknowledgement, execution result, status update, or error message.

$$E_{res}^{P,cmd} = \{\, e \mid e \text{ is a command response event from } P \,\} \tag{8}$$

where $e \in E_{res}^{P,cmd}$ represents an individual command response event from P.

Derived from Definition 2 (Interaction), $E_{res}^{I,tel}$ denotes the set of telemetry response events originating from an interface I. It represents events in which I provides telemetry-related information, such as sensor values, state information, or operational measurements generating from P.

$$E_{res}^{I,tel} = \{\, e \mid e \text{ is a telemetry response event from } I \,\} \tag{9}$$

where $e \in E_{res}^{I,tel}$ represents an individual telemetry response event from I.

Derived from Definition 2 (Interaction), $E_{req}^{I,cmd}$ denotes the set of command request events to I. It represents events in which I requests that a command, control action, or operational instruction be executed in P.

$$E_{req}^{I,cmd} = \{\, e \mid e \text{ is a command request event to } I \,\} \tag{10}$$

where $e \in E_{req}^{I,cmd}$ represents an individual command request event to I.

Derived from Definition 2 (Interaction), $E_{res}^{I,cmd}$ denotes the set of command response events originating from I. It represents events in which I provides a response to a command request, such as an acknowledgement, execution result, status update, or error message.

$$E_{res}^{I,cmd} = \{\, e \mid e \text{ is a command response event from } I \,\} \quad (11)$$

where $e \in E_{req}^{I,cmd}$ represents an individual command response event from I.

Algorithm 1 given below defines the procedure for establishing a two-way connection between P and I. The algorithm begins by initialising the set of physical twins as $P = \{p_1, p_2, \ldots, p_n\}$. For each physical twin $p \in P$, the algorithm first verifies that p belongs to set P. If so, p is added to I. Then connection c is established between p and I using the function $Connect(p, I)$. If c is successfully established, the algorithm processes two categories of bidirectional events: telemetry-related events (Equation (3) and command-related events (Equation (4).

For command-related interactions, the algorithm iterates through each command request event (Equation (10) generated by I. While the request and response events have not yet been completed, the algorithm $Wait$ for the command request. After it verifies whether the request is valid, executes the command (Equation (7) on p. Then it extracts the corresponding command response, and generates a command response event (Equation (8) subsequently sending it to I (Equation (11).

In the second part, the algorithm processes telemetry-related interactions. While telemetry request and response events remain incomplete, p entity $Wait(ta)$ for a telemetry request (Equation (5). Then it extracts the telemetry response, and generates a telemetry response event (Equation (6), subsequently sending this event (Equation (9) to I. This enables information such as sensor readings, state values, and operational measurements to be returned from p entity to its associated I entity.

Algorithm 1: Two-way connection between physical twins & Internet of Things

Initialise: $P = \{ p_1, p_2, \ldots, p_n \}$
for $p \in P$ **do**
  if $p \notin I$ **then**
    $I \leftarrow I \cup \{p\}$
  end
  $c \leftarrow Connect(p, I)$
  if $c = true$ **then**
    for $cmd \in C(cmd)$ **then**
      while $\{(I, e^{I,cmd}) \mid e^{I,cmd} = false\}$ **do**
        $e_{req}^{I,cmd} \leftarrow Wait(e^{I,cmd})$
        if $e_{req}^{I,cmd} = true$ **then**
          $e_{req}^{P,cmd} \leftarrow Execute(e_{req}^{I,cmd}, p)$
          $e_{res}^{P,cmd} \leftarrow Extract(e_{req}^{P,cmd})$
          $e_{res}^{I,cmd} \leftarrow Event(I, e_{res}^{P,cmd})$
        end
      end
    end
  end
  while $\{(I, e^{P,tel}) \mid e^{P,tel} = false\}$ **do**
    $e_{req}^{P,tel} \leftarrow Wait(ta)$
    $e_{res}^{P,tel} \leftarrow Extract(e_{req}^{P,tel})$
    $e_{res}^{I,tel} \leftarrow Event(I, e_{res}^{P,tel})$
  end
end

Digital Twins Layer

The digital twins layer describes conceptual definitions of environment, context, fault, and workflow states. These definitions help multi-agents to ground state changes and the capabilities required to

change these states in real world. Environment can include sensor-related states such as temperature, illuminance, and vibration, as well as physical positioning or operating states of robots. Faults can refer to any deviations that are unfavourable to system performance such as technical or environmental issues. As discussed in main section, every technical fault has to be taken into account because it carries significant value in decision-making and is one of primary concerns in grounding multi-agents as highlighted in much prior research. Context is a conceptual state of real world based on existing predefinitions in plans and guidelines.

In the proposed framework, digital twins layer describes abstract features in physical twins such as components, assets, systems, and processes in a form of telemetry features and command features. Telemetry features are streams of time-series data points that describe states in a physical twin. Command features are executable customisable actions that can alter the state in a physical twin. Telemetry features are not complicated to gather as time-series data points. But specific guidelines must be followed when implementing command features. Primarily, command features must validate requests, execute actions, and communicate to digital twins layer whether task successfully changed existing state or produced an error result. Even though Internet of Things can gather more data points about physical twins in a specific use case or scenario, researchers must determine which features are more significant than the others. This layer can then establish a two-way simulation of selected features in physical twins by using existing states and providing means to change states through Internet of Things. This initial task of selecting which features to include could cause scope or cost overruns as handling two-way communication is a complex process and requires considerable insight before designing digital twins. Another important aspect of designing a digital twin is that mapping does not require definitions as strict as those in physical twins. For example, in a physical twin sensor twins can be bundled together with a control board twin in a robot twin as one system to read telemetry events from sensors through robot's control board. However, this bundled system can be described as separate entities to make sense for multi-agents when making decisions based on digital twins. Telemetry features contain telemetry ingestion events and command features contain command execution events. Telemetry ingestion occurs in *I* and command execution originated from *DT*.

E_{req}^{DT} denotes a set of *DT* update requests, regardless of whether each request is a telemetry event or a command event. Each request represents a key-value mapping of properties or commands for DT.

$$E_{req}^{DT} = \{\, e \mid e \text{ is any update request event to } DT \} \tag{12}$$

where $e \in E_{req}^{DT}$ represents an update request event to DT.

E_{res}^{DT} denotes the set of *DT* update responses, regardless of whether each response corresponds to a telemetry event or a command event. Each response represents the path of the updated property or command and its modified value.

$$E_{res}^{DT} = \{\, e \mid e \text{ is any update response event from } DT \} \tag{13}$$

where $e \in E_{res}^{DT}$ represents an update response event from DT.

$E_{res}^{DT,tel}$ denotes the set of *DT* telemetry update responses. Each response represents the paths of the updated properties and modified values.

$$E_{res}^{DT,tel} = \{\, e \mid e \text{ is a telemetry update response event from } DT \} \tag{14}$$

where $e \in E_{res}^{DT,tel}$ represents an telemetry update response event from DT.

$E_{req}^{DT,cmd}$ denotes the set of DT update requests for commands. Each request represent a command name and payload object of required mappings in DT.

$$E_{req}^{DT,cmd} = \{\, e \mid e \text{ is a command request event to } DT\} \quad (15)$$

where $e \in E_{req}^{DT,cmd}$ represents an command request event to DT.

$E_{res}^{DT,cmd}$ denotes the set of DT update responses feedback from commands execution. Each response represents status, time elapsed during execution, and feedback message mappings in DT.

$$e_{res}^{DT,cmd} = \{\, e \mid e \text{ is a command response event from } DT\} \quad (16)$$

where $e \in e_{res}^{DT,cmd}$ represents an command response event from DT.

Update(e_{req}^{DT}) denotes DT update function applied to a telemetry event, a command request, or the corresponding command response. The feedback generated by this function produces a generalised DT update response, indicating the paths that have been modified in a DT instance. $CloudToDevice(I, e_{req}^{DT,cmd})$ It denotes I update function for executing a command event from a DT command request. This function requires an identifier for I, the command name, and the payload values. DT^x denotes set of telemetry or command paths in a DT. For example, DT^{id} denotes the identifier path of a DT, $DT^{var_temperature}$ denotes a telemetry path, and $DT^{command_movej}$ denotes a command path.

Algorithm 2 defines the ingestion process for telemetry and command responses into DT. For each response event received from I, then algorithm extracts non-null key-value pairs. Then it identifies the corresponding DT and constructs DT update request using only the keys that match valid DT paths. The update function is then applied to the DT by producing a generalised update response. Depending on whether the response corresponds to telemetry or command execution, this algorithm generates either a telemetry update event or a command update event for DT.

Algorithm 2: Ingest telemetry & commands to Digital Twins

Initialise: $e_{res}^{I,tel} = \{e_{res,i}^{I,tel} \mid i = 1, 2, \ldots, n\}$, $e_{res}^{I,cmd} = \{e_{res,i}^{I,cmd} \mid i = 1, 2, \ldots, n\}$
1 $\forall e \in e_{res}^{I,tel} \cup e_{res}^{I,cmd}$:
2 $\quad OP \leftarrow \{ (k, v) \mid k \in Keys(e) \wedge v = e[k] \wedge v \neq null \}$
3 **if** $dt \in DT$ **then**
4 $\quad dt^{id} \leftarrow L_{dt}(OP[p[id]])$
5 $\quad e_{req}^{DT} \leftarrow \{ (k, OP[k]) \mid k \in K(OP) \wedge k \in K_{dt}(dt^{id}) \}$
6 $\quad e_{res}^{DT} \leftarrow Update(e_{req}^{DT})$
7 $\quad$ **if** $e_{res}^{DT} = tel$ **then**
8 $\quad\quad e_{res}^{DT,tel} \leftarrow Event(dt, e_{res}^{DT})$
9 $\quad$ **else if** $e_{res}^{DT} = cmd$ **then**
10 $\quad\quad e_{res}^{DT,cmd} \leftarrow Event(dt, e_{res}^{DT})$
11 $\quad$ **end**
12 **end**

Algorithm 3 describes process of executing command requests generated by DT on I. Algorithm begins with a set of DT command request events denoted as $e_{req}^{DT,cmd}$. For each command request, this extracts all non-null key-value pairs and stores them in OP. If the corresponding DT instance exists in DT then algorithm identifies DT using a lookup function L_{dt}. It then iterates through each command path defined in the DT command set dt^{cmd}.

For each command path, the algorithm extracts required command fields including command flag, command name, and payload values. Then command is considered for execution only if it exists in the

set of valid I command requests $e_{req}^{I,cmd}$ and its command flag is set to true. When these conditions are satisfied, then algorithm constructs a DT command request using available non-null command values. This request is then sent to corresponding I through $CloudToDevice$ function.

After the command is sent, then algorithm waits for a command response from I using $Wait$ function. Once I command response $e_{res}^{I,cmd}$ is received, then algorithm extracts non-null response values and converts them into a DT command update request. The DT is then updated using $Update$ function by producing a DT command response $e_{res}^{DT,cmd}$. Finally, this response is converted into a DT event using the $Event$ function. Overall, Algorithm 3 enables command execution from DT to I and updates DT with execution feedback received from I.

Algorithm 3: Execute commands from Digital Twins on Internet of Things

Initialise: $e_{req}^{DT,cmd} = \{e_{req,i}^{DT,cmd} \mid i = 1, 2, \ldots, n\}$
$OP \leftarrow \{ (k, v) \mid k \in Keys(e_{req}^{DT,cmd}) \wedge v = e_{req}^{DT,cmd}[k] \wedge v \neq null \}$
if $dt \in DT$ **then**
  $dt^{id} \leftarrow L_{dt}(OP[L_{dtm}[id]])$
  for $cmd_{req}^{path} \in dt^{cmd}$ **do**
    $cmd_{req} \leftarrow \{ (k, cmd_{req}^{path}[k]) \mid k \in \{[flag], [command], [payload]\} \wedge (cmd_{req}^{path}[k] \neq null) \}$
    if $cmd_{req}[\in e_{req}^{I,cmd}])$ **then**
      if $cmd_{req}[flag] = true$ **then**
        $e_{req}^{DT,cmd} \leftarrow \{ (k, v) \mid k \in Keys(cmd_{req}) \wedge v = cmd_{req}[k] \wedge v \neq null \}$
        $e_{req}^{I,cmd} \leftarrow CloudToDevice(I, e_{req}^{DT,cmd})$
        $e_{res}^{I,cmd} \leftarrow Wait(e_{req}^{I,cmd})$
        $e_{req}^{DT,cmd} \leftarrow \{ (k, v) / k \in Keys(e_{res}^{I,cmd}) \wedge v = e_{res}^{I,cmd}[k] \wedge v \neq null \}$
        $e_{res}^{DT,cmd} \leftarrow Update(e_{req}^{DT,cmd})$
        $e_{res}^{DT,cmd} \leftarrow Event(dt, e_{res}^{DT,cmd})$
      end
    end
  end
end

Definition 3 (Match). $Match$ denotes the mapping between a prompt-context pair and its matched intent. For each prompt $\pi \in \Pi$ and context $ctx \in CTX$ the function identifies intent t using $T(\pi, ctx)$ and extracts corresponding arguments a using $A(t, \pi, ctx)$. Thus, each match result is represented as a pair $((\pi, ctx), (t, a))$, where t defines intended operation and a contains required attributes for executing that operation. The mapping between a prompt Πand its relevant intent is implemented using the MCP, where language models are used to provide and interpret the contextual information (ctx).

$$Match = \big((\pi, ctx), (t, a)\big) \mid \pi \in \Pi, ctx \in CTX, t = T(\pi, ctx), a = A(t, \pi, ctx)\} \quad (17)$$

Algorithm 4 describes prompting process for interacting with DT through user prompts and contextual information. The algorithm begins by initialising a set of prompts $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_n\}$ and a set of language model contexts $ctx = \{ctx_1, ctx_2, \ldots, ctx_n\}$. Each prompt is matched with its corresponding context using $Match$ function (Definition 3) which produces a matched result $match_i$. This matched result is then decomposed into two components: $match_t$ which represents matched intent or tool, and $match_a$ which represents matched arguments or extracted attributes.

If $match_t$ is identified as a DT related request, then algorithm extracts the DT identifier from $match_a$ and applies the lookup function L_{dt} to obtain corresponding DT identifier dt^{id}. The algorithm then checks whether the selected DT exists in the set of available DTs. If the request is a state retrieval request represented by $[get_state]$, then algorithm returns current state of DT using the $Get(dt^{id})$ function. If the request is a state modification request represented by $[set_state]$, then algorithm extracts command name and payload values from $match_a$. These values are used to construct a command request which is then converted into a DT command update request $e_{req}^{DT,cmd}$. The DT is updated using $Update$ function producing a command response $e_{res}^{DT,cmd}$. While the state change is still active the algorithm repeatedly retrieves updated state of DT using $Get(dt^{id})$ function and waits for a defined time interval ta before checking again. Once state change has completed then final DT state is returned.

Overall, Algorithm 4 enables prompt-based interaction with DT by matching user prompts and contextual information to DT operations. It supports both querying the current state of a DT and modifying its state through command-based updates. This allows natural-language or context-aware prompts to be translated into formal DT operations such as state retrieval, command generation, update execution, and state monitoring.

Algorithm 4: Prompting for digital twins

Initialise: $\Pi = \{\Pi_1, \Pi_2, ..., \Pi_n\}$, $ctx = \{ctx_1, ctx_2, ..., ctx_n\}$
$match_i \leftarrow Match(\Pi_i, ctx_i)$
$(match_t, match_a) \leftarrow match_i$
if $match_t = [digital_twin]$ **then**
  $dt^{id} \leftarrow L_{dt}(match_a[id])$
  if $dt \in DT$ **then**
    if $match_t = [get_state]$ **then**
      return $Get(dt^{id})$
    if $match_t = [set_state]$ **then**
      $cmd_{req} \leftarrow \{ (k, match_a[k]) \mid k \in \{[command], [payload]\} \wedge (match_a[k] \neq null) \}$
      $e_{req}^{DT,cmd} \leftarrow \{ (k, v) \mid k \in Keys(cmd_{req}) \wedge v = cmd_{req}[k] \wedge v \neq null \}$
      $e_{res}^{DT,cmd} \leftarrow Update(e_{req}^{DT,cmd})$
      while *state change = true* **do**
        $state = Get(dt_{id})$
        $Wait(ta)$
      end
      return *state*
    end
  end
end

Data Sources Layer

The data source layer provides means to store conceptual data models such as plans and guidelines. These structured consumable data used by agents to describe plans, guidelines, and related information. It is necessary for these data models to be simple enough to understand while also being rich in information required by agents. Based on results gathered during this experiment authors found that simpler data models performed well in both agent and orchestration-agent scenarios whereas larger data structures could cause hallucinations by agent and require more human attention to enable agent to navigate them in performing such tasks. It is necessary to identify proper definitions, capabilities, and relevant consumable data as these can affect reliability of the agent's output. Supplementary Table 1 presents an example of conceptual data models for guidelines and plans.

Algorithm 5 describes prompting process for guidelines where a set of prompts $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_n\}$ and their corresponding language model contexts $ctx = \{ctx_1, ctx_2, \ldots, ctx_n\}$ are first initialised. Each prompt and context pair is processed using $Match$ function (Definition 3) to identify matched intent by $match_t$ and the matched arguments by $match_a$.

If the matched task type is classified as a guideline, then algorithm extracts all non-null attribute-value pairs associated with the guideline request. It then searches the data source DS for an existing record whose title matches the extracted title value. If no matching record is found, then algorithm infers a new guideline by title and description from provided context and creates a new record. This newly inferred record is then added to DS. Finally, the algorithm returns matched or newly created guideline record.

Overall, Algorithm 5 enables prompt-based retrieval or creation of guideline records by matching user prompts and contextual information to guideline attributes.

Algorithm 5: Prompting for guidelines

Initialise: $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_n\}$, $ctx = \{ctx_1, ctx_2, \ldots, ctx_n\}$
$match_i \leftarrow Match(\Pi_i, ctx_i)$
$(match_t, match_a) \leftarrow match_i$
if $match_t = [guideline]$ **then**
  $match_a \leftarrow \{(attr, v) \mid attr \in Attributes(match_t) \wedge v = match_t[attr] \wedge v \neq null\}$
  $record \leftarrow \{(r.title) \mid r \in DS, \exists v\ ((title, v) \in match_a \wedge r.title = v)\}$
  if $record = \emptyset$ **then**
    $record \leftarrow \{(infer(ctx, title), infer(ctx, description)) \mid (title, description) \in record\}$
    $DS \leftarrow DS \cup \{record\}$
  end
  return *record*
end

Algorithm 6 describes prompting process for plans. It begins by initialising a set of prompts, $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_n\}$ and their corresponding language model contexts, $ctx = \{ctx_1, ctx_2, \ldots, ctx_n\}$. Each prompt-context pair is processed using the $Match$ function (Definition 3) which returns a matched intent $match_t$ and a set of matched arguments $match_a$. If the matched intent is classified as a plan then algorithm extracts all non-null attribute-value pairs associated with the plan request.

Then algorithm searches data source DS for an existing plan record whose name matches extracted plan name. If a matching record is found, then existing record is returned to the plan agent. If no matching record exists, then algorithm infers a new plan from the provided context. This inferred plan includes a name, description, and a sequence of steps. Each step contains a step name and description, together with associated DT state conditions and command actions. The state component defines the DT^{id}, property, and value to be checked. The *true* and *false* branches define DT commands to be executed depending on whether *state condition* is satisfied or not. Each command includes target DT^{id}, command name, and payload values. Once new plan record has been inferred, it is added to data store DS. Finally, the algorithm returns either existing matched plan or newly created plan.

Overall, Algorithm 6 enables prompt-based retrieval or creation of plan records by matching user prompts and contextual information to plan attributes, state conditions, and command-based execution steps.

Algorithm 6: Prompting for plans

Initialise: $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_n\}$, $ctx = \{ctx_1, ctx_2, \ldots, ctx_n\}$
$match_i \leftarrow Match(\Pi_i, ctx_i)$

```
(matchₜ, matchₐ) ← matchᵢ
if matchₜ = [plan] then
    matchₐ ← {(attr, v) | attr ∈ Attributes(matchₜ) ∧ v = matchₜ[attr] ∧ v ≠ null }
    record ← {(r.name) | r ∈ DS, ∃v ((name, v) ∈ matchₐ ∧ r.name= v)}
    if record = ∅ then
        record←{(
          infer(ctx,name),infer(ctx,description),
          [{infer(ctx, step.name), infer(ctx, step.description),
            [{infer(ctx, step.state.twinId), infer(ctx, step.state.property), infer(ctx, step.state.value)}|state ∈
            step.state],
            [{infer(ctx, step.true.twinId), infer(ctx, step.true.command),
              [{infer(ctx, step.true.payload.property), infer(ctx, step.true.payload.value)}|state ∈
              step.true.payload]
              }|state ∈ step.true],
            [{infer(ctx, step.false.twinId), infer(ctx, step.false.command),
              [{infer(ctx, step.false.payload.property), infer(ctx, step.false.payload.value)}|state ∈
              step.false.payload]
              }|state ∈ step.false],
          }|step ∈ STEP ]
          )|(name, description) ∈ matchₐ ∧ (step, STEP) ∈ matchₐ}
        DS ← DS ∪ {record}
    end
    return record
end
```

Multi-Agents Layer

The multi-agent layer provides a two-way communication interface between physical twins and humans by combining agent-to-system communication, agent-to-agent communication, and large language models. In this study multi-agents refer to independent AI agents and orchestration agents that use large language models to perform tasks within a workflow. The multi-agent layer supports the creation and deployment of these agents enabling them to retrieve information, update states, store structured knowledge, and coordinate actions across the framework.

Standardised communication protocols are necessary for multi-agent systems to interact with different agents and external systems. The agent-to-agent protocol provides a standardised messaging framework that enables agents and orchestration agents to discover, authenticate, and collaborate with one another. MCP tools enables agents to connect to external systems including digital twins, databases, and application programming interfaces. In this study MCP is used to define specific tools, specify their accessibility, and enable agents to retrieve existing states, set states through digital twins, and read structured data sources.

Appropriate security measures are required because these tools can retrieve and modify states through natural-language interactions. Access to MCP tools must therefore be controlled to ensure that only authorised agents or users can perform state-changing operations. Large language models support this process by interpreting natural-language inputs, generating human-like responses, and coordinating task execution within a continuous data and human feedback loop.

The framework included four main agents. The digital twin agent interacted with digital twins to retrieve and update existing states. The guideline agent stored structured guidelines, while the plan agent stored structured plans. The orchestration agent had access to the other agents and controlled their behaviour by providing an interface for interaction with humans.

The human feedback component provided human inference and guidance when the framework could not obtain sufficient information from digital twins or data sources. This gave humans greater control over state changes and enabled them to infer or modify plans when necessary. Human involvement is

particularly important when multi-agent workflows encountered faulty states, system errors, or unforeseen circumstances.

The proposed framework implemented two types of chat workflows: a single-agent chat with human-in-the-loop interaction and a multi-agent chat with human-in-the-loop interaction.

Algorithm 7 describes the single-agent chat workflow with human-in-the-loop interaction. This algorithm begins by initialising a set of agent configuration parameters $CFG = \{cfg_1, cfg_2, \dots, cfg_n\}$ and a set of prompts $\Pi = \{\Pi_1, \Pi_2, \dots, \Pi_m\}$. The configuration parameters are used to define the behaviour of the agent including its name, instructions, description, and temperature. These non-null configuration values are extracted and stored as the agent attribute set A.

Then it extracts available communication or service configurations from c including labels and uniform resource identifiers. These values are used to define set of MCP servers or external service connections denoted as MS. For each available service connection, corresponding tool and authentication information are identified and added to the set of MCP tools MT. This allows the agent to access external tools or systems during the conversation provided that the required authentication and approval conditions are satisfied.

A persistent agent PA is then created using the selected model and the extracted agent attributes. The available MCP tools are added to this persistent agent enabling it to perform tool-assisted tasks. A conversation thread TH is created for the persistent agent to maintain the dialogue context across messages and tool calls. The algorithm then processes each Π in the prompt set Π. If Π is a user-defined prompt it is added to the *TH* as a User. If Π is not null or empty it is added as a System message. *TH*, *PA*, and *msg* are then passed to the Run function which starts *PA* prompt life cycle.

During prompt life cycle, this algorithm continuously monitors *run* status. While the *run* is queued, in progress, or requires action then *run* is refreshed and the latest *run* step is retrieved. If *run* requires action, the algorithm identifies tools requested by *PA* and generates approval for the required tools that are available in MT. These approvals are submitted back to *run* allowing *PA* to continue execution. This step represents the human-in-the-loop component where tool use can be reviewed and approved before execution. Once *run* is completed, the latest agent response is retrieved from *TH* and returned as the final response *r*.

Overall, Algorithm 7 defines a single-agent chat architecture in which a persistent language-model-based agent is configured with instructions, connected to authorised tools, and executed through a managed conversation thread. Human-in-the-loop control is incorporated through approval of required tool actions ensuring that the agent can interact with external systems while maintaining oversight over tool execution.

Algorithm 7: Single-Agent chat (Human-in-the-Loop)

Initialise: $CFG = \{cfg_1, cfg_2, \dots, cfg_n\}$, $\Pi = \{\Pi_1, \Pi_2, \dots, \Pi_m\}$
$A \leftarrow \{ (k, v) \mid k \in Keys(cfg) \wedge k \in \{ Name, Instruction, Description, Temperature \} \wedge v = cfg[k] \wedge v \neq null \}$
$MS \leftarrow \{ (k, v) \mid k \in Keys(c) \wedge k \in \{ Label, URI \} \wedge v = c[k] \wedge v \neq null \}$
$MT \leftarrow \{ (tool, auth) \mid ms \in MS \wedge tool = Tool(ms) \}$
$PA \leftarrow PersistantAgent(Model, A)$
$PA \leftarrow PA \cup MT$
$TH \leftarrow Thread(PA)$
for $\Pi \in \Pi$ **do**
  if $\Pi = user_defined_prompt$ **then**
    $msg \leftarrow Message(TH, User, \Pi)$

```
    else if Π ≠ null ∧ Π ≠ empty then
        msg ← Message(TH, System, Π)
    end
    run ← Run(TH, PA, msg)
    while Status(run) ∈ { queued, in_progress, requires_action } do
        run ← Refresh(run)
        run_step ← LatestRunStep(run)
        if Status(run) = requires_action then
            task_approval ← { Approve(mt) | mt ∈ RequiredTools(run) ∧ mt ∈ MT }
            run ← SubmitApprovals(run, task_approval)
        end
    end
    r ← LatestAgentResponse(TH, run)
    return r
end
```

Human-in-the-loop workflows are also designed to address token usage limitations observed during earlier testing. These workflows actively involved humans in setting goals, modifying outputs, and managing outcomes generated by the orchestration agent. Although this increased the number of prompts per experiment it also improved accuracy because humans continuously guided the orchestration agent towards specific goals.

Algorithm 8 describes the multi-agent chat workflow with human-in-the-loop interaction. Unlike the single-agent workflow, this algorithm creates multiple persistent agents *PA* from a set of agent configurations $CFG = \{cfg_1, cfg_2, \dots, cfg_n\}$ and connects them through an orchestration agent *OA*. Each configuration defines the attributes of an individual agent such as its name, instructions, description, and temperature. These non-null values are extracted to form the agent attribute set A which is then used to create PA.

For each agent configuration, the algorithm also extracts the available service or tool connection details including labels and uniform resource identifiers. These are used to construct the set of MCP servers or external service connections MS. The corresponding tools and authentication information are then obtained and stored in MT. These tools are attached to *PA*, allowing each *PA* to access the external systems required for its specific role. The created *PA* are added to the overall set of agents which are later managed by the orchestration agent *OA*.

After all *PA* have been created, the orchestration agent OA is initialised using set of available *PA*. The *OA* coordinates the behaviour of the individual agents and determines which agent or tool should be used to respond to a given prompt. A conversation thread TH is then created for *OA* so that the interaction history and execution context can be maintained throughout the conversation.

The algorithm then processes each prompt in the prompt set $\Pi = \{\Pi_1, \Pi_2, \dots, \Pi_m\}$. If Π is a user-defined prompt it is added to the thread as a *User* message. If the prompt is not null or empty, it is added as a *System* message. The *TH*, *OA*, and *msg* are passed to the Run function to begin execution. During execution, the algorithm monitors the *run* status. While the *run* is queued, in progress, or requires action, the *run* is refreshed and the latest run step is retrieved.

If the *run* requires action, then algorithm identifies tools requested during execution and generates approvals for the required tools that are available in MT. These approvals are submitted back to *run* allowing the execution to continue. This stage represents the human-in-the-loop mechanism where tool use and state-changing actions can be reviewed and approved before they are executed. Once *run* is completed, the latest response *r* generated by *OA* is retrieved from *TH* and returned.

Overall, Algorithm 8 defines a multi-agent chat architecture in which multiple *PA* are created, connected to authorised tools, and coordinated by an *OA*. *OA* then manages collaboration among specialised *PA* while the human-in-the-loop approval process provides oversight over tool use and external system interactions.

Algorithm 8: Multi-Agent chat (Human-in-the-Loop)

Initialise: $CFG = \{cfg_1, cfg_2, ..., cfg_n\}$, $\Pi = \{\Pi_1, \Pi_2, ..., \Pi_m\}$
for $cfg \in CFG$ **do**
  $A \leftarrow \{ (k, v) \mid k \in Keys(cfg) \land k \in \{ Name, Instruction, Description, Temperature \} \land v = cfg[k] \land v \neq null \}$
  $MS \leftarrow \{ (k, v) \mid k \in Keys(cfg) \land k \in \{ Label, URI \} \land v = cfg[k] \land v \neq null \}$
  $MT \leftarrow \{ (tool, auth) \mid ms \in MS \land tool = Tool(ms) \}$
  $PA \leftarrow PersistantAgent(Model, A)$
  $PA \leftarrow PA \cup MT$
  $PA \leftarrow PA \cup \{ PA \}$
end
$OA \leftarrow OrchestrationAgent(PA)$
$TH \leftarrow Thread(OA)$
for $\Pi \in \Pi$ **do**
  if $\Pi = user_defined_prompt$ **then**
    $msg \leftarrow Message(TH, User, \Pi)$
  else if $\Pi \neq null \land \Pi \neq empty$ **then**
    $msg \leftarrow Message(TH, System, \Pi)$
  end
  $run \leftarrow Run(TH, OA, msg)$
  while $Status(run) \in \{ queued, in_progress, requires_action \}$ **do**
    $run \leftarrow Refresh(run)$
    $run_step \leftarrow LatestRunStep(run)$
    if $Status(run) = requires_action$ **then**
      $task_approval \leftarrow \{ Approve(mt) \mid mt \in RequiredTools(run) \land mt \in MT \}$
      $run \leftarrow SubmitApprovals(run, task_approval)$
    end
  end
  $r \leftarrow LatestAgentResponse(TH, run)$
  return r
end

Use Case: AddQual Limited

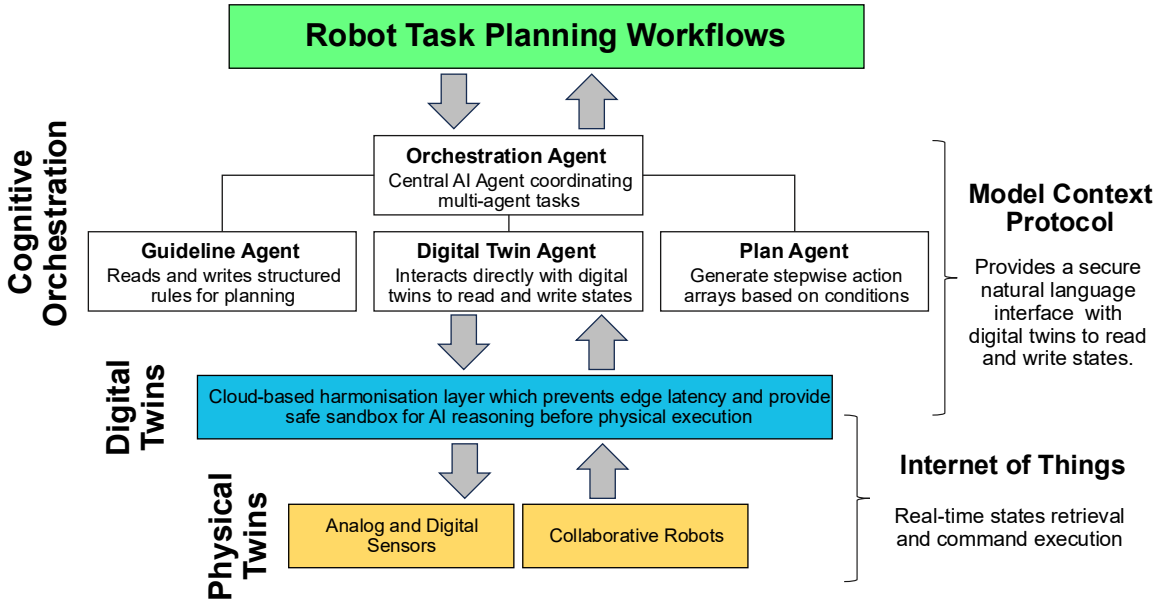


Figure 6. Use case.

The figure illustrates a robot task planning workflow in which high-level planning modules interact with physical robotic systems through an intermediate cloud-based harmonisation layer. At the top, the workflow begins with robot task planning processes represented by multiple planning blocks connected through bidirectional information flow. These planning components appear to exchange task goals, reasoning outputs, and execution feedback. Below them, a cloud-based harmonisation layer acts as a safety and coordination interface. This layer reduces edge latency, provides a secure sandbox for AI reasoning, and validates plans before they are executed in the physical environment. At the bottom, analogue and digital sensors provide environmental input, while collaborative robots carry out the planned actions. The arrows indicate continuous feedback between sensing, cloud reasoning, task planning, and robotic execution.

In order to evaluate this framework, the authors followed a product qualification use case from aerospace domain. AddQual Limited[65] provides product qualification using non-destructive testing by inspecting of high-value components across various industries including aerospace domain offers specialised metrology services to companies in Derby. One use case in this process involves handling aerospace components during metrology process using metrology scanning equipment, collaborative robots, and sensors to automate this workflow which is vital for scaling the business and reducing labour hours. The proposed framework targeted this scenario to develop a multi-agentic digital twin using large language models to explore real-time integration of environment in order to automate metrology scanning process. The proposed framework found that robot plans did not account for changing environmental states as frequent refinements led to repeated process interventions and manual validation of the environment. In addition upskilling existing employees to perform robot-related engineering tasks is complicated because it required highly domain-specific knowledge which caused interruptions to business process.

This experiment is performed extensively using simulations with 6,856 workflow prompts with limited on-site evaluation due to expensive nature of the robotic equipment. The results suggest that augmenting existing workflows with an agentic interface helped humans interact with robots using natural language, requiring only minimal domain-specific knowledge and no programming languages. The framework is also able to design and execute dynamic plans on digital twins while following guidelines expressed in natural language.

The proposed framework used hardware, simulations, and software components. The experiment is designed to interact with both hardware and simulated interfaces: (1) digital infrared sensors; (2) analogue sensors to measure temperature and illuminance; (3) a Universal Robots UR10e collaborative robot (UR Cobot) arm to perform positioning in a three-dimensional Cartesian plane; (4) a Robotiq Two Finger Gripper 2F-85 (Robotiq Gripper) which is a mechanical gripper used to grip and release objects; and (5) ZEISS GOM Inspect (GOM) a metrology scanner equipped with a mechanical robot arm and scanning unit. Due to time and scope constraints, the proposed framework programmed interfaces for the UR Cobot and Robotiq Gripper only. Both UR Cobot and the Robotiq Gripper are designed to interface with the physical device as well as with their simulation program (UR Sim) provided by Universal Robots. GOM Inspect is more complicated to implement because it required domain knowledge of the metrology process as well as specific licensing requirements. The physical twins layer consists of 9 hardware components including UR Cobot, Robotiq Gripper, GOM, and 2 analogue sensors and 4 digital sensors.

UR Cobot is used to manoeuvre payloads between loading and unloading positions. The collaborative robot contains a control board that is used to connect sensors through either digital or analogue outputs. It also has built-in 37 telemetry features and 11 command features, as detailed in Supplementary Table 3: UR Cobot Features. These features are programmed using python socket programming instead of UR Script to provide better control over the collaborative robot. Robotiq Gripper is used to grip and release items as required in the given scenario. The Robotiq Gripper has its own set of 9 telemetry features and 3 command features, as detailed in Supplementary Table 4: Robotiq Gripper Features. Features are limited compared with those of the collaborative robot. Authors programmed the gripper using Python socket programming instead of UR Script to achieve greater control over gripper hardware. GOM is the metrology scanning device integrated with a KUKA robot. Due to scope limitations and licensing issues, the proposed framework did not integrate this platform directly. Instead, the proposed framework simulated the device in a program with the necessary 2 telemetry features and 2 command features , as detailed in Supplementary Table 5: GOM Features. These features were simulated to perform the experiment.

Apart from the robotic systems, this use case included 2 analogue sensors and 2 digital sensors to gather environmental data. The analogue sensors are used to collect continuous voltage data using a temperature sensor and a illuminance sensor. These sensors are physically connected to the analogue input pins on the control board and configured for voltage values in the teach pendant of UR Cobot. These values are then collected from their associated telemetry properties through the UR Cobot's socket. The temperature sensor is used to gather internal temperature values in degrees Celsius and the illuminance sensor is used to gather lux values to measure brightness in the internal environment. The digital sensors are used to collect discrete voltage data (0,1) using 4 infrared sensors. These sensors are also physically connected to the digital input pins on the control board and their values are collected from their associated telemetry properties through the UR Cobot's socket. These infrared sensors are used to detect whether any item is present in the loading or unloading positions.

Algorithm 1 is implemented using a command-line interface (CLI) application written in Python. Figure 9 (a) below shows a screenshot of the application's log. The entire codebase can be accessed in the P1 folder within the solution directory GitHub[1]. The application is interfaced with API Management Service (APIMS) endpoints to manage IoT devices through dynamic resource allocation using the Azure IoT Hub Device Provisioning Service (DPS). The secure endpoints on the backend server are granted admin consent through Microsoft Entra applications to enable application-to-application authentication using API permissions thereby allowing secure communication without human

authentication mechanisms. This effectively restricted access to Azure endpoints to this application and prevented unauthorised access by malicious users or applications. To minimise the risk of exposing client certificates and secrets, study used Azure's default authentication mechanism together with access to Azure Key Vault to store sensitive access keys.

Algorithm 2 is carried out once a telemetry event reaches Azure IoT Hub. An Event Grid trigger Function App ingests telemetry events into Azure Digital Twins based on the device and its associated digital twin. Figure 9 (d) below shows a screenshot of the application's activity log. The C# .NET code for this can be accessed in the P5 folder within the solution directory in GitHub repository[1]. The application then mapped the body of the telemetry payload to the associated properties in the relevant digital twin model by using its digital twin ID. To reduce the risk of security key exposure, the proposed framework used the default authentication mechanism and the required permissions are assigned through a system-managed identity.

The digital twins layer provided real-time simulation of 14 digital twins based on telemetry events and the necessary command request and response values to establish and maintain connectivity between the Internet of Things and Application Programming Interface (API). Azure Digital Twins service is used together with an Event Grid Trigger Function App to request commands with payloads for the Internet of Things layer. Figure 9 (e) below shows a screenshot of the application's activity log. The C# .NET code for this can be accessed in the P6 folder within the solution directory in GitHub[1]. Figure 7 shows a knowledge graph representation of the hierarchical relationships between the digital twins. Initially, the proposed framework designed the *factory* twin as the main twin connecting *robotiq_gripper*, *ur_cobot*, *gom*, and *sensor* twins. Then grouped sensor twins according to their functionality as *illuminance_sensor* for illuminance sensors, *temperature_sensor* for temperature sensors, and *infrared_sensor* for infrared sensors. Each sensor is then numbered by adding a suffix and connected to the appropriate sensor category based on its functionality. Although the *factory*, *sensor*, *illuminance_sensor*, *temperature_sensor*, and *infrared_sensor* twins did not contribute to changing states, this structure helped group them together when presenting the framework to non-technical users. The *robotiq_gripper*, *ur_cobot*, and *gom* twins had both telemetry and command property fields to retrieve and update their states. However, *illuminance_sensor_1*, *temperature_sensor_1*, *infrared_sensor_1*, *infrared_sensor_2*, *infrared_sensor_3*, and *infrared_sensor_4* had only telemetry property fields to retrieve their states. Figure 9 (c) below shows a screenshot of the knowledge graph view in Azure Digital Twins.

The proposed framework used *property* fields instead of *telemetry* fields in the Digital Twins Definition Language (DTDL) definitions to store telemetry data as multi-agent systems could then retrieve state of a digital twin at any given time. In contrast, telemetry events would have required a time-series database to store these events and additional retrieval mechanisms to access them. The *command* fields are not implemented in Azure Digital Twins because this feature is still under preview. To circumvent this limitation, authors defined *property* fields as *command* fields with sufficient space to store request and response payloads. In addition, the proposed framework added a Boolean field as a flag to identify which command had been requested.

Algorithm 3 is adopted when an user requested a specific command from a relevant digital twin. This request triggered a digital twin update event which is then processed by an Event Grid Trigger Function App to identify the relevant digital twin check the flag and retrieve the request values associated with the command field. As discussed earlier, telemetry ingestion did not contain this flag and instead updated the telemetry property fields as well as any responses from relevant command requests. Figure

9 (e) below shows a screenshot of the application's activity log. The C# .NET code for this can be accessed in the P6 folder within the solution directory in GitHub repository[1]. To reduce security risks such as exposed security keys authors used default authentication in this Event Grid Trigger Function App and assigned a system-managed identity with the relevant permissions for Azure IoT Hub and Azure Digital Twins through Identity and Access Management.

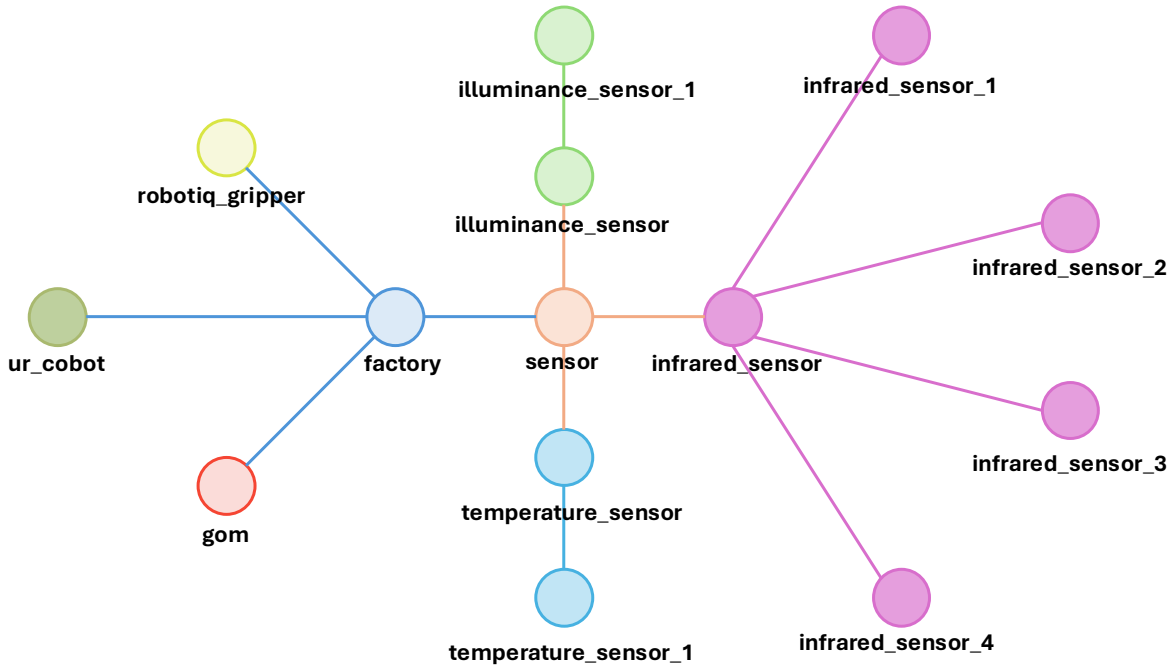


Figure 7. Hierarchical digital twin knowledge graph representation.

Knowledge graph representation of the hierarchical relationships between the digital twins in Azure Digital Twins. The factory twin is modelled as the main twin and connected to the robotiq_gripper, ur_cobot, gom, and sensor twins. Sensor twins are further grouped by functionality into illuminance_sensor, temperature_sensor, and infrared_sensor categories, with individual sensors connected to their respective groups.

An API provided a means of standardising outputs generated by Azure services into a common data format. The API managed data access between: (1) the physical twins layer and the digital twins layer; and (2) the digital twins layer and data sources layer with the multi-agent layer. The data access between the physical twins layer and Azure services included three endpoints implemented to manage the Azure IoT Hub instance from the command-line interface. The data access between the digital twins layer and MCP included 13 endpoints used to manage digital twins through the MCP server. Finally, the data access between the data sources layer and MCP included five endpoints designed to interface with guidelines and 8 endpoints designed to interface with plans. These endpoints are implemented using an App Service application, as it provided Microsoft Entra authorisation headers for the endpoints. This application is written in C# .NET. The code can be accessed in the P4 folder of our GitHub repository[1]. The endpoints are secured through an application-to-application authentication mechanism using app roles, with the allowed member type set to applications. The authors then granted the necessary API permissions to the relevant Entra applications, which helped manage accessibility from a centralised view. The authors then used Azure API Management Service (APIMS) to control access, mask endpoint origins, and evaluate usage in real time using built-in metrics. During implementation, the authors used the default authentication mechanism with the relevant access permissions assigned to its system-managed App Service instance. This provided a more secure access method than client access keys or security certificates, as it increased security by minimising the exposure of such keys.

MCP provided a means to interface APIs with the multi-agent layer using standardised and callable tools designed to interact with external systems and AI agents. This layer contained 3 MCP servers which are designed to provide natural language interfaces related to endpoint functionalities for interaction with the associated AI agents. These MCP servers hosted MCP tools which are endpoints that provided support to agents during prompt interactions. To implement MCP servers in Azure, the authors used MCP Trigger Function Apps written in C# and deployed them as Azure Function Apps using Flex Consumption plans.

The proposed framework implemented Algorithm 4 using 14 *digital twins* MCP tools, Algorithm 5 using 12 *guideline* MCP tools, and Algorithm 6 using 6 *plan* MCP tools to interact with API interface layer endpoints. Each tool are identified by *tool name*, *description*, and *property descriptions* which made it easier for AI agents to determine which tool should be used during prompt inference. The code related to each MCP server can be accessed in the P7 folder for *digital twins*, the P8 folder for *planning*, and the P9 folder for *guidelines* in GitHub[1]. Figure 9 (f) below shows a screenshot of the digital twins MCP tools. Figure 9 (g) below shows a screenshot of the planning MCP tools. Figure 9 (h) below shows a screenshot of the guideline MCP tools. To secure these MCP servers, the authors added Microsoft identity with an overall scope for all tools by using Microsoft Entra. Then assigned Microsoft Entra application identities to each AI agent so that they could authenticate with each corresponding MCP server. At the same time access to API layer is enabled by granting admin consent for relevant API permissions. Study used Azure Log Analytics Workspace to query logs created by each MCP server for evaluation purposes.

The proposed framework integrated multi-agent using *Azure Foundry* service. The authors implemented using *Azure Foundry Classic* as *Microsoft Agent Framework* and *Azure Foundry New* struggled to integrate authorised MCP servers due to lack of documentation. Figure 9 (b) below shows a screenshot of the workflow CLI tool. The code related to workflows application can be accessed in the P10 folder in GitHub[1].

The workflows application implemented Algorithm 7 for single-agent workflows and Algorithm 8 for multi-agent workflows using the GPT-4.0 language model. The proposed framework integrated authenticated MCP tools secured through Microsoft Entra authentication identities for agents. Predefined prompts are then used to experiment with these workflows. Human intervention is added for multi-agent workflows, as the authors identified an increased number of issues specific to multi-agent systems, as discussed earlier in the Results and Discussion sections. At the end of each experiment, the workflow application stored agent and human interactions, resource consumption data, and database snapshots to generate results for visualisation. The Figure 8 below shows the end-to-end Azure architecture used in the proposed framework.

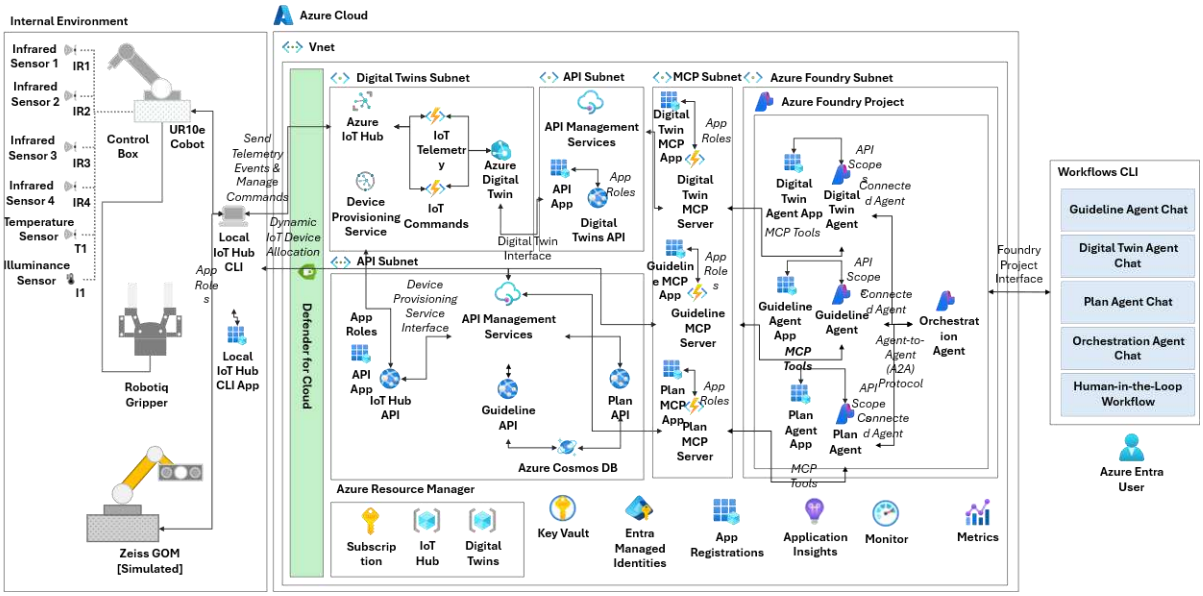


Figure 8. End-to-end Azure cloud architecture.

The proposed framework used a hybrid event-driven serverless architecture to group Azure services, event subscriptions, data services, and serverless applications into a single end-to-end solution. Communication between the multi-agent system and physical twins is event-driven and occurred across multiple serverless applications. Communication protocols are selected based on reliability requirements and the options provided by the Azure platform during implementation. The architecture is grouped into three main sections: the internal environment, Azure cloud, and workflows. The internal environment contained physical devices, such as UR Cobot, Robotiq Gripper, GOM, and environmental sensors including temperature, illuminance, and infrared sensors used in the use case. The Azure cloud is grouped into a single virtual network to manage accessibility, and each layer is organised into virtual subnets to control accessibility among its components. The workflows section contained interfaces for single-agent chat, orchestration-agent chat, and multi-agent human-in-the-loop workflow chats.

Since this framework provides end-to-end interactions using a large language model, the proposed framework integrated various security measures provided by Azure cloud services. Initially, dynamic resource allocation is used for Internet of Things devices to reduce potential threats, such as stolen client access keys or security certificates. This approach allowed client access keys to be created dynamically during initial startup, enabling devices to establish a connection with the Internet of Things platform rather than relying on a pre-configured set of settings. Neither client access keys nor security certificates are stored in configuration files, environments, or any other physical medium. Next, a combination of Azure Entra application scopes is used to define endpoints, along with default authentication instead of client secrets, to interact with the cloud. The framework avoided using client keys or security certificates directly and instead used Azure Key Vault to retrieve keys. This helped avoid vulnerabilities during development and deployment caused by exposed keys. Azure configuration services are used to store other sensitive information. Apart from that, the entire architecture is deployed in the UK South region. Compliance with data protection laws, such as GDPR, and localised storage of customer data are necessary. Although this framework did not store any sensitive information, it is designed with these requirements in mind. Figure 9 given below shows screenshots of tool outputs produced by the proposed framework.

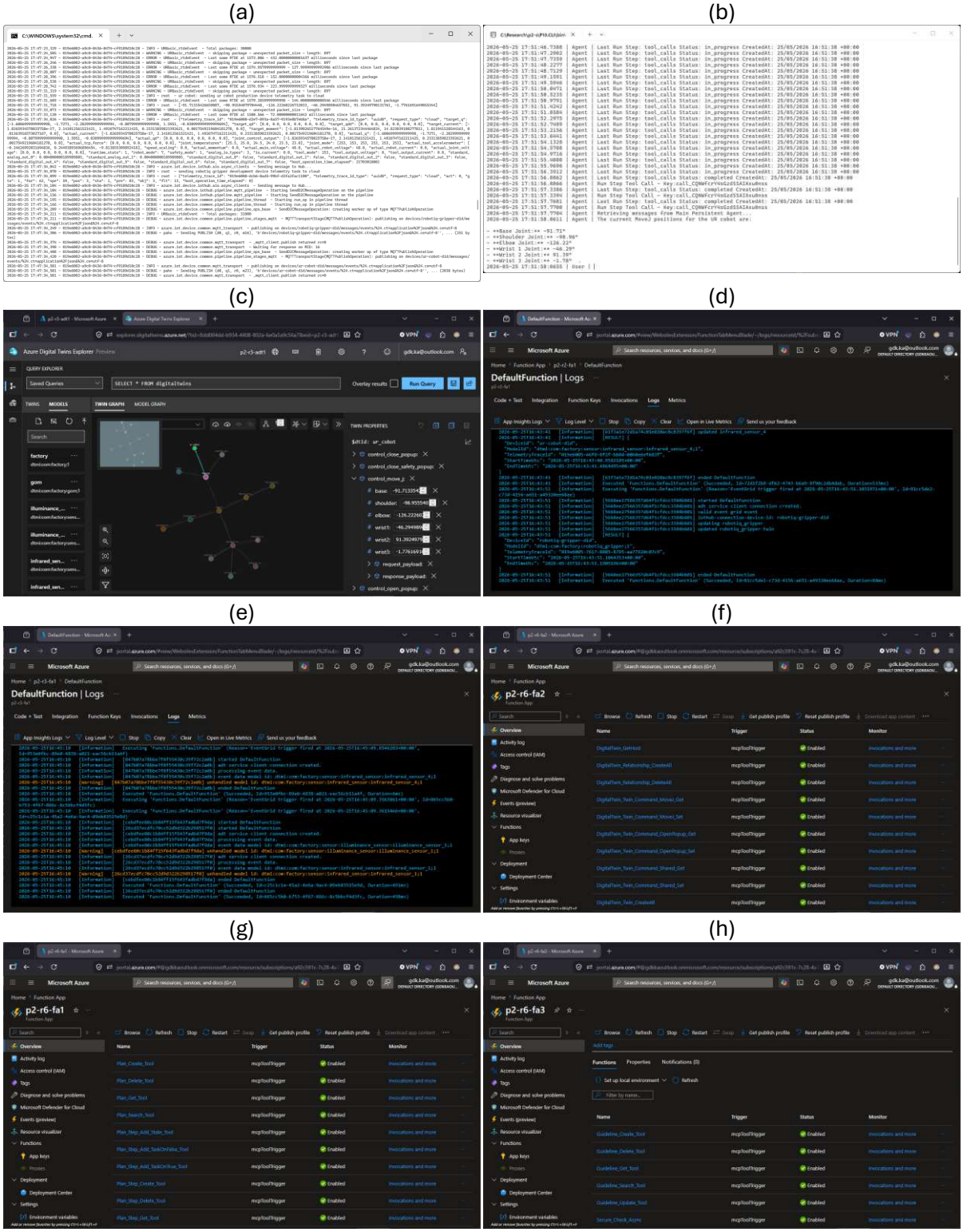


Figure 9: Screenshots of Azure workflow implementation.

This figure presents the main software components used to enable telemetry ingestion, command execution, digital twin synchronisation, and agent-based workflow orchestration.

a) **Local IoT Hub CLI tool:** Shows application log where simulated or device-generated telemetry events and command messages are produced, transmitted, and monitored during execution.

b) **Workflow CLI tool:** Supports both single-agent and multi-agent workflows and includes human-in-the-loop interaction for validating or approving actions during the workflow process.

c) **Azure Digital Twins:** Shows knowledge graph view by illustrating relationships between digital twin entities and their connected components within modelled environment.

d) **IoT Ingestion:** Shows the activity log of *Azure Event Grid Trigger Function App* responsible for ingesting IoT telemetry events and updating the corresponding Azure Digital Twins instances.

e) **Command Execution:** Shows the activity log of the digital twin command trigger *Azure Event Grid Trigger Function App* which processes cloud-side command events and supports interaction between *Azure Digital Twins* and *Azure IoT Hub*.

f) **Digital Twins MCP tools:** Deployed through *Azure MCP Trigger Function App* enabling agent workflows to access and interact with digital twin resources.

g) **Plan MCP tools:** Exposed through the same cloud-based *Azure MCP Trigger Function App* supporting workflow planning and task coordination.

h) **Guideline MCP tools:** Provide structured operational guidance to the agent workflow.

This article demonstrated an end-to-end machine-to-multi-agentic digital twin framework with human-in-the-loop workflow support. The system presents a robotic task-planning use case for AddQual showing how digital twins and multi-agent workflows can be combined to support intelligent decision-making and task execution. Demonstration videos are provided as supplementary material and the complete dataset, results, and source code with README guidance for implementation have been published on GitHub[1].

Data availability

The dataset generated and analysed during this study is publicly available in the project's GitHub repository[1].

Code availability

The source code implementing the proposed framework is publicly available in the project's GitHub repository[1].

Acknowledgement

This study is part of the establishment of a cross-domain agent-based digital twin service hub project (RG24) funded by the Vice-Chancellor's Growth Award Scheme 2025-26 at the University of Derby. The authors gratefully acknowledge AddQual Ltd[65] and its team for providing their support in guidance for the use case.

Author contributions

G.K: Conceptualisation, Methodology, Implementation, Testing, Validation, and Writing.
M.H, W.A.K: Supervision, Review, and Editing. F.M, B.A: Review and Editing.

Competing Interests

The authors declare no competing interests.

References

1. Kuruppu Appuhamilage, G. D. K. gayanvoice/machine-to-multi-agent. *GitHub* https://github.com/gayanvoice/machine-to-multi-agent (2026).

2. Rasheed, A., San, O. & Kvamsdal, T. Digital Twin: Values, Challenges and Enablers From a Modeling Perspective. *IEEE Access* **8**, 21980–22012 (2020).

3. Tao, F., Zhang, H., Liu, A. & Nee, A. Y. C. Digital Twin in Industry: State-of-the-Art. *IEEE Transactions on Industrial Informatics* **15**, 2405–2415 (2019).

4. Ferrari, A. & Willcox, K. Digital twins in mechanical and aerospace engineering. *Nat Comput Sci* **4**, 178–183 (2024).

5. Katsoulakis, E. *et al.* Digital twins for health: a scoping review. *npj Digit. Med.* **7**, 77 (2024).

6. Batty, M. Digital twins in city planning. *Nat Comput Sci* **4**, 192–199 (2024).

7. Tao, F., Zhang, H. & Zhang, C. Advancements and challenges of digital twins in industry. *Nat Comput Sci* **4**, 169–177 (2024).

8. San, O., Rasheed, A., Bozdemir, E. & Deng, J. The evolution of digital twins from reactive to agentic systems. *Nat Comput Sci* **6**, 6–10 (2026).

9. Kapteyn, M. G., Pretorius, J. V. R. & Willcox, K. E. A probabilistic graphical model foundation for enabling predictive digital twins at scale. *Nat Comput Sci* **1**, 337–347 (2021).

10. Wagg, D. J., Worden, K., Barthorpe, R. J. & Gardner, P. Digital Twins: State-of-the-Art and Future Directions for Modeling and Simulation in Engineering Dynamics Applications. *ASME J. Risk Uncertainty Part B* **6**, (2020).

11. Hughes, L. *et al.* AI Agents and Agentic Systems: A Multi-Expert Analysis. *Journal of Computer Information Systems* **65**, 489–517 (2025).

12. Antonesi, G. *et al.* A systematic review of transformers and large language models in the energy sector: towards agentic digital twins. *Applied Energy* **401**, 126670 (2025).

13. Multi-agent AI systems need transparency. *Nat Mach Intell* **8**, 1–1 (2026).

14. Xin, H., Kitchin, J. R. & Kulik, H. J. Towards agentic science for advancing scientific discovery. *Nat Mach Intell* **7**, 1373–1375 (2025).

15. Nicoletti, B. & Appolloni, A. A digital twin framework for enhancing human–agentic AI–machine collaboration. *J Intell Manuf* https://doi.org/10.1007/s10845-025-02637-x (2025) doi:10.1007/s10845-025-02637-x.

16. Hasan, A. & Nguyen, D. T. Integrating agentic AI and digital twins for intelligent decision-making systems. *Array* **29**, 100721 (2026).

17. Deng, X. *et al.* GMATP-LLM: A General Multi-Agent Task Dynamic Planning Method using Large Language Models. in *2025 44th Chinese Control Conference (CCC)* 5792–5798 (2025). doi:10.23919/CCC64809.2025.11179615.

18.Cong, W., Tao, W. & Jinsong, B. Agentic digital twin-embedded maintenance methodology for energy equipment: A self-evolving operational paradigm. *Journal of Manufacturing Systems* **84**, 100–116 (2026).

19.Ding, Y. *et al.* Integrating action knowledge and LLMs for task planning and situation handling in open worlds. *Auton Robot* **47**, 981–997 (2023).

20.Babu, H., Schillinger, P. & Asfour, T. Adaptive Domain Modeling with Language Models: A Multi-Agent Approach to Task Planning. in *2025 IEEE 21st International Conference on Automation Science and Engineering (IEEE CASE)* (arXiv, Los Angeles, California, USA, 2025). doi:10.48550/arXiv.2506.19592.

21.Yang, Z. *et al.* Text2Reaction : Enabling Reactive Task Planning Using Large Language Models. *IEEE Robotics and Automation Letters* **9**, 4003–4010 (2024).

22.Singh, I. *et al.* ProgPrompt: program generation for situated robot task planning using large language models. *Auton Robot* **47**, 999–1012 (2023).

23.Singh, I. *et al.* ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. in *2023 IEEE International Conference on Robotics and Automation (ICRA)* 11523–11530 (2023). doi:10.1109/ICRA48891.2023.10161317.

24.Chen, Y. *et al.* AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers. in *2024 IEEE International Conference on Robotics and Automation (ICRA)* 6695–6702 (2024). doi:10.1109/ICRA57147.2024.10611163.

25.Yoneda, T. *et al.* Statler: State-Maintaining Language Models for Embodied Reasoning. in *2024 IEEE International Conference on Robotics and Automation (ICRA)* 15083–15091 (2024). doi:10.1109/ICRA57147.2024.10610634.

26.Huang, W. *et al.* Inner Monologue: Embodied Reasoning through Planning with Language Models. in *Proceedings of The 6th Conference on Robot Learning* 1769–1782 (PMLR, 2023).

27.Zhou, Z., Song, J., Yao, K., Shu, Z. & Ma, L. ISR-LLM: Iterative Self-Refined Large Language Model for Long-Horizon Sequential Task Planning. in *2024 IEEE International Conference on Robotics and Automation (ICRA)* 2081–2088 (2024). doi:10.1109/ICRA57147.2024.10610065.

28.Arenas, M. G. *et al.* How to Prompt Your Robot: A PromptBook for Manipulation Skills with Code as Policies. in *2024 IEEE International Conference on Robotics and Automation (ICRA)* 4340–4348 (2024). doi:10.1109/ICRA57147.2024.10610784.

29.Guan, L., Valmeekam, K., Sreedharan, S. & Kambhampati, S. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. in *Proceedings of the 37th International Conference on Neural Information Processing Systems* 79081–79094 (Curran Associates Inc., Red Hook, NY, USA, 2023).

30.Zhao, X., Li, M., Weber, C., Hafez, M. B. & Wermter, S. Chat with the Environment: Interactive Multimodal Perception Using Large Language Models. in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* 3590–3596 (2023). doi:10.1109/IROS55552.2023.10342363.

31.Cao, H. *et al.* LLM-Collab: a framework for enhancing task planning via chain-of-thought and multi-agent collaboration. *ACI* **4**, 328–348 (2024).

32.Tellex, S., Gopalan, N., Kress-Gazit, H. & Matuszek, C. Robots That Use Language. *Annual Review of Control, Robotics, and Autonomous Systems* **3**, 25–55 (2020).

33.Rana, K. *et al.* SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning. in *Proceedings of The 7th Conference on Robot Learning* 23–72 (PMLR, 2023).

34.Liu, Z., Bahety, A. & Song, S. REFLECT: Summarizing Robot Experiences for Failure Explanation and Correction. Preprint at https://doi.org/10.48550/arXiv.2306.15724 (2023).

35.Raman, S. S. *et al.* CAPE: Corrective Actions from Precondition Errors using Large Language Models. Preprint at https://doi.org/10.48550/arXiv.2211.09935 (2024).

36.Gupta, S., Yao, K., Niederhauser, L. & Billard, A. Action Contextualization: Adaptive Task Planning and Action Tuning Using Large Language Models. *IEEE Robotics and Automation Letters* **9**, 9407–9414 (2024).

37.Ichter, B. *et al.* Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. in *Proceedings of The 6th Conference on Robot Learning* 287–318 (PMLR, 2023).

38.Huang, W., Abbeel, P., Pathak, D. & Mordatch, I. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. in *Proceedings of the 39th International Conference on Machine Learning* 9118–9147 (PMLR, 2022).

39.DeepMind, G. *Demonstrating Large Language Models on Robots*. (2023). doi:10.15607/RSS.2023.XIX.024.

40.Schick, T. *et al.* Toolformer: language models can teach themselves to use tools. in *Proceedings of the 37th International Conference on Neural Information Processing Systems* 68539–68551 (Curran Associates Inc., Red Hook, NY, USA, 2023).

41.Li, J. *et al.* AI agent-based virtual model development, diagnosis, and calibration for building digital twins. *Journal of Industrial Information Integration* **48**, 100990 (2025).

42.Shen, X., Wagg, D. J., Tipuric, M. & Bonney, M. S. Digital twins as self-models for intelligent structures. *Sci Rep* **15**, 30327 (2025).

43.Lee, J., Li, J. & Yoon, S. From design to operation: Multi-agent AI for virtual in-situ modeling of digital twins in BIM. *Automation in Construction* **179**, 106477 (2025).

44.Liu, C. *et al.* From insight to autonomous execution: VLM-enhanced embodied agents towards digital twin-assisted human-robot collaborative assembly. *Robotics and Computer-Integrated Manufacturing* **98**, 103176 (2026).

45.Marah, H. & Challenger, M. MADTwin: a framework for multi-agent digital twin development: smart warehouse case study. *Ann Math Artif Intell* **92**, 975–1005 (2024).

46.Collao, C. *et al.* An Agentic AI-based Architecture for Digital Twins Specialized in Predictive Maintenance: Application to Ball Mills. *IFAC-PapersOnLine* **59**, 96–101 (2025).

47.Tucker, M., Aksaray, D., Paul, R., Stein, G. J. & Roy, N. Learning Unknown Groundings for Natural Language Interaction with Mobile Robots. in *Robotics Research* (eds Amato, N. M., Hager, G., Thomas, S. & Torres-Torriti, M.) 317–333 (Springer International Publishing, Cham, 2020). doi:10.1007/978-3-030-28619-4_27.

48.Chernova, S. *et al.* Situated Bayesian Reasoning Framework for Robots Operating in Diverse Everyday Environments. in *Robotics Research* (eds Amato, N. M., Hager, G., Thomas, S. & Torres-Torriti, M.) 353–369 (Springer International Publishing, Cham, 2020). doi:10.1007/978-3-030-28619-4_29.

49.Kant, Y. *et al.* Housekeep: Tidying Virtual Households Using Commonsense Reasoning. in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIX* 355–373 (Springer-Verlag, Berlin, Heidelberg, 2022). doi:10.1007/978-3-031-19842-7_21.

50.Silver, T. *et al.* Generalized Planning in PDDL Domains with Pretrained Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence* **38**, 20256–20264 (2024).

51.LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. https://www.computer.org/csdl/proceedings-article/iccv/2023/071800c986/1TJj13NLZ0k.

52.Lin, K., Agia, C., Migimatsu, T., Pavone, M. & Bohg, J. Text2Motion: from natural language instructions to feasible plans. *Auton Robot* **47**, 1345–1365 (2023).

53.Skreta, M. *et al.* Errors are Useful Prompts: Instruction Guided Task Programming with Verifier-Assisted Iterative Prompting. https://doi.org/10.48550/ARXIV.2303.14100 (2023) doi:10.48550/ARXIV.2303.14100.

54.Carta, T. *et al.* Grounding Large Language Models in Interactive Environments with Online Reinforcement Learning. in *Proceedings of the 40th International Conference on Machine Learning* 3676–3713 (PMLR, 2023).

55.Fuller, A., Fan, Z., Day, C. & Barlow, C. Digital Twin: Enabling Technologies, Challenges and Open Research. *IEEE Access* **8**, 108952–108971 (2020).

56.Liang, J. *et al.* Code as Policies: Language Model Programs for Embodied Control. in *2023 IEEE International Conference on Robotics and Automation (ICRA)* 9493–9500 (2023). doi:10.1109/ICRA48891.2023.10160591.

57.Wang, R. *et al.* LLM-based Robot Task Planning with Exceptional Handling for General Purpose Service Robots. in *2024 43rd Chinese Control Conference (CCC)* 4439–4444 (2024). doi:10.23919/CCC63176.2024.10661966.

58.Generative Expressive Robot Behaviors using Large Language Models | Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction. https://dl.acm.org/doi/10.1145/3610977.3634999.

59.Mahadevan, K. *et al.* Generative Expressive Robot Behaviors using Large Language Models. in *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction* 482–491 (Association for Computing Machinery, New York, NY, USA, 2024). doi:10.1145/3610977.3634999.

60.Lin, B. Y. *et al.* SWIFTSAGE: a generative agent with fast and slow thinking for complex interactive tasks. in *Proceedings of the 37th International Conference on Neural Information Processing Systems* 23813–23825 (Curran Associates Inc., Red Hook, NY, USA, 2023).

61.Agashe, S., Fan, Y., Reyna, A. & Wang, X. E. LLM-Coordination: Evaluating and Analyzing Multi-agent Coordination Abilities in Large Language Models. in *Findings of the Association for Computational Linguistics: NAACL 2025* (eds Chiruzzo, L., Ritter, A. & Wang, L.) 8038–8057

(Association for Computational Linguistics, Albuquerque, New Mexico, 2025). doi:10.18653/v1/2025.findings-naacl.448.

62. Jia, S. *et al.* Adversarial and Cooperation Tasks in Multi- Agent System with Large Language Models. in *2024 IEEE International Conference on Unmanned Systems (ICUS)* 1840–1847 (2024). doi:10.1109/ICUS61736.2024.10839948.

63. Wang, Z. *et al.* Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. in *Proceedings of the 37th International Conference on Neural Information Processing Systems* 34153–34189 (Curran Associates Inc., Red Hook, NY, USA, 2023).

64. Yoon, S., Song, J. & Li, J. Ontology-enabled AI agent-driven intelligent digital twins for building operations and maintenance. *Journal of Building Engineering* **108**, 112802 (2025).

65. AddQual Limited. AddQual. *AddQual* https://www.addqual.com/.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- 1testpromptsv120260716.zip
- 2natmachintellsupplementaryinformationv220260716.pdf
- 3MovieCreatingaguidelineusingorchestrationagent.mp4
- 6MovieExecutingaplanusingorchestrationagent.mp4
- 4MovieExecutingaguidelineusingorchestrationagent.mp4
- 2resultsv120260716.zip
- 1MovieExploringtheframework.mp4
- 2MovieExecutingcommandsusingorchestrationagent.mp4
- 5MovieCreatingaplanusingorchestrationagent.mp4