LLM Agents Perform Controlled Experiments Using Simulation Models

Yuchen Xia, Michael Weyrich, Nasser Jazdi, Johannes Stümpfle, Johannes Sigel, Akshay Narla
Institute for Industrial Automation and Software Engineering
University of Stuttgart, Stuttgart, Germany
{yuchen.xia | michael.weyrich | nasser.jazdi | johannes.stuempfle | johannes.sigel | akshay.narla}@ias.uni-stuttgart.de

Gavin K. Reynolds*, Anna Jawor-Baczynska†, Pol Llopart‡
AstraZeneca
*Sustainable Innovation & Transformational Excellence (xSITE), Pharmaceutical Technology & Development, Operations
†Chemical Development, Pharmaceutical Technology & Development, Operations
‡Data Analytics & AI (DA&AI), Operations IT
*Macclesfield, UK; ‡Barcelona, Spain
{Gavin.Reynolds | Anna.Jawor-Baczynska | Pol.Llopart}@astrazeneca.com

***Abstract*—Large language models (LLMs) have shown strong capabilities in reasoning, planning, and tool use, but many scientific and engineering tasks require more than plausible text and code generation. They require understanding how a system responds to intervention, which in practice depends on controlled experimentation. In this work, we propose a multi-agent framework that enables LLM agents to conduct controlled experiments with scientific simulation models for pharmaceutical process design. Given a user query and a baseline configuration, the system constructs a structured task representation, designs experiments, executes comparative simulation, interprets the resulting outcomes, and synthesizes evidence-based recommendations for process parameter optimization. By coupling language models with high-fidelity simulation models in an interactive agent framework, the proposed system supports reasoning through intervention, comparison, and observation. As a result, it produces more specific and actionable outputs than language-only reasoning. In an industrial application setting, this advantage is reflected in higher output specificity as well as improved user-rated correctness and helpfulness. Ablation studies and visualized case analyses further demonstrate the effectiveness and practical utility of simulation-integrated experimental reasoning.**

***Index Terms*—LLM, multi-agent system, simulation, tool-augmented reasoning, AI for science, process optimization**

I. Introduction

Large language models (LLMs) have shown promising capabilities in multi-step reasoning, planning, and tool use. However, many scientific and engineering tasks require more than plausible text generation. They require determining how a system responds to intervention, which in practice means reasoning through controlled comparison rather than through language generation alone.

Controlled experimentation is a central mechanism of scientific inquiry and engineering problem-solving. To understand how a process should be improved, one typically formulates a hypothesis, varies a selected factor while keeping other conditions fixed, observes the resulting change, and compares it against a reference condition. This logic is essential for identifying causal effects and for producing conclusions that are specific, testable, and actionable.

Current LLM-based systems do not naturally operate in this mode. Even when they are equipped with external tools, they are often used to retrieve information or execute isolated functions, rather than to carry out structured experimental comparison. As a result, their outputs may remain suggestive rather than evidential, especially in tasks where reliable conclusions depend on comparing outcomes under controlled intervention.

This issue is particularly relevant in scientific and industrial applications, where important knowledge is often embodied in simulation models that capture system dynamics under varying operating conditions. In pharmaceutical process design, for example, simulation models can provide a practical basis for evaluating candidate process modifications and their consequences.

Motivated by this setting, we investigate how LLMs can be placed in a simulation-based experimental environment for scientific reasoning. Rather than treating simulation as a passive auxiliary tool, we consider it as an environment in which hypotheses can be tested through controlled intervention and comparison. This perspective provides the basis for the framework developed in this work.

II. Related Work

A. Related application fields and use cases

Across scientific and engineering domains, LLMs have increasingly been applied to domain-specific workflows. In chemistry, prior work has demonstrated tool-augmented chemical reasoning in ChemCrow [1], autonomous experiment design and execution in Coscientist [2], and end-to-end synthesis development in LLM-RDF [3]. In simulation-centered engineering settings, recent systems have supported OpenFOAM-based computational fluid dynamics setup and refinement in OpenFOAMGPT [4], natural-language-to-CFD automation in

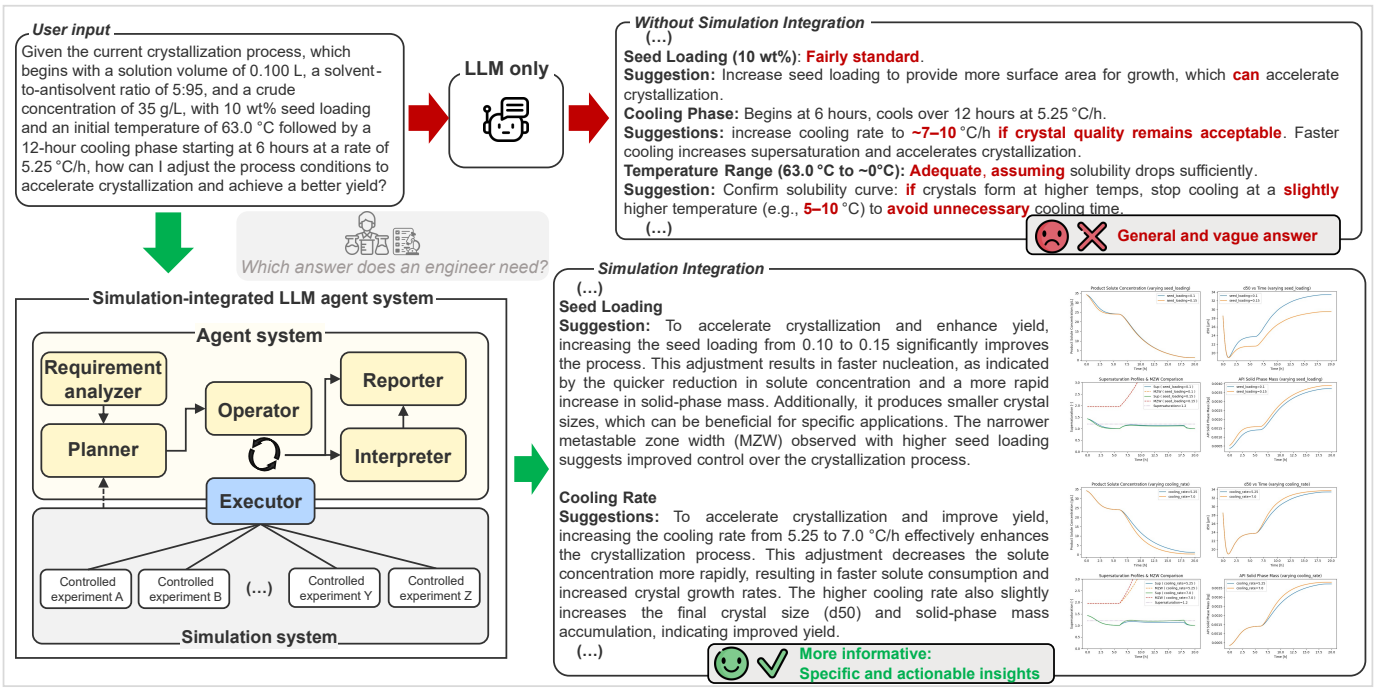


Fig. 1. Graphical abstract. Simulation-integrated agent system delivers precise, evidence-based insights to support engineering decision-making.

AutoCFD [5], and broader end-to-end simulation research workflows in ASA [6].

These studies demonstrate the growing applicability of LLMs to domain-specific scientific and engineering tasks. However, most existing systems emphasize retrieval, workflow automation, code generation, or simulation setup, rather than using simulation models as an environment for controlled experiments and comparative reasoning. As a result, simulation is typically treated as a tool for task execution, not as an experimental substrate for testing hypotheses through variable intervention and outcome interpretation.

B. Reasoning mechanism and agent framework

Prior studies have shown that LLMs can perform iterative reasoning and interact with external tools, as in ReAct [7], Toolformer [8], and Gorilla [9]. Multi-agent frameworks such as AutoGen [10], HuggingGPT [11], and CAMEL [12] assign functional roles to agents for solving general and domain-specific tasks. Communicative Agents [13], [14] further show how multi-round collaboration can improve performance in structured workflows.

More recent work has emphasized explicit role decomposition and the separation of planning from execution [15], as in ConAgents [16] and Plan-And-Act [17]. Reliable tool use has also been improved through clearer and more standardized tool descriptions, as in EASYTOOL [18]. These works provide important design principles for building structured LLM agent systems. However, they do not directly address how such agent architectures can be organized around controlled experimental reasoning over scientific simulation models.

C. Simulator and tool integration

A growing body of work has explored the integration of LLMs with tools, software environments, and simulation-related components. Existing systems have shown that LLM agents can access tools [19] and Web APIs [20], and can be connected to simulation-oriented workflows such as OpenFOAM-based CFD environments [4] and broader automated simulation research pipelines [6]. Multi-agent systems have also been used to formulate, execute, and validate physics-based simulations, for example in mechanics problems in MechAgents [21] and in protein design and analysis in ProtAgents [22].

At the same time, many simulator-related LLM studies remain situated in simplified, embodied, or sandbox-style environments, including 3D simulators [23], rule-based game settings [24], and virtual environments used for behavioral exploration [25], [26]. In such settings, simulation often serves as a testbed for action generation or planning behavior rather than as a high-fidelity scientific environment for controlled comparison and evidence-grounded reasoning.

D. Contributions beyond prior work

The main contributions of this work are as follows:

Scientific reasoning as structure. The proposed agent architecture organizes reasoning according to a scientific problem-solving paradigm. The system decomposes a task into subgoals and requirements, designs controlled experiments to test hypotheses, observes the resulting outcomes, and synthesizes these observations into conclusions. This structure supports systematic and verifiable reasoning for engineering applications.

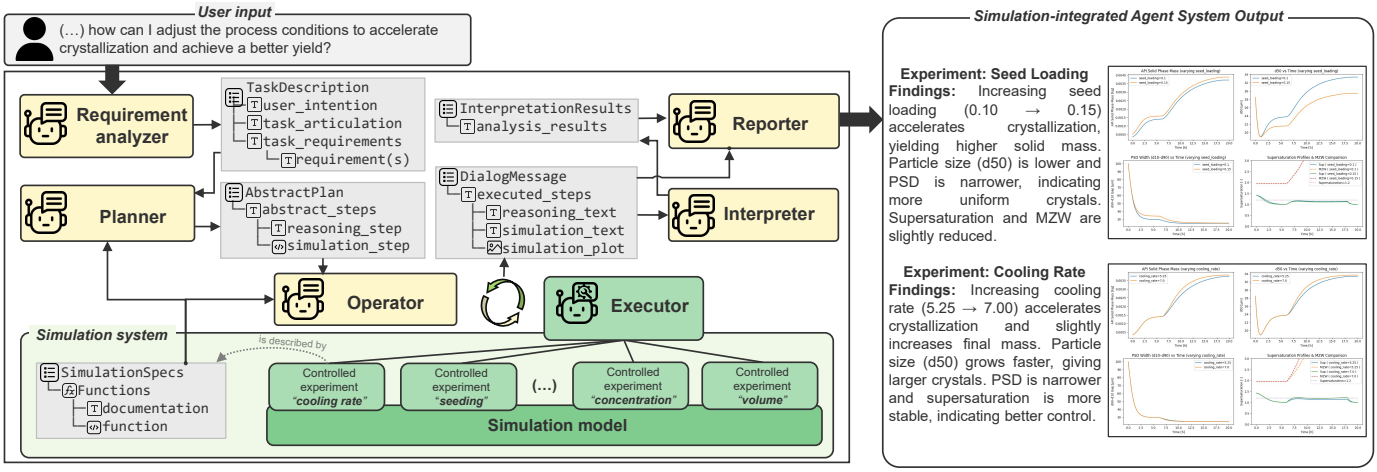


Fig. 2. Information flow in the proposed simulation-integrated agent framework.

Scientific simulation. While prior work has demonstrated the feasibility of grounding LLMs with simulation software, often in simplified or game-like environments, our framework is designed to interact with high-fidelity scientific simulations that reproduce dynamic physical and chemical phenomena. This enables the system to draw on complementary knowledge from both the simulation model and the language model.

Application-driven design. The framework is developed under real-world application constraints. By formalizing the reasoning process as a directed graph, the system provides clear observability and visualization of intermediate reasoning artifacts, which supports practical industrial use.

III. Method

Fig. 2 illustrates the proposed simulation-integrated agent framework. The system processes a user query through a pipeline of specialized agents and generates recommendations for engineer users.

The overall design is organized as a structured information-processing workflow in which intermediate reasoning artifacts are explicitly generated, transformed, and passed between agents. In the context of this work, this workflow enables the LLM-based system to carry out controlled experimental reasoning: it analyzes the task, identifies relevant intervention variables, plans simulation-based comparisons, executes parameterized experiments, interprets the outcomes, and synthesizes the results into a final recommendation.

A. Agent Framework Design

The proposed framework consists of six distinct agents. Five of them are LLM-driven agents, each guided by a dedicated prompt, while one, the Executor Agent, is implemented as a rule-based software component. Each agent is responsible for a specific functional role within the overall pipeline.

The Requirement Analyzer Agent receives the user input, interprets the underlying user intention, and reasons over it to produce a task articulation together with a structured list of requirements. These outputs jointly form the task description, which serves as the basis for downstream reasoning.

The Planner Agent takes the task description as input and performs step-by-step reasoning to generate an abstract plan without committing to concrete execution details. The Planner is provided with a list of available simulation functions, referred to as Simulation Specs, where each function is annotated with a concise description. Based on the task requirements and available simulation capabilities, the Planner produces an abstract plan that outlines the logical sequence of reasoning steps and identifies where simulation functions are relevant. The Planner is explicitly instructed to remain at the abstract level and not to generate detailed execution logic or outcomes.

The Interactive Operator Agent receives the abstract plan and concretizes it through detailed reasoning. In particular, it operationalizes those plan steps that require simulation-based comparison. Whenever the Operator reaches a step involving a simulation function, it generates Python code with fully specified input parameters and delegates the execution to the Executor. After receiving the execution results, it resumes subsequent reasoning. In this way, the Operator converts abstract reasoning steps into executable controlled simulation experiments.

The Interactive Executor Agent executes modularized simulation functions in a controlled Python interpreter environment. It returns both textual and graphical outputs and interacts directly with the Operator by providing execution feedback. As a deterministic software component, it is responsible for reliable tool execution rather than language reasoning.

The Interpreter Agent semantically interprets simulation-generated plots using a vision-capable LLM. It produces textual insights from visual simulation outputs and summarizes the outcomes of the parameterized experiments conducted through the simulation model.

The Reporter Agent observes the overall reasoning and execution process, aggregates the textual outputs generated by the previous agents, and produces a user-facing response that summarizes both the task-solving process and the final results.

Taken together, these agents organize the system into a

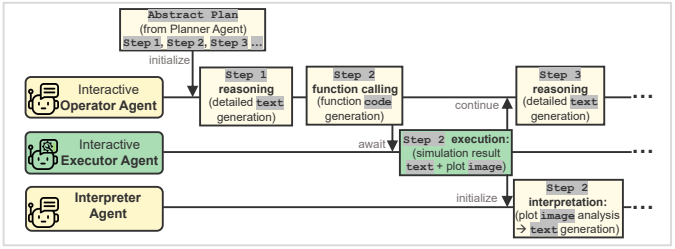


Fig. 3. Operator–Executor–Interpreter interaction protocol during simulation invocation.

structured reasoning pipeline that progressively transforms a user query into explicit intermediate artifacts, simulation-based evidence, and final recommendations.

B. Interactive Interface Between LLM Agents and the Simulation Model

The interface between the agent system and the simulation model is realized through the coordinated interaction of three agents: the Operator, the Executor, and the Interpreter, as shown in Fig. 3. The Operator and Executor form a two-agent dialogue loop for simulation invocation, while the Interpreter analyzes the visual outputs produced by the simulation.

Starting from the abstract plan, the Operator expands each relevant plan step into concrete reasoning text. When a simulation experiment is required, it generates the corresponding function-call code and sends it to the Executor. The Executor runs the code, obtains both textual and visual outputs from the simulation functions, and returns the textual execution results to the Operator, which then continues its reasoning. In parallel, the visual outputs, such as plots, are passed to the Interpreter, which extracts semantic insights in textual form.

This interaction protocol enables an adaptive and fault-tolerant tool-use process. If a function call fails, the deterministic Executor returns an error message. The Operator can then revise the function call and retry generation until the execution succeeds or a predefined retry limit is reached. This design allows the system to recover from malformed or incomplete tool invocations while maintaining the continuity of the reasoning process.

C. Structured Graph-Based Visualization

To systematically observe and analyze the generated content from the proposed system, we visualize the reasoning trajectories, as shown in Fig. 4. The operation of the proposed system can be represented as a directed graph that captures the flow of reasoning from the initial task description to the execution and interpretation of simulation results. Nodes denote discrete generated artifacts, such as task descriptions, requirements, planning steps, simulation calls, and outputs, while edges capture their logical or procedural dependencies. The graph uses four types of edges to represent the reasoning flow and the relationships between intermediate system artifacts: *derives*, *motivates*, *yields*, and *summarized*. Together, these nodes and edges form the system's reasoning trajectories, providing clear observability and diagnosability.

IV. Experiments

This section evaluates the proposed system from both quantitative and qualitative perspectives. We conduct a comparative analysis to understand how simulation integration, requirement analysis, and the agent-based framework each affect the reasoning process, output specificity, and practical usability.

A. Experimented System Variants

We evaluate four system configurations, with all agent-based variants powered by GPT-4o:

- **Full system (agent + simulation integration + requirement analysis):** complete pipeline
- **No simulation:** same system, but with simulation functions disabled
- **No requirements:** same system, but with the Requirement Analyzer disabled
- **LLM only:** a vanilla LLM (GPT-4o) directly prompted with the user task

These variants allow us to isolate the contribution of integrated simulation, requirement structuring, and the multi-agent workflow.

B. Tasks

All system variants are evaluated on a test set consisting of five task scenarios in pharmaceutical crystallization process design (see Appendix: Reasoning Trajectories). Each task specifies a user-defined baseline experiment configuration together with a task goal.

Given a natural-language user query q with an optimization goal g (e.g., maximizing yield), and a baseline process configuration represented by a parameter vector $\mathbf{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(n)})$, the system determines how to improve the outcome by conducting controlled simulation experiments.

The simulation-integrated agent system is expected to perform the following sequence of operations:

- Identify the elements of the baseline parameter vector $\mathbf{x}$ that are relevant to the optimization goal g
- For each selected element $x^{(i)}$ in $\mathbf{x}$, generate a controlled perturbation

$$x^{(i)'} = x^{(i)} + \delta^{(i)} \tag{1}$$

  while keeping all other elements unchanged, yielding

$$\mathbf{x}' = (x^{(1)}, \ldots, x^{(i)'}, \ldots, x^{(n)}) \tag{2}$$

- Using the simulation function $f(\mathbf{x})$, compare the baseline result $f(\mathbf{x})$ with the perturbed result $f(\mathbf{x}')$
- Summarize the findings from these controlled comparisons and recommend an improved parameter vector $\mathbf{x}^*$ such that $f(\mathbf{x}^*)$ is closer to g

This pipeline reflects the principle of a structured scientific inquiry: hypothesize $\rightarrow$ intervene $\rightarrow$ observe $\rightarrow$ analyze $\rightarrow$ report. It also serves as a fundamental unit for solving more complex optimization problems that require clear and specific conclusions.

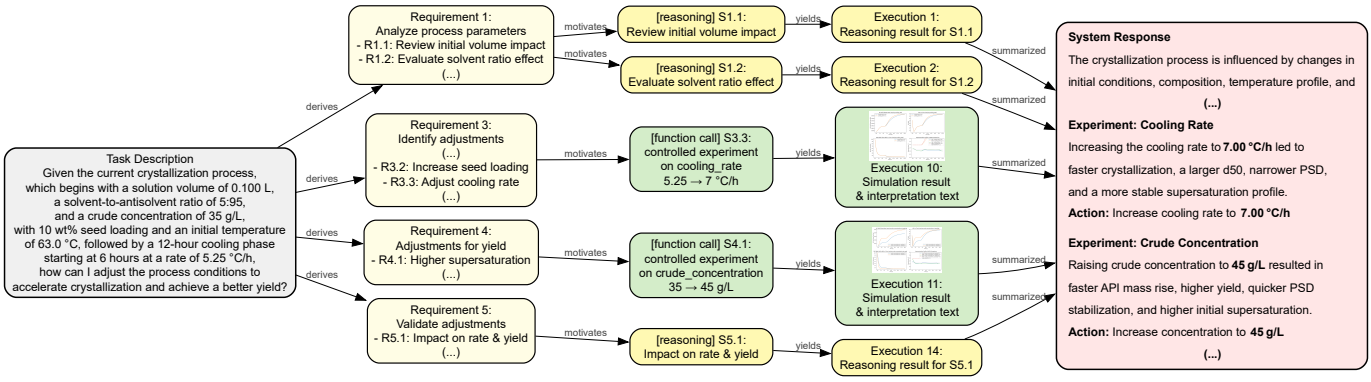


Fig. 4. Graph illustrating the reasoning trajectories from user task to system response.

C. Simulation Model

The simulation model used in this work is a proprietary industrial asset. It is built on first-principles equations combined with semi-empirical formulations calibrated through fitted parameters. The model consists of differential and algebraic equations and is capable of reproducing key system dynamics, including mass transfer, phase transitions, and time-evolving material distributions in a crystallization process.

The model is integrated into the agent system through software interfaces, where simulation capabilities are exposed as callable functions with defined specifications accessible to the LLM agents, as illustrated in Fig. 2.

D. Data Collection

For each combination of system variant and task, we collect the following data:

- Final outputs, together with intermediate reasoning steps and simulation plots. These are further represented as reasoning trajectory graphs composed of nodes and edges (see Fig. 4), with detailed examples provided in Appendix: Reasoning Trajectories
- Evaluation data, including (i) technical metrics that quantify linguistic uncertainty, such as lexical statistics and fuzziness scores, and (ii) user annotations assessing the correctness and practical usefulness of the system outputs

V. Results Evaluation

A. Qualitative Result Comparison

Simulation enables precise, evidence-grounded recommendations. To illustrate the effect of simulation integration on reasoning quality, we compare outputs from system variants with and without simulation integration on a representative task, as shown in the following example:

> **User input:** "Under the current conditions, I want to grow crystals with a median size of 50 μm. What process modifications would achieve this goal?"

When given this user input, a vanilla LLM (powered by GPT-4o without simulation integration) produces a vague and heuristic response, indicating that the answer is fuzzy but safe and therefore has high entropy and uncertainty:

> **Without simulation:** "The initial temperature of 63.0 °C and a cooling rate of 5.25 °C/h may lead to rapid supersaturation and smaller crystals. A slower cooling rate could allow more time for crystal growth. Implementing a slower cooling rate, such as 3 °C/h, may allow more time for crystal growth and potentially achieve larger crystals."

The proposed system generates the following specific, evidence-based answer grounded in the simulation results:

> **Full system:** "A higher cooling rate of 5.25 results in a greater API solid-phase mass, larger median particle sizes (d50), and a narrower particle size distribution (PSD) width compared to a cooling rate of 3.0. The increased cooling rate also leads to higher supersaturation and a larger metastable zone width (MZW), promoting faster nucleation and growth. To achieve a median particle size of 50 μm, consider further increasing the cooling rate beyond 5.25."

This contrast illustrates a shift from vague heuristics to actionable and precise reasoning enabled by simulation integration. A more detailed qualitative illustration of the system reasoning behaviors is provided in the Appendix.

B. Quantitative Evaluation Metrics

To assess how simulation integration enhances language model reasoning, we adopt three complementary evaluation perspectives. First, we measure reasoning quality using two core metrics: specificity and correctness. Second, we evaluate practical usefulness through user-rated helpfulness scores. Third, we assess the simulation-calling behavior of the system, including the precision and recall of simulation calls made by the LLM agents, as well as whether the agents' hypotheses are validated by simulation outcomes.

1) Metrics for Reasoning Specificity: We use two metrics to evaluate reasoning specificity.

- **Linguistic fuzziness analysis:** the frequency per 1,000 words of expressions signaling uncertainty, including vague modifiers, hedging structures, range expressions, and weak logical connectives. Examples include vague modifiers such as "somewhat," "likely," and "may"; hedging structures such as "if," "would," and "could"; range expressions such as "5 to 10 °C"; and weak logical connectives such as "can," "may," and "might".

- **LUCI score [0–1]:** a normalized metric [27] for quantifying linguistic uncertainty. For example, a score of 0.13 indicates that 13% of sentences in a paragraph are marked as uncertain, using the implementation from [28].

2) Metrics for Reasoning Correctness and Usefulness: For correctness and usefulness, we do not use automated evaluation methods such as LLM-as-a-Judge, since LLMs lack real-world experience with the specific scenario knowledge and detailed facts required in this application domain. Standard LLM benchmarks are also too general to be applicable in this context.

We therefore adopt user evaluation. Two senior domain specialists review the full reasoning process, provide commentary, and rate the system outputs based on the following two questions:

- **Correctness [1–5]:** How consistent are the results and intermediate reasoning with your empirical knowledge?
- **Helpfulness / Usefulness [1–5]:** How helpful or useful would the system's output be to an end user in a practical production environment?

The reported correctness and helpfulness scores are averaged over five distinct task scenarios.

3) Metrics for LLM-Agent Invoked Simulation Calls: In the proposed framework, the Planner Agent is responsible for planning simulation steps, while the Operator Agent parameterizes changes to specific input variables and invokes the corresponding simulation functions. This process is evaluated using two types of metrics.

- **Simulation call precision / recall (Sim. P/R):** Precision is defined as the proportion of simulation calls made by the system that are appropriate. Recall is defined as the proportion of necessary simulation calls that were actually made by the system, where necessity is determined through user annotation.
- **Simulation confirmation (Sim. Conf.):** Simulation confirmation measures the proportion of system-generated hypotheses validated by simulation outputs. A hypothesis is considered confirmed if simulation results are consistent with predictions and move the system closer to the optimization goal. For example, if increasing the cooling rate is hypothesized to accelerate crystallization and simulation results show a corresponding increase, the hypothesis is considered confirmed.

C. Evaluation Results with Quantitative Metrics

Table I summarizes reasoning quality metrics across five crystallization tasks, each formulated as an optimization problem. The evaluation includes manual inspection of complete reasoning trajectories represented as graphs, with detailed examples provided in Appendix: Reasoning Trajectories. Specifically, we analyze 17 simulation results generated under the full-system configuration and 30 results from the ablated no-requirement configuration.

As shown in Table I, the Full System outperforms all ablated variants in terms of output specificity, correctness, and helpfulness. Its responses contain the fewest vague expressions, at

TABLE I
REASONING QUALITY METRICS UNDER THE FOUR SYSTEM VARIANTS (MEAN ± STANDARD DEVIATION).

System Variant	Fuzzy Words / 1k ↓	LUCI ↓	Correct ↑	Helpful ↑	Sim. P/R ↑	Sim. Conf. ↑
Full System	13.7 ± 6.7	0.13 ± 0.10	4.1	4.2	94% / 64%	76%
Without Requirement	17.7 ± 5.9	0.17 ± 0.07	3.4	3.8	65% / 69%	45%
Without Simulation	51.3 ± 4.7	0.33 ± 0.09	(< 3.0)	(< 3.0)	N.A.	N.A.
LLM-only	57.1 ± 14.6	0.36 ± 0.11	(< 3.0)	(< 3.0)	N.A.	N.A.

13.7 per 1,000 words, and achieve the lowest LUCI uncertainty score, 0.13, indicating more precise and specific reasoning. This is especially important for scientific and engineering tasks. In contrast, the No Simulation and LLM-only variants contain substantially more vague and uncertain expressions, with LUCI scores of 0.33 and 0.36, respectively. While some of these outputs may not directly contradict known facts, their lack of specificity limits their practical usefulness for actionable decision-making.

D. Experiment Hypotheses Confirmed by Simulation (Sim. Conf.)

The full system achieves 94% simulation call precision, and 76% of its reasoning hypotheses are supported by simulation results. For example, the system may generate a hypothesis such as increasing the cooling rate, execute this as a controlled experiment through the simulation model using perturbed parameter inputs, and obtain outputs that confirm measurable improvement toward the goal. This completes the loop of hypothesis $\rightarrow$ experiment execution $\rightarrow$ validation.

Overall, the full system is positively evaluated by domain specialists, achieving an average correctness score of 4.1 and a helpfulness score of 4.2.

E. Ablation on Requirement Analysis (No Requirement)

Removing the Requirement Analyzer increases simulation recall to 69%, but reduces simulation precision to 65%. This suggests that without explicit reasoning over task requirements, the system tends to overuse available simulation functions and produce less targeted simulation calls. The outputs also become less certain according to lexical analysis, and both correctness and usefulness scores decline. This result aligns with prior findings [7], [17] showing the value of intermediate reasoning steps in complex task solving.

F. Ablation on Simulation Functions (No Simulation)

The weakest performance is observed when simulation functions are removed. In this setting, all agents rely solely on the LLM's prior knowledge, without access to experimental context. As a result, the outputs become significantly less specific, as reflected by high fuzziness and LUCI scores. Domain users regarded scores below 3.0 (< 3.0) as unusable in this setting because the outputs were too vague to support meaningful insight extraction.

G. Comparison with LLM-Only Prompting

When given the same query, the LLM-only baseline also produces overly vague responses, as reflected in high fuzziness

and LUCI scores. Because the outputs generated without simulation integration contain excessive hedging and fuzzy statements, they are considered unverifiable and unhelpful for engineering users.

H. Summary of Quantitative Evaluation

These results support the effectiveness of the proposed system design. First, simulation integration enables more precise and evidence-based reasoning. Second, requirement analysis contributes to better reasoning performance. Third, the agent-based architecture successfully combines the strengths of language models and simulation models, organizing the reasoning process into a traceable and testable workflow that mirrors scientific inquiry. As a whole, this design yields outputs that are not only more specific and informative, but also practically useful for engineering optimization tasks.

VI. Discussion and Generalizable Insights

The overall reasoning quality of the simulation-integrated agent system is governed by a unified principle: approximation fidelity. This principle manifests differently across the two model types. For the simulation model, it refers to physical fidelity, i.e., the degree to which the simulator reproduces real-world dynamics. For the language model, it corresponds to hypothesis correctness, i.e., the extent to which its reasoning proposes correct experimental interventions toward the optimization goal. This unification becomes measurable through whether simulation outcomes confirm the LLM's hypotheses, which can be understood as a form of virtual empirical validation.

The fidelity of the simulation model plays a crucial role. A low-fidelity simulator, such as a game-like environment, may introduce distorted context and misleading details, causing the language model to ground its reasoning in unrealistic or non-existent scenarios and thereby inject noise instead of knowledge. Conversely, a poorly trained LLM may generate incorrect hypotheses and fail to interact meaningfully with the simulation. In both cases, the resulting system lacks the scientific accuracy required for high-stakes engineering applications.

In this application, the developed system is intended as an assistant for decision support. It synthesizes the outputs of both models and presents the agent's reasoning process in a traceable form for the user. The knowledge derived from both models is therefore combined in a complementary way, rather than being dominated by either one. This is important because perfect simulations are rarely achievable, and LLMs do not possess precise training for all specific scenarios. The practical role of the system is therefore not to replace user judgment, but to provide traceable, evidence-grounded support for decision-making under these limitations.

VII. Conclusion

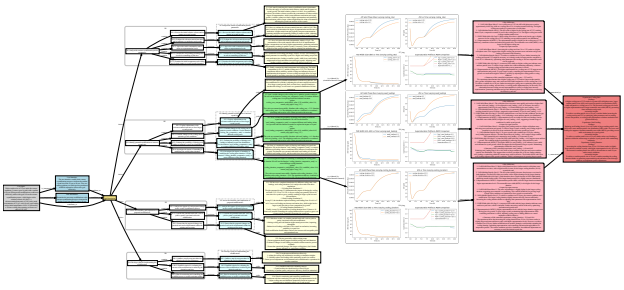
This work presents a simulation-integrated agent system that enables LLMs to reason through controlled experiments for optimization tasks. Rather than directly generating recommendations, the system analyzes the problem, formulates intervention hypotheses, executes simulation-based comparisons, observes the outcomes, and incorporates the resulting evidence into its final conclusions. In this way, the overall reasoning process follows the logic of scientific inquiry: hypothesize, intervene, observe, and report.

The results show that simulation integration improves reasoning quality by making system outputs more specific, more evidence-grounded, and more useful for practical decision-making. By coupling the hypothesis-generation capability of LLMs with the mechanistic fidelity of scientific simulation, the proposed framework provides a minimal and generalizable approach to optimization-oriented reasoning. Demonstrated here in pharmaceutical process design, the framework suggests a broader direction for future LLM systems: stronger reasoning may depend not only on more capable language models, but also on how they are connected to structured sources of experimental evidence.

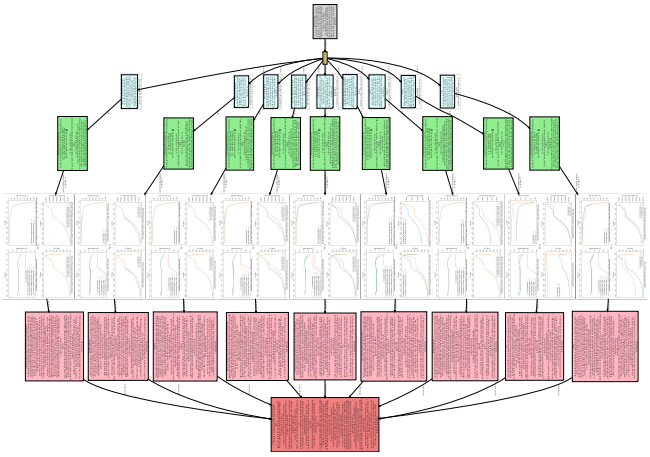
Appendix: Reasoning Trajectories

The reasoning trajectories are automatically generated digital artifacts rendered as Scalable Vector Graphics (SGV-images). For clear inspection, please view them in PDF format and zoom in to inspect the details.

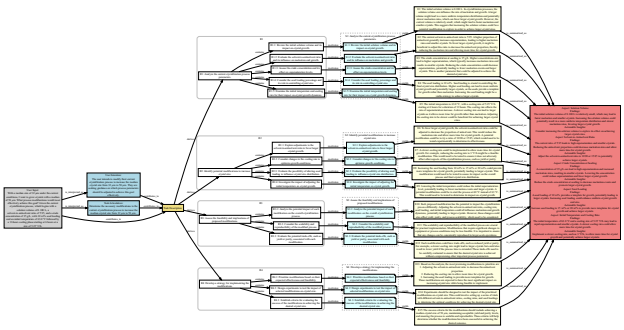
Task Sample 1 (Full system)



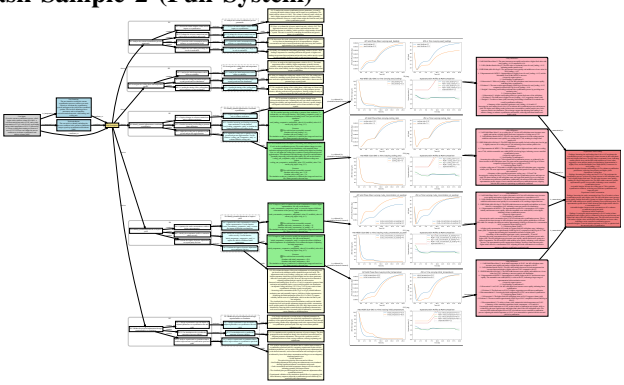
Task Sample 1 (Ablation: No Requirement Analysis)



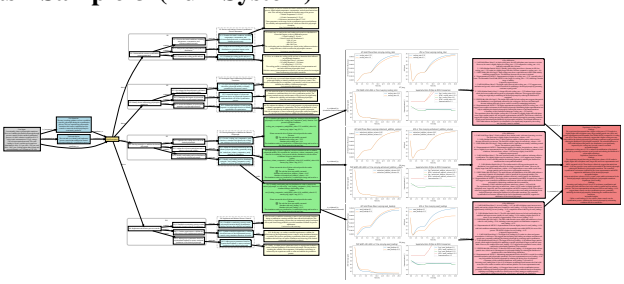
Task Sample 1 (Ablation: No Simulation integration)



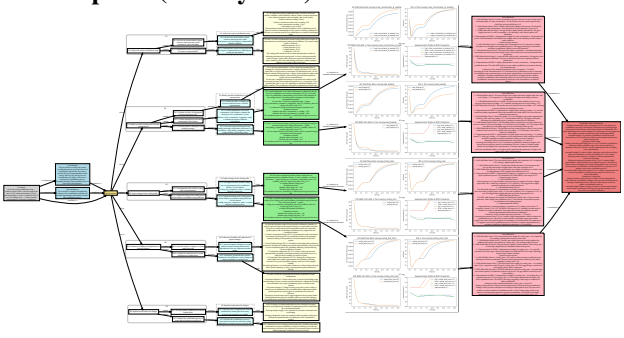
Task Sample 2 (Full System)



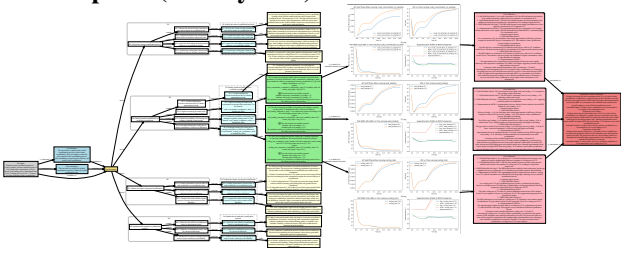
Task Sample 3 (Full System)



Task Sample 4 (Full System)



Task Sample 5 (Full System)



REFERENCES

[1] A. M. Bran *et al.*, "Augmenting large language models with chemistry tools," *Nature Machine Intelligence*, 2024.

[2] D. A. Boiko *et al.*, "Autonomous chemical research with large language models," *Nature*, 2023.

[3] Y. Ruan *et al.*, "An automatic end-to-end chemical synthesis development platform powered by large language models," *Nature Communications*, 2024.

[4] S. Pandey *et al.*, "Openfoamgpt: A retrieval-augmented large language model (llm) agent for openfoam-based computational fluid dynamics," *Physics of Fluids*, 2025.

[5] Z. Dong *et al.*, "Fine-tuning a large language model for automating computational fluid dynamics simulations," *Theoretical and Applied Mechanics Letters*, 2025.

[6] Z. Liu *et al.*, "Toward automated simulation research workflow through llm prompt engineering design," *Journal of Chemical Information and Modeling*, 2025.

[7] S. Yao *et al.*, "React: Synergizing reasoning and acting in language models," in *ICLR*, 2023.

[8] T. Schick *et al.*, "Toolformer: Language models can teach themselves to use tools," in *NeurIPS*, 2023.

[9] S. G. Patil *et al.*, "Gorilla: Large language model connected with massive apis," in *NeurIPS*, 2024.

[10] Q. Wu *et al.*, "Autogen: Enabling next-gen llm applications via multi-agent conversation framework," in *COLM*, 2024.

[11] Y. Shen *et al.*, "Hugginggpt: solving ai tasks with chatgpt and its friends in hugging face," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2023.

[12] G. Li *et al.*, "Camel: Communicative agents for "mind" exploration of large language model society," in *Advances in Neural Information Processing Systems*, 2023.

[13] C. Qian *et al.*, "Chatdev: Communicative agents for software development," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

[14] Y. Xia *et al.*, "Towards autonomous system: flexible modular production system enhanced with large language model agents," in *2023 IEEE 28th ETFA*, 2023.

[15] Y. Xia, "Integrating large language model agents with digital twins for industrial autonomous systems," 2026.

[16] Z. Shi *et al.*, "Learning to use tools via cooperative and interactive agents," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

[17] L. E. Erdogan *et al.*, "Plan-and-act: improving planning of agents for long-horizon tasks," in *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

[18] S. Yuan *et al.*, "Easytool: Enhancing llm-based agents with concise tool instruction," in *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2025.

[19] Y. Zhao *et al.*, "Let me do it for you: Towards llm empowered recommendation via tool learning," in *Proc. SIGIR 2024*, 2024.

[20] Y. Qin *et al.*, "Toolllm: Facilitating large language models to master 16000+ real-world apis," in *The Twelfth International Conference on Learning Representations*, 2024.

[21] B. Ni *et al.*, "Mechagents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge," *Extreme Mechanics Letters*, 2024.

[22] A. Ghafarollahi *et al.*, "Protagents: protein discovery via large language model multi-agent collaborations combining physics and machine learning," *Digital Discovery*, 2024.

[23] W. Huang *et al.*, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*, 2022.

[24] E. Akata *et al.*, "Playing repeated games with large language models," *Nature Human Behaviour*, vol. 9, pp. 1380–1390, 2025.

[25] G. Wang *et al.*, "Voyager: An open-ended embodied agent with large language models," *Transactions on Machine Learning Research*, 2024.

[26] Y. Xia *et al.*, "Llm experiments with simulation: Large language model multi-agent system for simulation model parametrization in digital twins," in *2024 IEEE 29th ETFA*, 2024.

[27] V. Vincze, "Uncertainty detection in natural language texts," Ph.D. dissertation, University of Szeged, 2014.

[28] B. S. Meyers, "Luci: Linguistic uncertainty classifier interface," GitHub repository, 2017.