An Architecture for Integrating Large Language Models with Digital Twins and Automation Systems

Yuchen Xia, Nasser Jazdi, Michael Weyrich
Institute of Industrial Automation and Software Engineering
University of Stuttgart
Stuttgart, Germany
yuchen.xia@ias.uni-stuttgart.de; nasser.jazdi@ias.uni-stuttgart.de; michael.weyrich@ias.uni-stuttgart.de

***Abstract*— Large Language Models (LLMs) offer flexible reasoning capability but lack physical embodiment, while traditional automation systems can execute physical processes yet lack cognitive capability. This paper presents a layered architecture that bridges this gap by integrating LLMs with digital twins and physical automation systems, with practical case studies as proof of concept. The proposed architecture comprises three layers: a cognitive layer powered by LLMs, a bridging layer based on digital twins, and a physical layer consisting of technical process and automation system. Within the digital twin layer, we introduce three design paradigms for structuring information to support effective LLM integration: state snapshot modeling, event message modeling, and plan sequence modeling. These paradigms are demonstrated through prototypical case studies on robotic automation control and process simulation. To address challenges such as hallucination, task complexity, and system reliability, we distill a set of practical strategies, including multi-agent system design, human validation and test-driven development. Additionally, we propose the concept of "Return on Intelligence" as a conceptual tool for evaluating the efficacy of investments in intelligent automation. This research contributes to the theoretical foundation and the architecture design for developing intelligent, adaptive automation systems powered by LLMs.**

Keywords*— *Large Language Model, Industrial Automation System, Digital Twin, Autonomous System, Intelligent Robotics, Multi-Agent System

I. Introduction

Traditional industrial automation systems—comprising machinery, sensors, actuators, controllers, and manufacturing execution systems—excel at performing repetitive tasks efficiently. However, as market demands shift and production requirements diversify, these rigid systems struggle to adapt. Factory customers seeking swift, cost-effective reconfigurations to maintain competitiveness often encounter knowledge barriers, since modifying and maintaining automation systems typically require specialized technical expertise.

Given these challenges, several key questions emerge:

- How can automation systems overcome their inherent rigidity to become more adaptable and flexible?
- What can provide the intelligence required to handle complexity, particularly when human expertise may be limited or unavailable?
- By what mechanisms can automation systems obtain the intelligence they need for adaptive task-solving?
- From which sources should the information and knowledge required for intelligent decision-making be obtained, and how can they be effectively utilized?

One promising answer lies in large language models that are capable of interpreting textual data and reasoning based on patterns in existing human knowledge. By integrating LLMs into automation environments, it becomes possible to enable more flexible task solving, contextual reasoning, and user interaction.

However, LLMs lack physical embodiment and cannot directly perceive or influence real-world systems. To unlock their potential in industrial automation, they must be tightly integrated with digital twins—virtual representations of physical assets and processes that act as a bridge between physical reality and digital intelligence.

This paper makes the following contributions:

- It proposes a **three-layer architecture** that connects LLMs with digital twins and physical automation systems, enabling intelligent and adaptive control in industrial environments.
- It distills **three design paradigms** for representing dynamic system information in digital twins to support effective LLMs interaction: **system snapshot modeling**, **event-log-based modeling**, **planning-based modeling**.
- The architecture and design are demonstrated with case studies in realistic use cases, and we propose the concept of "**Return on Intelligence**" as a conceptual tool for evaluating the efficacy of investments in intelligent automation.
- It outlines practical strategies for addressing common challenges associated with LLMs, such as hallucination, by applying **multi-agent design**, **human validation**, **and test-driven development**.

This paper is presented with the following structure: Section II introduces the fundamental problem and the conceptual architecture. Section III introduces the 3-layer system architecture consisting of the physical, digital twin, and cognition layers. Section IV presents the design paradigms in greater depth, focusing on how information is modeled to support LLM reasoning. Section V showcases case studies that demonstrate practical implementations of the proposed architecture. Section VI discusses key insights, design considerations, and practical recommendations based on the case studies. Finally, Section VII concludes the paper with a summary and future work.

II. Conceptual Architecture

This section outlines the fundamental problem and presents high-level guiding ideas to form the solution.

A. The fundamental problem

LLMs operate within a digital environment and lack direct grounding in the physical world. Their logical reasoning is primarily based on semantic relationships and knowledge patterns derived from textual data, making them inherently disconnected from real-world processes. Consequently, LLM cannot directly influence physical systems without additional mechanisms or interfaces.

To address this issue, three fundamental challenges must be resolved:

1. Converting real-world information into data representation interpretable by LLM.
2. Establishing connections between the LLM outputs and actionable operations capable of influencing other systems.
3. Integrating LLM-driven reasoning to automate task as requested by users.

B. *Conceptual Architecture*

To tackle these issues, we propose a three-layer conceptual architecture that provides a bridge between digital intelligence and physical automation (Figure 1). Each layer focuses on a distinct aspect of the system:

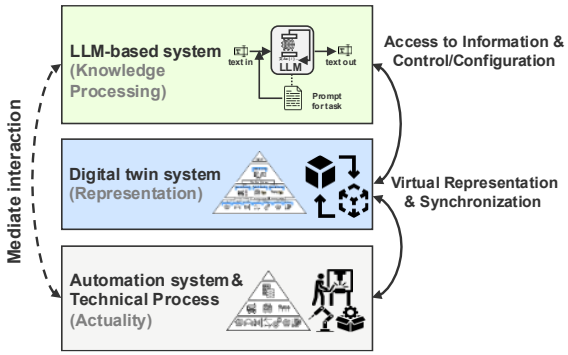

Figure 1 Three-Layer Architecture for LLM-Integrated Automation Systems

- **Automation System & Technical Process** serve as the foundational layer, responsible for monitoring and executing real-world industrial operations through machinery, sensors, actuators, and controllers.
- **Digital Twin System** acts as a virtual representation of the physical system, this middle layer maintains real-time synchronization with the automation layer. It structures representation that can be consumed by the LLM, while also receiving and mediating control or configuration commands back to the physical system.
- **LLM-based System** processes structured representation inputs provided by the digital twin. It performs reasoning, process represented knowledge, and generates outputs such as control commands, plans, or configuration settings. These outputs are then translated into executable actions through the digital twin interfaces.

Overall, the digital twin system serves as the bridge to ground the LLM to the physical reality.

III. System Architecture

This section explains the technical necessities for system design, introduces the components required for its implementation, and illustrates them with various system diagrams with different levels of detail and focus. Figure 2 shows the system architecture overview.

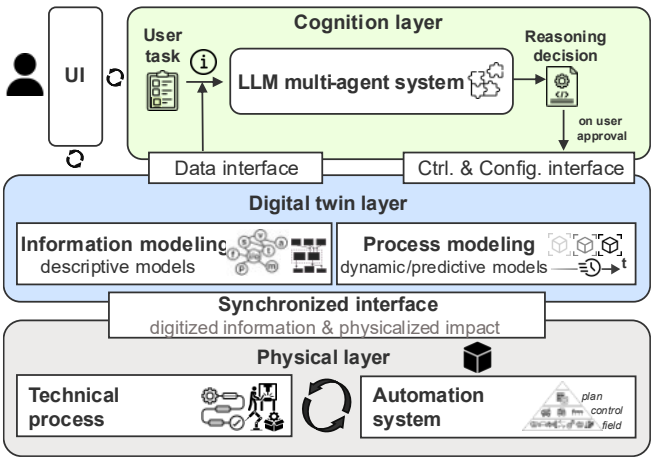

Figure 2 System Architecture Overview

A. *Physical layer:*

A **technical process** [1], which involves operations of materials and machines to achieve industrial objectives. First of all, a technical process can be executed manually, but to increase efficiency, precision, and productivity, an **automation system** is typically employed, reducing manual effort in executing and controlling the technical process.

In most industrial setups, automation follows a hierarchical structure known as the automation pyramid [1]: at the field level, sensors and actuators connect directly to machines. At the control level, Programmable Logic Controllers (PLCs) handle real-time control logic. At the planning level, a Manufacturing Execution System (MES) coordinates higher-level tasks such as production scheduling. These control and planning functions are delegated to specialized software components.

B. *Digital twin layer*

To enable advanced monitoring, control, maintenance and optimization of automated processes, the concept of digital twin is introduced [2]. A digital twin system is a software system that provides virtual model representation of physical assets, processes, or systems that enable real-time synchronization between the digital and physical domains.

To facilitate conceptual understanding and design, we classify digital twin model representations into two distinct types in the scope of this paper:

- **Information modeling** provides a static representation of entities and processes. It captures descriptive knowledge about what exists, including various factors and their relationships.
- **Process modeling**[2], in contrast, is structured around the dimension of time progression or a sequential order of execution. Once executed, these models can simulate dynamic processes and predict system behavior.

This distinction arises from how we model "time". In information modeling, time is treated as one dimension among many dimensions in describing the system's composition and entity relationships. By contrast, process modeling fundamentally organizes the representation around the progression of time.

[1] DIN IEC 60050-351 definition of "technical process": the entirety of interacting operations within a system through which material, energy, or information is transformed, transported, or stored.

[2] Minor note: please note that the word "process" has different contextual meanings in "technical process" (operations within a system) and "process modeling" (way of modeling); "modeling" is an activity of creating "models".

This distinction reflects a deeper philosophical view: our perception of reality is shaped by the experience of time moving forward—also referred to as the "arrow of time"[3]—which influences how humans reason. In [4], the author expresses skepticism about the natural existence of time and argues that it is a measure derived from how humans perceive change. These philosophical foundations are relevant when investigating how knowledge is formed and can be applied to approximate and control reality. Modeling systems based on temporal progression is not just an engineering requirement but also a natural extension of how we form and apply existing human knowledge—patterns that LLMs can also learn and utilize, as evidenced in [5], [6], [7].

Returning from our philosophical detour discussed above, we require a modeling perspective that facilitates a more transparent and analyzable understanding of how the automation system should behave over time. The core objective of (intelligent) automation is to initiate the correct control action at the appropriate time to execute operations in the technical process. To evaluate whether the system is functioning correctly as intended, its actual execution can be compared against the predefined technical process. Figure 3 introduces such a modeling perspective, illustrating the key properties and elements of the proposed system architecture.

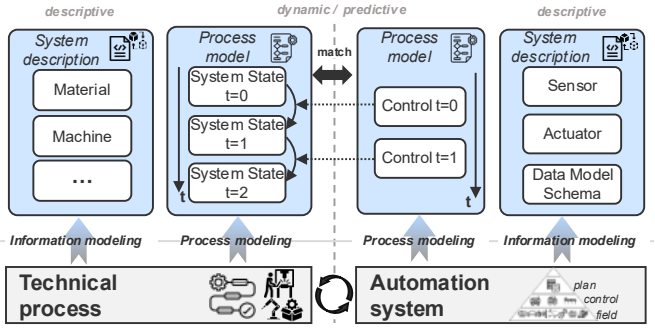

Figure 3 Representing Technical Processes and Automation Systems via Information and Process Modeling

On the *Technical Process* side (Figure 3 left), the Information Modeling provides a static description of the system, describing materials, machines and other interested entities. The process modeling structures the interested physical parameters into a sequence of system states (t=0, t=1, t=2, ...) that can represent how the process evolves over time.

On the *Automation System* side (Figure 3 right), the process modeling provides the representation of how the process is executed by applying control actions at each state. The Information Modeling captures the system's composition and functions, including sensors, actuators, and data models, which provide static knowledge about the automation system.

The central "match" between the two process models signifies that the automation system's control actions must align with (and execute) the time-based plan defined by the technical process. Meanwhile, the information modeling on each side provides the descriptions and relationships necessary for interpreting the process models.

C. Cognition layer: LLM-based system

By having access to digital twins that provide detailed data and knowledge about the overall system, the LLM obtains the necessary information to perform reasoning tasks. A typical simplified integration mechanism is illustrated in Figure 4. Within this mechanism, the LLM is implemented as a software component (referred to as the LLM agent) responsible for processing information to accomplish a specific task. It receives system information through a data interface, then processes this information based on the instruction of a structured prompt template that guides the model's behavior in generating output. The resulting output from the LLM is then parsed into structured text or executable code, which can be subsequently handled by the control or configuration interface provided by the digital twin software. This interface enables LLM's reasoning outcomes to influence and interact with other connected systems.

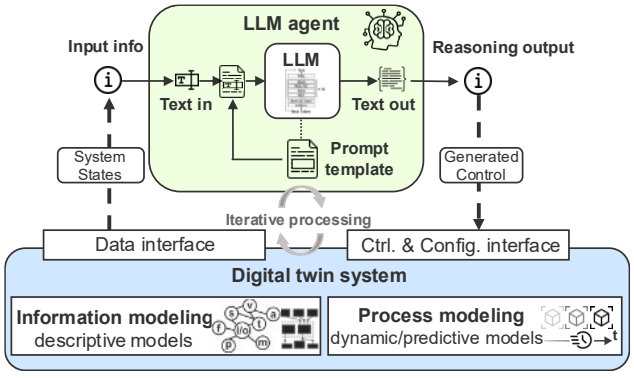

Figure 4 LLM Agent Integration with Digital Twin for Reasoning

IV. Design Paradigms

This section provides a more detailed look at design paradigms for representing information in the digital twin layer to support LLM reasoning, as well as task-solving in the cognitive LLM layer.

A. Digital Twin Design: aiming for LLM integration

Empirical investigation in [8] indicate that LLMs learn and reason with descriptive knowledge and procedural knowledge in different ways. Information provided by the information modeling (descriptive knowledge) and the process modeling (procedural knowledge) offers the contextual foundation required for LLM-based task execution. However, a key challenge lies in designing operational mechanisms that enable LLMs to effectively utilize this knowledge.

1) Time-based dynamic mechanisms modeling

Technical processes evolve continuously over time as real-world transformations unfold, with each transformation requiring specific decisions or control actions (cf. Figure 3). From a modeling perspective, this can be represented as a series of discrete steps (t=0, t=1, t=2, …), each capturing the relevant system states information.

In practice, this can be implemented using state machines or system snapshots that record key system properties at specific moments. The LLM iteratively receives the updated system state—such as sensor readings or process status—as input. Since most LLMs operate on textual prompts, the digital twin must convert each updated system state into a structured text or code format (such as JSON). For each LLM invocation, the prompt is updated with the latest information provided by the digital twin software.

This enables the LLM to reason over dynamically updated system information and determine the next appropriate

control action. The process is iterative, and further details are presented in the following section.

2) The three design paradigms

Building on empirical insights from our previous research and other related literature, we synthesize three primary paradigms for representing evolving system information and integrating it into LLM-based control: system snapshot modeling [9], event log-based modeling [10], [11], and planning-based modeling[12]. Each approach provides a distinct mechanism for capturing dynamic system behavior, offering different design paradigms for LLM integration, as symbolically illustrated in Figure 5:

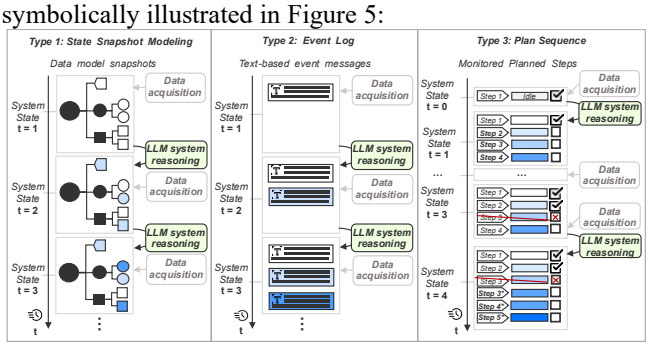

Figure 5 Three Design Paradigms for Structuring System Information to Support LLM Reasoning

Type 1: System Snapshot Modeling (State Modeling)

This mechanism relies on capturing a sequence of snapshots of the system's state at discrete time steps (t=1, t=2, t=3). Figure 5 uses colored symbols to illustrate these sequential changes in data structure. Each snapshot can use concrete modeling formats (JSON or a structured text format) to represent the state of the system at that specific moment. The LLM system reasons over these representations to derive control actions for the next state. A case study[3] illustrating this approach can be found in [9].

Type 2: Event-Log-Based Modeling

The information can be modeled in an event log. Planning and control rely on two key prerequisites: time and information. These can be technically captured in the form of an event. In this approach, the system state is represented as a chronological sequence of event messages. These logs contain textual information such as system activities, status updates, or triggering events recorded over time. The LLM system interprets this temporally ordered sequence to analyze context and determine appropriate next actions. A case study[4] illustrating this approach can be found in [10], [11].

Type 3: Planning-Based Modeling

In this approach, system states are defined in advance as part of a structured plan that outlines a sequence of predefined steps required to complete a task. The LLM system monitors the progress of the plan and compares the current execution status against the expected status. When discrepancies or failures are detected (e.g., a step fails or becomes infeasible), the LLM can propose a revised plan (replanning). A case[5] illustrating this approach can be found in [12].

Commonality: Temporal Discretization of Information

Although these three types involve different design patterns, they share a common constraint shaped by the operational nature of state-of-the-art LLMs (i.e., prompting and response): to be processed by the LLM, information must be discretized in time and serialized into text as part of the prompt during each reasoning cycle (cf. Figure 4).

In Type I, each system state is captured in a snapshot model at regular time intervals and serialized into text for insertion into the LLM agent's prompt. In Type II, information is modeled as events: the system generates event entries in chronological order, the LLM agent fetches relevant information from the event log, and its outputs are also recorded as events. In Type III, the agent generates a plan covering a limited future scope. A common feature across these mechanisms is their ability to encapsulate information within discretized time slots.

B. LLM System Design: aiming for flexible task automation

This subsection presents a structured approach to designing LLM-based systems for flexible task automation, while addressing common challenges such as task complexity, hallucination and reliability.

1) Task-solving as core function

A key advantage of integrating LLMs into industrial automation is their ability to solve tasks flexibly. These tasks typically originate from user needs or from events that arise during runtime and require a response. A user interface is required to connect the LLM-agent layer with the digital twin system, enabling both real-time system monitoring and task-oriented interaction.

A typical setup uses a front-end application with a chatbot interface, allowing users to dispatch tasks, view system status, and receive results. This arrangement enables intuitive, text-based interaction with processes that may be too intricate for the user, while the LLM system performs on-demand reasoning and interprets detailed technical logic.

2) Challenge of task complexity

In practical scenarios, depending on the use case, tasks can be too complex for a single LLM agent to handle effectively, often leading to reduced accuracy. A solution to this challenge is task decomposition combined with a multi-agent system design (cf. Figure 6). In this approach, the task is divided into manageable sub-tasks, which are then assigned to multiple LLM agents. Each agent specializes in a specific aspect of information processing and reasoning, thus improving overall performance and accuracy. The computational complexity of the multi-agent system shall generally scale proportionally with the complexity of the original task—that is, more difficult tasks require longer reasoning processes and more generated text by the agents.

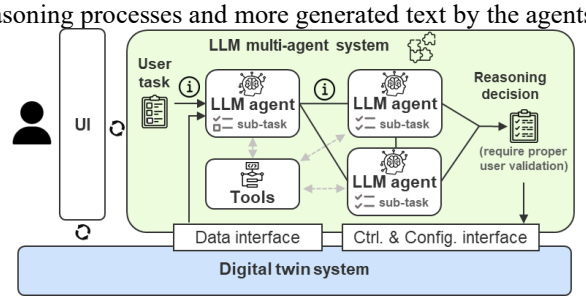

Figure 6 Multi-Agent Design and Tool Integration of LLM System

[3] State Modeling: GitHub: LLM interacts with simulation models
[4] Event-Log-Based: GitHub: LLM controlled automation based on event messages
[5] Planning-Based: GitHub: LLM agents plan flexible production tasks

It is important to note that not all subtasks need to be handled by LLM agents. Some operations, such as mathematical calculations, information retrieval, pathfinding, or executing specialized code for software functions and robotic skills, can be performed more efficiently and reliably by dedicated software components, also known as "Tools" (cf. Figure 6). To improve overall task-solving performance, LLM agents can be designed to integrate with these tools and delegate specific functions accordingly.

3) "Hallucination" and task reliability requirements

Hallucination is a broad term referring to cases where an LLM generates incorrect or undesired outputs. Some studies categorize hallucinations as either model-intrinsic or model-extrinsic[13]; however, this distinction alone does not provide actionable guidance for mitigation and solution in terms of system engineering. Based on our investigation and practical experience, we identify four specific types of hallucinations, each with distinct underlying causes and corresponding countermeasures.

1. **Insufficient Model Knowledge:** The model lacks the required knowledge patterns or semantic associations for accurate reasoning. This limitation can stem from various sources, including technical limitations in the model's training process, the inherent difficulty of representing certain domain-specific knowledge in text form, or gaps in the available human knowledge captured in the training data.
   Countermeasures**:**
   - Choose a more capable base model.
   - Fine-tune the model with domain-specific data.
   - Integrate external software components for specialized tasks (e.g., RAG for information lookup [14], [15], or tool-using [16], [17], [18]).
2. **Misalign with intended task and user preference:** Models are typically trained on broad datasets and fine-tuned for varied tasks; they may not match the specific goals or style preferences of a particular context. A correct answer is not necessarily a useful one if it does not align with the user's intent. Additionally, imprecise or poorly formulated prompts can create a mismatch between the task and the expected output, preventing the LLM from performing accurate and relevant reasoning.
   Countermeasures:
   - Select and fine-tune models that align more closely with the domain and task.
   - Provide clearer and instructive prompts.
   - Iteratively improve prompts to adapt LLM agent behavior
3. **Ambiguous and general text input:** If a task is insufficiently articulated or ambiguous, the model cannot execute it with certainty or the required level of specificity. A lack of contextual clarity in the task or question description can result in unpredictable outputs, as the model must infer a direction of reasoning while generating its response.
   Countermeasures:
   - Supply additional context or supporting data.
   - Ask the user for clarifications to remove ambiguities.
4. **Complexity mismatch:** The computational complexity of the reasoning process should generally scale with the complexity of the task [19]. Short reasoning process leads to reduced result accuracy.
   Countermeasures:
   - Decompose large tasks into smaller subtasks using a multi-agent system [12].
   - Apply step-by-step reasoning (Chain-of-Thought [7], ReAct [20]).
   - Extend the reasoning process before the LLM generates a final decision [20], [21].

4) Necessities of UI design in response to reliability issue

While the system can streamline task-solving by reducing human effort in information retrieval, data interpretation, reasoning and content drafting, users must ultimately verify the outputs to assess plausibility and maintain accountability. Outputs with imperfect accuracy may still be useful across different use cases, depending on the specific requirements and the degree to which they reduce human workload. To meet practical reliability requirements, the LLM system shall be limited to functioning as an assistant system, with final result validation still remaining the user's responsibility. Therefore, the user must be an integral part of the system's design, acting as both the initiator and validator of the task-solving process, as illustrated in Figure 6.

V. Case Studies and Applications

This section presents case studies applying the proposed architecture and design paradigms across varied scenarios. The architecture is applied to each use case with different implementation details and technology stacks, prototyped in a lab setting. Each case is accompanied by demonstration videos showcasing how intelligent automated systems may look and operate.

A. Planning-Based Control of a Modular Automation Robotic System (Design Paradigm Type 3)

This case study [12] demonstrates how LLMs can be applied to plan and control a modular robotic automation system. Based on a user-specified task, the system autonomously generates a corresponding production plan and executes it through the underlying automation infrastructure.

In this setup, LLM agents are designed to interpret descriptive information provided by digital twins (implemented with AAS) and control the physical system through structured service interfaces, as illustrated in Figure 7. A key feature of this approach is the **hierarchical control service interface** within the digital twin, organized into two abstraction levels: (1) **coarse-granular skills**, representing higher-level operations at the automation module level, and (2) **fine-granular functionalities**, corresponding to low-level control actions at the component level.

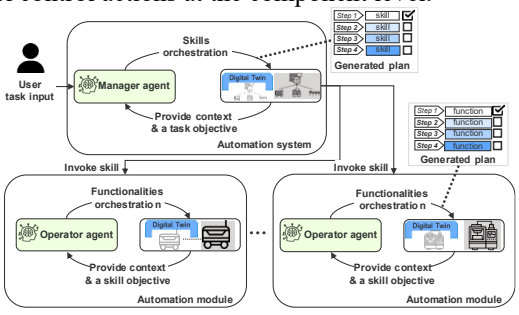

Figure 7 Task Decomposition Mechanism and the Design of Manager and Operator Agents.

The LLM agents decompose the user-defined task and orchestrate a sequential plan combining atomic skills and functions to accomplish the task. The process modeling is **planning-based**: given a user task as input, an LLM agent first generates a high-level production plan consisting of skill invocations across relevant automation modules. Each skill is then further decomposed through LLM reasoning into a detailed sequence of function calls at the component level.

The LLM agents are systematically designed using a manager-operator agent hierarchy. The **Manager Agent** is responsible for interpreting high-level user-defined tasks, decomposing them into machine skills, and distributing these skills across relevant automation modules. When a skill command reaches a specific automation module, an **Operator Agent** takes over, further decomposing the skill into a detailed sequence of executable function calls at the component level.

These plans are executed through the digital twin's service interfaces. The **system state** maintains only the **current execution status** of the plan and supports replanning when triggered by exception events or execution failures.

This approach enables dynamic adaptation and flexible task execution within a modular automation environment, supported by the intelligent reasoning capabilities of LLMs.

B. Event-Driven Control of Industrial Automation System with LLM (Design Paradigm Type 2)

This case study [10], [11] presents an event-driven information modeling approach that continuously provides real-time data updates to LLM agents. This allows the LLM to interpret events within the physical automation environment and make informed decisions to control the system.

A key innovation in this approach is the semantic enhancement of raw data from field components through an event-driven information modeling mechanism. Traditional industrial automation systems often produce data at a low semantic level—such as binary signals from sensors or numeric feedback from actuators—which inherently lack sufficient semantic clarity for meaningful interpretation. To address this limitation, the approach introduces a dedicated Data Observer software component. This Data Observer continuously monitors low-level signals from field components, including sensor states, actuator signals, and controller statuses, and translates these raw data signals into semantically rich textual event descriptions through predefined semantic annotations.

These semantically enhanced events are generated dynamically during system operation and are stored within an event log memory. The event log provides a clear history of system activities, forming a basis for LLM-driven reasoning and control. LLM agents interact with this event log memory by subscribing to specific event notifications, which are generated in real-time by the digital twin middleware upon detecting changes in the automation system or the underlying technical process.

The agent system in this implementation is specifically structured around three defined roles: **Manager Agent**, **Operator Agent**, and **Summarization Agent**. The Manager Agent listens to high-level user commands or event triggers, then generates task plans based on the incoming semantically enriched events. The Operator Agent subsequently executes these plans by translating high-level task directives into executable function calls on the field automation components. The Summarization Agent continuously observes the event log memory, providing concise operational summaries to users for improved transparency and interpretability.

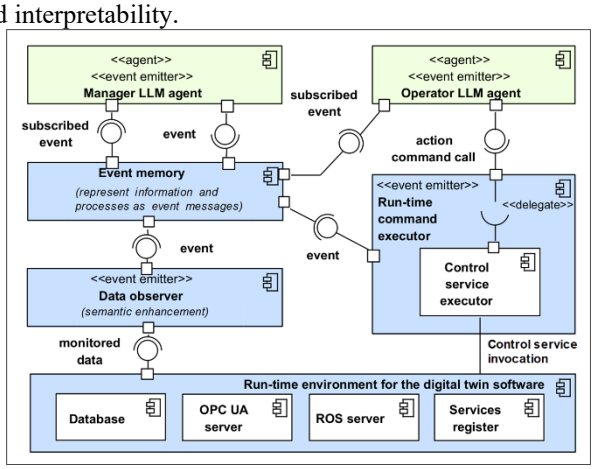

Figure 8 System Component Diagram of the Implementated System for Event-Driven Control of Industrial Automation System with LLM

The concrete implementation was realized using established industrial communication protocols and frameworks such as OPC UA and ROS, as shown in Figure 8. Additionally, the system facilitates structured dataset creation derived from operational event logs, which further supports the supervised fine-tuning of LLM models. This allows improvement of model accuracy, tailoring the LLMs precisely to downstream industrial control tasks.

C. Reasoning on System State Snapshot of Simulated Process (Design Paradigm Type 1)

This case study [9] introduces a specialized multi-agent LLM framework designed explicitly for the autonomous parametrization of digital twin simulation models. Parametrization of technical processes typically requires human knowledge and iterative experimentation. This use case directly addresses that challenge by deploying multiple specialized LLM agents to automatically explore the parameter space of digital twin simulations, identifying feasible and optimal parameter configurations for simulated technical processes.

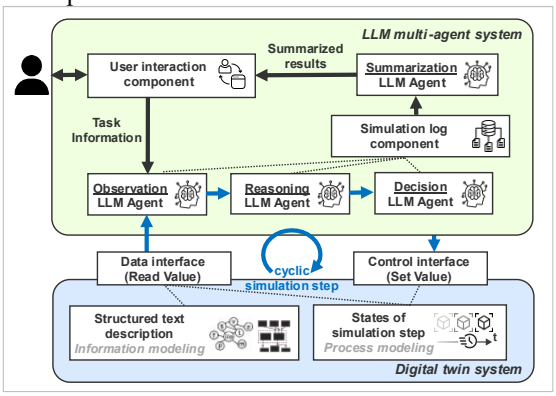

Figure 9 The System Overview of the Use Case: LLM experiments with Simulation Model to Determine Parameter Settings

In this implementation (Figure 9), the technical process is modeled as a simulation composed of discrete execution steps, with each step associated with a corresponding system

state representation. As the simulation progresses, these snapshots are generated incrementally, providing a structured context for agent-based reasoning.

The cognition layer is implemented as an LLM-based system, consisting of four distinct agent roles that operate iteratively within a structured processing pipeline:

- **Observation and Reasoning Agents** process raw data from the ongoing simulation, extract meaningful insights from state snapshots, and apply heuristic reasoning to interpret and analyze the observed states.
- **Decision Agent** generates executable control actions in the form of function calls, dynamically setting the simulation parameters to guide the simulation toward desirable outcomes.
- **Summarization Agent** consolidates agents output, executed control, and other system information, ultimately generating a concise, user-friendly summary report highlighting effective parameter configuration identified during the simulation run.

This case study demonstrates the potential of LLM-based multi-agent systems to autonomously parametrize digital twin simulations. It highlights a promising research direction for integrating LLM intelligent reasoning into process simulation and control.

VI. Discussion

This section discusses key actionable insights from the proposed architecture and use cases, including when LLMs are suitable for industrial automation, how integration depth affects system capabilities, and practical recommendations for system development. These insights help inform future development and investment decisions.

A. Application goals: when to apply llm in intelligent automation?

The application of LLM-based automation is particularly advantageous when the following characteristics are present:

Knowledge-intensive Tasks: Tasks requiring semantic interpretation of textual data and perform reasoning based on existing human knowledge.

Flexibility and On-demand: Tasks requiring adaptability, where rigid automation falls short. LLMs enable systems to respond flexibly to varying, unforeseen demands.

Communicative Tasks: Tasks involving text processing, interaction with user, explanation, insights analysis, recommendation, and report content generation.

B. The concept of "Return on Intelligence"

To support the evaluation of LLM integration in automation systems, we introduce the concept of **"Return on Intelligence",** analogous to "Return on Investment" and illustrated in Figure 10. It refers to the reduction in human effort, compared to conventional systems, resulting from the development of an intelligent automation solution.

For instance, LLM-enhanced systems like intelligent robots, advanced automation, or service chatbots may initially demand higher human effort during the development phase (front-loading of efforts, marked with red in Figure 10). This increased effort can arise from the complexities involved in system design, knowledge representation modeling, and integration processes. However, this investment can be justified by reduced operational effort and greater flexibility during the system's usage phase (marked with green in Figure 10)., especially when users face cognitive challenges or knowledge barriers in accessing and operating automation systems. Furthermore, consolidating knowledge in digital twin and enabling its automated utilization through LLM agents can make the digital twin and LLM a strategic software asset.

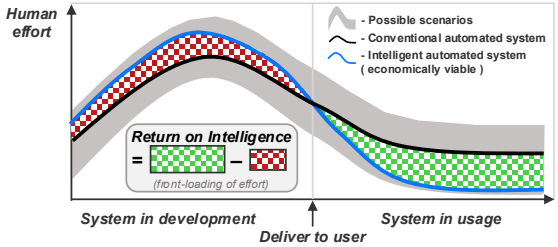

Figure 10 "Return on Intelligence" Concept Principal Illustration. (The trends of the curves do not represent exact quantitative values.)

For systems handling straightforward, repetitive tasks that does not require much intelligence, the LLM might yield Negative "Return on Intelligence", especially in the case that a conventional system based on simple rules and algorithmic logic can have lower development and operation costs and barely requires human intervention.

The human effort and development cost can vary depending on the task use case. For example, building a simple question-answering chatbot may require less effort than developing a system that integrates digital twins or simulation-based task-solving capabilities.

While this paper provides a design architecture and demonstration in a laboratory setup, realizing this "Return on Intelligence" and making LLM integration economically beneficial requires further collaboration between industry and academia for further investigation and to yield evaluation results on specific use cases.

C. Integration Levels of LLM-based Automation Systems

To discuss how the integration of digital twins and automation systems transforms LLMs from passive knowledge processors into intelligent agents capable of real-world interaction and control, it is useful to examine the progressive levels of system integration. The table below compares three stages of LLM integration—standalone usage, combination with digital twins, and full integration with automation systems. Each stage enhances the system's capabilities in data access, modeling, and interaction, reflecting a progressive path toward more autonomous and intelligent automated systems.

Table 1 Integration Levels of LLM-powered Systems

LLM	**LLM + Digital Twins**	**LLM + Digital Twins + Automation System**
External Knowledge & Data Access		
Search engine, text file attachments, tools integration	High-fidelity models that replicate reality	Real-time data acquisition from the physical reality
Modeling		
Textual data	Systematic knowledge and representation modeling (serialized as textual data)	Synchronized data for model update from physical processes
	Integration of simulation models (serialized as textual data)	Feedback to the LLM reasoning process and generated commands
Interaction Capability		
Primarily text-based interaction	Interaction with other systems via digital twin software interfaces	Perception and actuation to interact with physical world

D. Test-driven development

Drawing from our projects experience, we formed a task-centric, test-driven methodology in building, integrating, and validating such systems in LLM projects:

1) Define Tasks and Test Cases

Begin by defining the typical tasks intended for automation. Design typical test cases based on these tasks to evaluate whether the chosen LLM has the baseline knowledge and capabilities to perform them successfully. Automated benchmarking can be first realized using these test cases, and the quality of LLM responses can be further automatically evaluated through LLM-as-a-judge methods [22].

2) Tackle Complexity with Task Decomposition and LLM Agent Design

If a task is too complex for a single LLM, create a structured LLM multi-agent system. Break down the task into smaller, manageable sub-tasks within a task processing pipeline. Assign specialized LLM agents to sub-tasks that match the characteristics described in Section VI.A..

3) Integrate Domain-Specific Knowledge and Tools

If the task requires knowledge and capability beyond the LLM's built-in capabilities, apply supporting methods such as tool usage [16], [17], [18], RAG variants [14], [15]. Some other options are also listed in IV.B.3)., where "hallucination" problem is discussed.

4) Iterative Testing and Refinement

Use iterative testing throughout development to assess system performance, uncover issues, and refine the information processing pipeline. Concentrate the LLM agent's reasoning function on sub-tasks where it is most effective, based on task characteristics (as outlined in VI.A) and observed benchmark outcomes.

VII. Conclusion

This paper presents a three-layer architecture that integrates LLMs, digital twins, and automation systems to enable intelligent task automation in industrial environments. Digital twins serve as the critical bridge between the physical system and cognitive reasoning, allowing LLMs to perceive and influence real-world processes. We propose three design paradigms—state snapshots, event logs, and planning sequences—to structure system information for LLM-driven control. To manage task complexity and improve reliability, we employ multi-agent design and task decomposition strategies. Realistic use cases demonstrate the practical implementation and benefits of the proposed approach. We introduce the concept of "Return on Intelligence" to assess the value of integrating LLMs into automation systems. Moving forward, future efforts could focus on connecting academic research insights with industrial needs, transforming the proposed architecture into more innovative and value-adding applications.

Acknowledgment

This work was supported by Stiftung der Deutschen Wirtschaft (SDW) and the Ministry of Science, Research and the Arts of the State of Baden-Wuerttemberg within the support of the projects of the Exzellenzinitiative II.

References

[1] M. Weyrich, *Industrial Automation and Information Technology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2024. doi: 10.1007/978-3-662-69243-1.

[2] B. Ashtari Talkhestani *et al.*, "An architecture of an Intelligent Digital Twin in a Cyber-Physical Production System," *At-Automatisierungstechnik*, vol. 67, no. 9, pp. 762–782, Sep. 2019, doi: 10.1515/AUTO-2019-0039/PDF.

[3] A. S. Eddington, *The nature of the physical world*, vol. 39, no. 5. Dent, 1928.

[4] J. Barbour, *The End of Time: The Next Revolution in Physics*. Weidenfeld & Nicholson, 1999.

[5] W. Gurnee and M. Tegmark, "Language Models Represent Space and Time," in *International Conference on Learning Representations*, 2023. doi: 10.48550/ARXIV.2310.02207.

[6] I. Dasgupta *et al.*, "Language models show human-like content effects on reasoning tasks," Jul. 2022, doi: 10.48550/arXiv.2207.07051.

[7] J. Wei *et al.*, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., Curran Associates, Inc., 2022, pp. 24824–24837.

[8] L. Ruis *et al.*, "Procedural Knowledge in Pretraining Drives Reasoning in Large Language Models," in *ICLR 2025*, Nov. 2024. [Online]. Available: https://openreview.net/forum?id=1hQKHHUsMx

[9] Y. Xia, D. Dittler, N. Jazdi, H. Chen, and M. Weyrich, "LLM experiments with simulation: Large Language Model Multi-Agent System for Simulation Model Parametrization in Digital Twins," in *2024 IEEE 29th ETFA*, IEEE, Sep. 2024, pp. 1–4. doi: 10.1109/ETFA61755.2024.10710900.

[10] Y. Xia, J. Zhang, N. Jazdi, and M. Weyrich, "Incorporating Large Language Models into Production Systems for Enhanced Task Automation and Flexibility," *Automation 2024*, Jul. 2024, doi: 10.51202/9783181024379.

[11] Y. Xia, N. Jazdi, J. Zhang, C. Shah, and M. Weyrich, "Control Industrial Automation System with Large Language Models," Sep. 2024, doi: 10.48550/arXiv.2409.18009.

[12] Y. Xia, M. Shenoy, N. Jazdi, and M. Weyrich, "Towards autonomous system: Flexible modular production system enhanced with large language model agents," *IEEE ETFA*, 2023, doi: 10.1109/ETFA54631.2023.10275362.

[13] Z. Ji *et al.*, "Survey of Hallucination in Natural Language Generation," *ACM Comput Surv*, vol. 55, no. 12, Dec. 2023, [Online]. Available: https://dl.acm.org/doi/10.1145/3571730

[14] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., Curran Associates, Inc., 2020, pp. 9459–9474. doi: 10.48550/arXiv.2501.00309.

[15] H. Han *et al.*, "Retrieval-Augmented Generation with Graphs (GraphRAG)," Dec. 2024, doi: 10.48550/arXiv.2005.11401.

[16] B. Paranjape, S. Lundberg, S. Singh, H. Hajishirzi, L. Zettlemoyer, and M. T. Ribeiro, "ART: Automatic multi-step reasoning and tool-use for large language models," Mar. 2023.

[17] Y. Qin *et al.*, "ToolLLM Facilitating Large Language Models to Master 16000+ Real-world APIs," in *The 12th International Conference on Learning Representations*, 2024.

[18] Anthropic, "Model Context Protocol (MCP)." [Online]. Available: https://docs.anthropic.com/en/docs/agents-and-tools/mcp

[19] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters," Aug. 2024.

[20] S. Yao *et al.*, "ReAct: Synergizing Reasoning and Acting in Language Models," in *International Conference on Learning Representations*, 2023.

[21] D. Guo *et al.*, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," Jan. 2025.

[22] J. Gu *et al.*, "A Survey on LLM-as-a-Judge," Nov. 2024, [Online]. Available: https://arxiv.org/abs/2411.15594