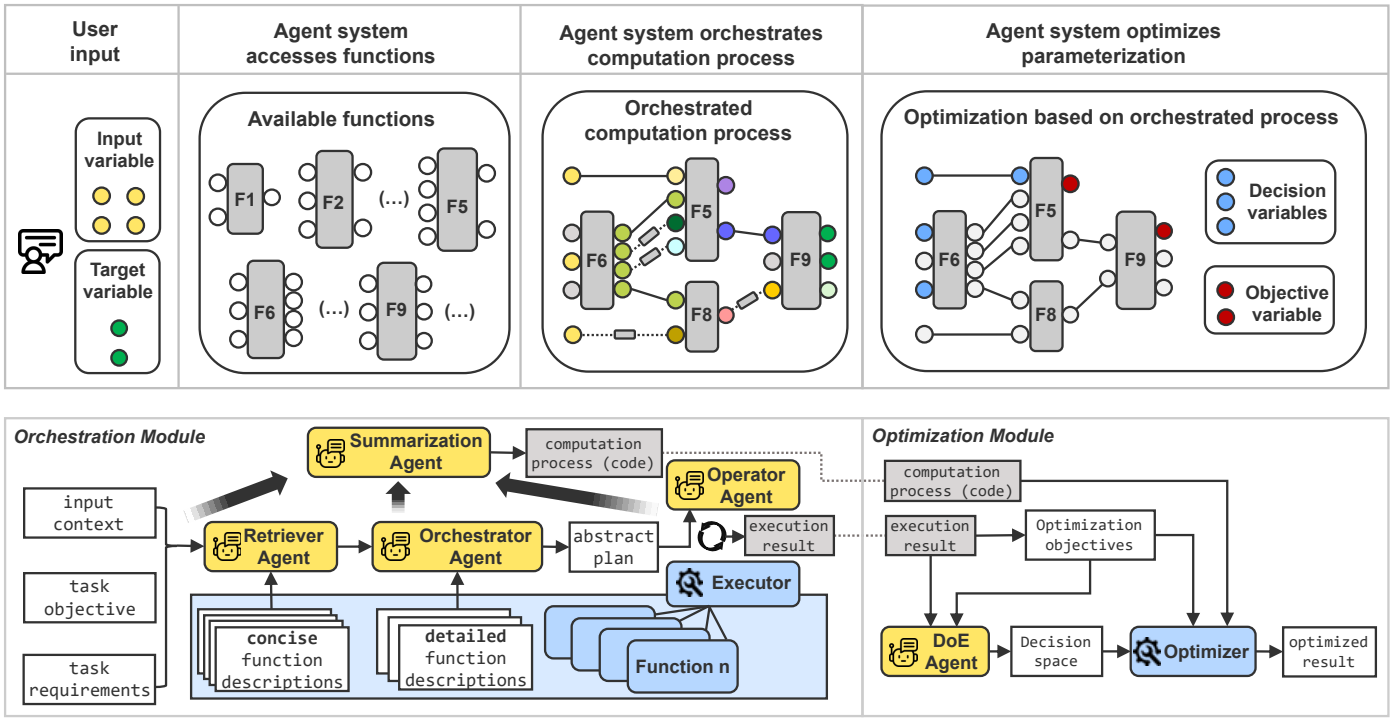
Graphical Abstract

Simulation-Integrated LLM-Agent System for Computation Orchestration and Parameter Optimisation in Process Engineering

Yuchen Xia, Michael Weyrich, Nasser Jazdi, Johannes Stümpfle, Johannes Sigel, Akshay Narla, Gavin K. Reynolds, Anna Jaworr-Baczynska, Matthew Molloy, Pol Llopart



Simulation-Integrated LLM-Agent System for Computation Orchestration and Parameter Optimisation in Process Engineering

Yuchen Xia[a,*], Michael Weyrich[a,*], Nasser Jazdi[a], Johannes Stümpfle[a], Johannes Sigel[a], Akshay Narla[a], Gavin K. Reynolds[b,*], Anna Jawor-Baczynska[c], Matthew Molloy[d], Pol Llopart[e]

[a] *Institute for Industrial Automation and Software Engineering, University of Stuttgart, Pfaffenwaldring 47, 70569, Stuttgart, Germany*
[b] *Sustainable Innovation & Transformational Excellence (xSITE), Pharmaceutical Technology & Development, Operations, AstraZeneca, Macclesfield, United Kingdom*
[c] *Chemical Development, Pharmaceutical Technology & Development, Operations, AstraZeneca, Macclesfield, United Kingdom*
[d] *Global Product Development, Pharmaceutical Technology & Development, Operations, AstraZeneca, Macclesfield, United Kingdom*
[e] *Data Analytics & AI (DA&AI), Operations IT, AstraZeneca, Barcelona, Spain*

Abstract

Large language models demonstrate strong capabilities in reasoning over text-based knowledge, but they remain disconnected from the mechanistic and numerical knowledge embedded in simulation models. This limitation is critical in engineering problem-solving, where reliable answers often require verifiable computation rather than volatile text generation alone. This paper presents a simulation-integrated LLM-agent framework for orchestrating computation processes and optimising process parameters to solve engineering problems. Given a natural-language input describing an engineering problem, the system constructs and executes a computation process by orchestrating relevant simulation functions to obtain quantitative outputs to solve the problem. When the computed outputs fail to satisfy user-specified target ranges, the system initiates parameter optimisation by determining adjustable variables in the orchestrated computation process and searching for parameter adjustments that improve alignment with the user-specified targets. The system is evaluated in the context of pharmaceutical manufacturing through two sets of experiments: benchmark testing of engineering task-solving performance and additional case-study testing on process parameter optimisation. Results demonstrate that the system achieves a 75.0%–91.5% correctness rate in quantitative computation; the optimisation results further show that the system can assist users in identifying effective parameter adjustments across different engineering domains and simulation types. Overall, the proposed system demonstrates an industrial AI application for quantitative engineering decision support.

Keywords: Large language models, agent systems, process engineering, simulation models, tool orchestration, simulation-based optimisation, engineering decision support

1. Introduction

Recent progress in large language models (LLMs) has created new opportunities for accessing and using engineering knowledge through natural-language interaction [1, 2, 3, 4]. However, a fundamental limitation remains: not all domain knowledge is captured in language. The scaling laws of LLM training on text corpora [5, 6] are approaching diminishing returns [7, 8], as the availability of high-quality human-generated text is inherently limited. When required knowledge is missing or insufficiently learned, LLMs may produce vague heuristic responses, fail to converge consistently across repeated attempts [9], or abstain from answering [10, 11]. Such behaviour limits their reliability in industrial engineering applications.

This limitation is particularly critical in engineering tasks, which require precise, quantitative, and reproducible results rather than reasonable text generation. In engineering domains, a substantial portion of knowledge is embedded in domain-specific simulation models, which constitute important organisational knowledge assets in a company [12]. Typically, simulation models encode the mechanistic dynamics of physical and chemical processes and provide quantitative predictions that cannot be sufficiently captured by text corpora alone.

Integrating simulation models with language models provides a principled approach to grounding LLMs in domain-specific knowledge. However, three challenges remain. First, the value proposition of industrial LLM applications in engineering remains insufficiently explored and systematically characterized [13], as such systems must satisfy requirements that differ fundamentally from those of general-purpose chatbots [2, 3]. Second, the methodological integration of exter-

*Corresponding authors.
Email addresses: yuchen.xia@ias.uni-stuttgart.de (Yuchen Xia), michael.weyrich@ias.uni-stuttgart.de (Michael Weyrich), nasser.jazdi@ias.uni-stuttgart.de (Nasser Jazdi), johannes.stuempfle@ias.uni-stuttgart.de (Johannes Stümpfle), johannes.sigel@ias.uni-stuttgart.de (Johannes Sigel), akshay.narla@ias.uni-stuttgart.de (Akshay Narla), Gavin.Reynolds@astrazeneca.com (Gavin K. Reynolds), Anna.Jawor-Baczynska@astrazeneca.com (Anna Jawor-Baczynska), Matthew.Molloy@astrazeneca.com (Matthew Molloy), pol.llopart@astrazeneca.com (Pol Llopart)

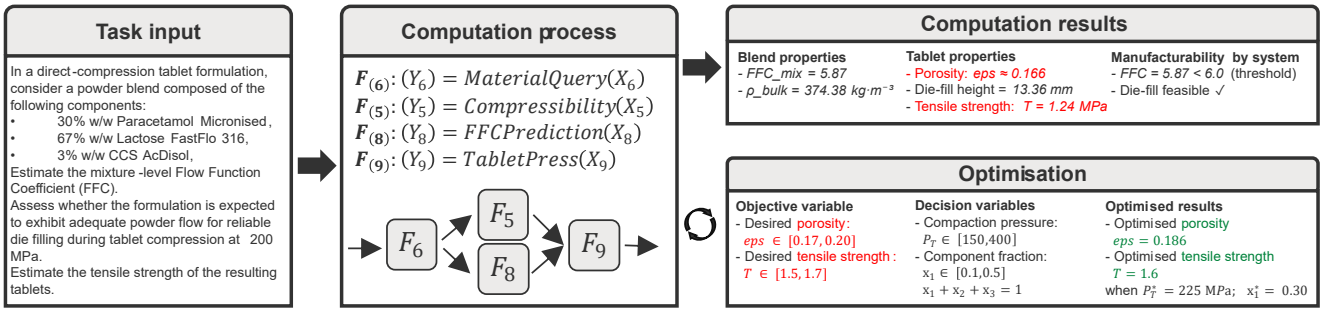


Figure 1: Illustration of the use case and representative outputs: a user-defined task is transformed into simulation-based computation and subsequent optimisation.

nal engineering knowledge into LLM reasoning remains a non-trivial challenge [3, 14], because simulation models and language models differ fundamentally in their representations, interfaces, and reasoning mechanisms. Directly exposing simulation functions through existing tool-use mechanisms can lead to context overload and incorrect function invocations [15], particularly when tasks are complex. Third, industrial deployment requires verifiable evaluation of performance and practical reliability; quantitative benchmark tests are therefore needed to demonstrate whether such systems are usable in realistic engineering use cases.

To address these challenges, this work makes the following contributions:

- We define the industrial application use case of employing LLM agents and simulation models for engineering problem solving and process optimisation, abstract the underlying problem, and formalise its key constructs.
- We design a simulation-integrated LLM-agent framework that realises this application by orchestrating simulation functions into executable computation processes and performing parameter optimisation when required.
- We design a benchmark to evaluate the system performance in computation-process orchestration and demonstrate the optimisation capability of the implemented system.

The proposed methodology addresses the identified gaps in the state of the art, as summarised in Section 7, and is contextualised within pharmaceutical manufacturing. Fig. 1 presents a representative use case, illustrating how a user-defined task is processed through simulation-based computation and parameter optimisation.

2. Application Scenario

Within a pharmaceutical manufacturing company, a large body of process knowledge is embodied in simulation models developed over time. These models include mechanistic models that capture physical and chemical dynamics, as well as semi-empirical and statistical models calibrated from experimental data. These models are implemented as executable modules in domain-specific codebases, with each module capturing a particular process mechanism, estimating a material property, or predicting a quality-related outcome.

Despite their value, these models are difficult for individual engineers to comprehend and utilise effectively. Understanding which model applies to a specific scenario, how to provide the required inputs, how to interpret outputs, and how to compose multiple models to answer a specific engineering question imposes significant cognitive load. As a result, the knowledge embedded in these simulation models remains underutilised, creating a gap between available process knowledge and on-demand decision support.

The following use case is considered: a user poses a process-related query. A software system with access to simulation models translates the query into a structured computational task and constructs an executable computation process to produce quantitative answers. The system may further receive user-desired target ranges for selected computed variables, triggering an optimisation procedure when the variables fail to satisfy these desired ranges.

3. Theoretical Abstraction of the Problem

This industrial application scenario can be formalised as a computation-and-optimisation problem, as illustrated in Fig. 2.

A task consists of (i) input variables whose values are known or provided and (ii) target variables to be computed. The system is equipped with a library of callable simulation functions, each with defined input and output properties. Using LLM-based reasoning and software-based logic, the system selects and orchestrates relevant functions into a computation process that produces the required target variables.

In addition, the system can receive quality specifications that define acceptable ranges for variables. When a computed target variable lies outside its acceptable range, the system initiates an optimisation process by identifying adjustable decision variables, inferring a feasible decision space, and searching for a parameter configuration that improves the objective value. The optimised parameter suggestion is then returned to the user as a recommended adjustment.

3.1. Variable Semantics Representation

A key requirement for enabling LLMs to correctly interpret quantities and function interfaces is that variables are repre-

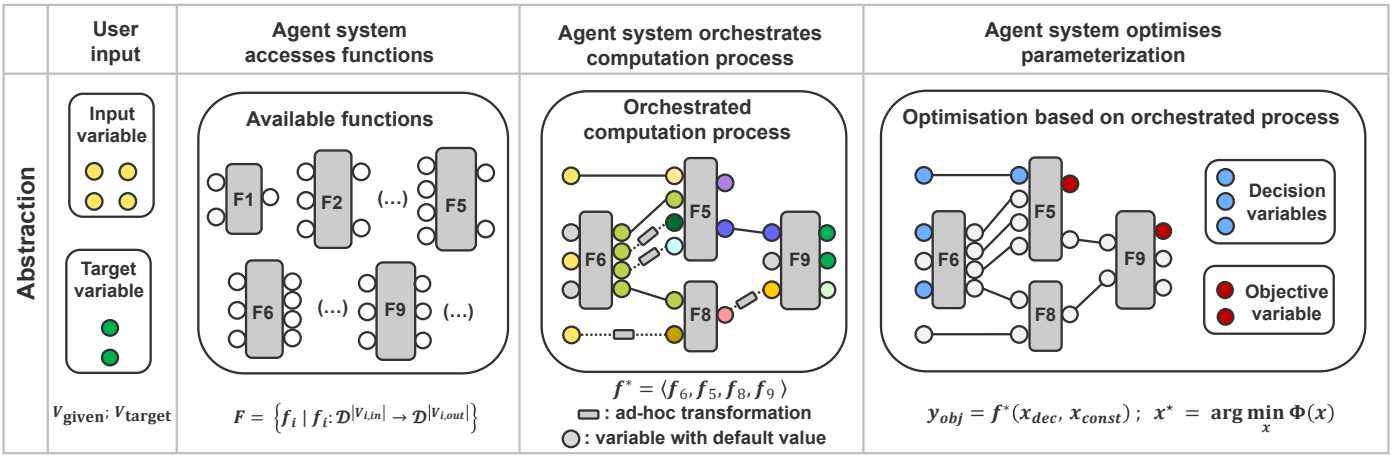


Figure 2: **Abstraction of the Application Problem.** From left to right: (i) input and target variables are inferred from the user query; (ii) relevant simulation functions are retrieved from the repository; (iii) the agent system orchestrates these functions into an executable computation process that produces the target variables; (iv) if any target value violates its acceptable range, it is treated as an objective variable, and the system adjusts decision variables via repeated computation process evaluations to restore in-range values.

sented in a semantically clear and identifiable manner.

We formalise the semantic representation method as follows: every computable variable is defined by an *entity* and a *property*. The entity refers to the physical or conceptual object involved in the task, and the property denotes a characteristic of that entity. A variable is semantically identifiable as a pair (e, p), and its value status is represented by a datum d:

$$d = \mathrm{val}(e, p) \equiv \mathrm{val}(v), \tag{1}$$

Thus, a variable v is representable by the triplet (e, p, d), expressing that "property p of entity e takes value d," or equivalently, "the value of variable v is d." For example:

- the bulk density of a particular powder mixture is $350\,\mathrm{kg/m^3}$,
- the configured compaction pressure of a pressing machine is $200\,\mathrm{MPa}$,
- the warning status of a compaction process is *normal*.

This semantic formalisation enables variables to be structured, stored, and manipulated within a data model, ensuring their semantics remain interpretable to code programs, LLMs and users.

3.2. Standardised Simulation Functions

Simulation functions in the system are formalised in two complementary aspects: their internal computational logic and their external function interface.

- Each function encapsulates reusable computational logic that solves a class of engineering problems and can be applied across different entity instances.
- The interface of each function f is standardised according to the variable representation. Specifically, the input interface $\mathbf{V}_f^{\mathrm{in}}$ and output interface $\mathbf{V}_f^{\mathrm{out}}$ specify the variables consumed and produced by f, respectively. The semantics of each variable $v \in \mathbf{V}$ are identified by its associated entity type and property pair (e_{type}, p).

A standardised simulation function is therefore viewed as a transformation from the input-value space $\mathcal{D}^{|\mathbf{V}_f^{\mathrm{in}}|}$ to the output-value space $\mathcal{D}^{|\mathbf{V}_f^{\mathrm{out}}|}$:

$$f : \mathcal{D}^{|\mathbf{V}_f^{\mathrm{in}}|} \to \mathcal{D}^{|\mathbf{V}_f^{\mathrm{out}}|}. \tag{2}$$

The system maintains a repository of standardised simulation functions in the form of a set, as illustrated in Fig. 2:

$$\mathbf{F} = \left\{ f_i \;\middle|\; f_i : \mathcal{D}^{|\mathbf{V}_{f_i}^{\mathrm{in}}|} \to \mathcal{D}^{|\mathbf{V}_{f_i}^{\mathrm{out}}|} \right\}. \tag{3}$$

3.3. Computation Process Orchestration

Given a task $t = (\mathbf{V}_{\text{input}}, \mathbf{V}_{\text{target}})$, the problem is to select and orchestrate simulation functions into a computation process that produces the required target variables. This requires identifying relevant functions and resolving their input–output dependencies. Whenever an input required by a function is not directly available, the missing value must be resolved. This may require ad-hoc variable transformations between connected functions, as illustrated in Fig. 2.

The selected functions and variables form a directed dependency graph whose edges represent the flow of variables across functions. The graph can be topologically converted into a partially ordered sequence $f^* = \langle f_1, f_2, \ldots, f_m \rangle$. Executing such a sequence yields the values of the target variables.

3.4. Optimisation of Objective Variables over the Orchestrated Process

If a computed variable violates its desired range after execution of the orchestrated computation process, an optimisation problem can be formulated. Formally, a range violation occurs when

$$\exists v \in \mathbf{V}_{\text{prescribed}} \quad \text{s.t.} \quad d_v \notin [L_v, U_v],$$

in which case v becomes an objective variable to be optimised, i.e., $v \in \mathbf{V}_{\text{obj}} \subseteq \mathbf{V}_{\text{prescribed}}$.

To correct such violations, a set of adjustable decision variables $\mathbf{V}_{\text{dec}}$ needs to be defined. Each decision variable $v \in \mathbf{V}_{\text{dec}}$ is assigned an admissible adjustment interval $[L_v^{\text{dec}}, U_v^{\text{dec}}]$, and the bounded decision space is defined as the Cartesian product of these intervals:

$$\mathcal{X}_{dec} = \prod_{v \in \mathbf{V}_{\text{dec}}} [L_v^{\text{dec}}, U_v^{\text{dec}}]. \tag{4}$$

For any candidate decision vector $\mathbf{x}_{dec} \in \mathcal{X}_{dec}$, the orchestrated computation process f^* is re-evaluated to obtain the updated values of the objective variables. The deviation of each objective variable from its acceptable range is quantified by a loss function $\ell_v(\mathbf{x}_{dec})$, and the overall optimisation objective aggregates the individual losses:

$$\Phi(\mathbf{x}_{dec}) = \sum_{v \in \mathbf{V}_{\text{obj}}} w_v \, \ell_v(\mathbf{x}_{dec}), \tag{5}$$

where w_v denotes importance weights determined by the engineering context. The resulting optimisation problem is therefore defined as:

$$\mathbf{x}^\star = \arg \min_{\mathbf{x}_{dec} \in \mathcal{X}_{dec}} \Phi(\mathbf{x}_{dec}), \tag{6}$$

If an acceptable configuration exists, i.e., all objective variables lie within their prescribed acceptable ranges, the corresponding decision-variable values constitute recommended process parameter adjustments; otherwise, the configuration with the lowest objective loss found within the evaluation budget is returned as the optimisation result.

4. Solution Methodology

This section presents the methodology for operationalising the formalised problem into a software solution. The proposed system follows a three-phase procedure, as illustrated in Fig. 3.

1. **Task Structuring**: transforming a user input into a structured task representation.
2. **Simulation Computation Process Orchestration**: constructing and executing a simulation-function computation process that yields the required target variables.
3. **Parameter Optimisation**: adjusting decision variables to optimise objective variables into desired ranges.

4.1. Task Structuring

An LLM agent analyses the free-text input and transforms it into a structured textual model consisting of the following fields:

- **Articulated input:** a reformulated and clarified version of the user input.
- **Task objective:** a specification of the overall purpose of the system operation, together with the target variables to be computed.

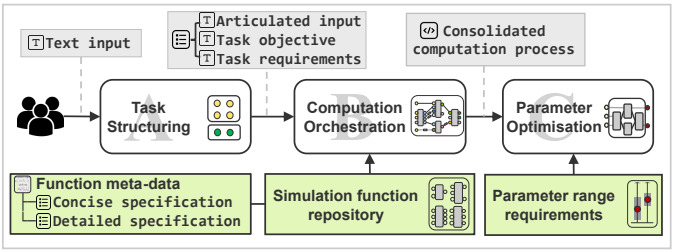


Figure 3: Methodology overview.

- **Task requirements:** a set of inferred requirements that guide the fulfilment of the task objective.

4.2. Simulation Computation Process Orchestration

Based on the structured task, the system constructs a computation process that produces the target variables by orchestrating simulation functions. The system operates as follows:

4.2.1. Function Specification

All simulation functions are maintained in a function repository, each with textual specifications accessible to LLM agents to read. To balance reasoning precision and prompt context efficiency, each function specification is provided in two levels of verbosity:

- **Concise specification**: a short description summarising the purpose and interface of the function.
- **Detailed specification**: a comprehensive description including extra variable semantics, modelling mechanisms, and usage examples.

As shown in Fig. 4, concise specifications enable efficient filtering of relevant functions from a large repository, while detailed specifications support more precise reasoning during orchestration.

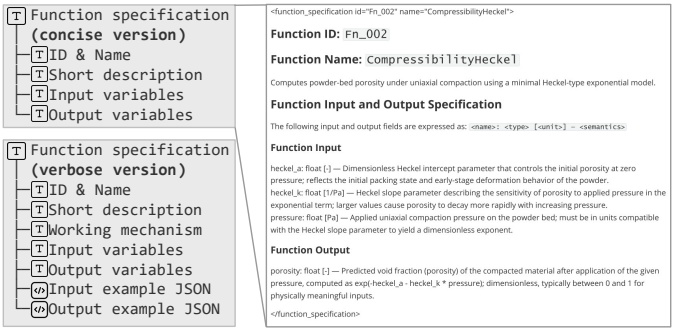


Figure 4: Structure of the two-level function specifications (left), along with an example of the concise version of the function specification (right).

4.2.2. Retrieval of Task-relevant Functions

Given the structured task model and the concise function specifications, a retrieval agent identifies task-relevant simulation functions, narrowing the function library to a candidate subset for subsequent computation process planning, as shown in Fig. 5.

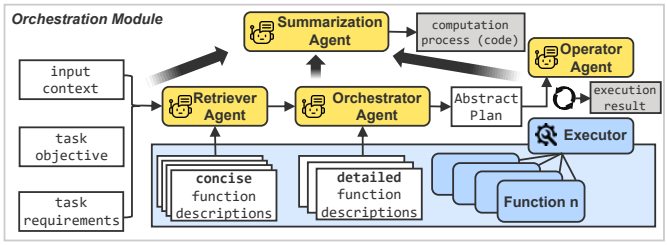


Figure 5: System module for function orchestration in solving user tasks.

4.2.3. Abstract Plan Construction

Using the candidate functions and their detailed specifications, a planning agent constructs an abstract computation plan represented as an ordered sequence of simulation functions $f^* = \langle f_1, f_2, \ldots, f_m \rangle$ that collectively produce the required target variables.

At this stage, the plan specifies only the function sequence and logical ordering, without concrete input values or execution-level details. It therefore serves as a blueprint for the subsequent execution phase.

Since the candidate function set is much smaller than the original library, detailed specifications can be incorporated into the LLM prompt context to support more precise reasoning about function dependencies and variable associations.

4.2.4. Interactive Plan Execution

The abstract plan is executed through an interaction between an **operator agent** powered by an LLM and a deterministic **executor** that executes simulation-function code.

The operator agent incrementally concretises the ordered abstract plan. For each step in the plan, it generates a function call specifying the input values required to invoke the corresponding simulation function. These inputs follow the entity–property representation introduced in Section 3.1 and are expressed in JSON format.

The executor receives the function-call payload, invokes the corresponding simulation function, and returns the resulting outputs. The operator then continues the reasoning process using the execution results as additional context. Through this iterative interaction, the operator and executor sequentially execute the planned functions. Intermediate variable values are passed from one function to the next, thereby linking the input and output variables across the computation process.

This mechanism, in effect, operationalises the computation process formalised in the problem definition in Section 2, as illustrated in Fig. 2.

4.2.5. Summarisation and Computation Process Consolidation

The reasoning and execution process generates textual records of simulation function calls, intermediate reasoning traces, and final results. A summarisation agent consolidates these into two outputs:

- a user-facing summary that explains how the task was solved and highlights the key results.
- an executable Python script that captures the logical computation process derived from the reasoning and execution processes.

The Python script provides a deterministic and reproducible implementation of the computation process and also serves as the evaluation function for the optimisation procedure described in the following section.

4.3. Parameter Optimisation

In many task scenarios, the variables computed by simulation functions must lie within predefined acceptable ranges reflecting the engineering requirements. When a computed variable falls outside its acceptable range, optimisation is required to adjust controllable parameters, i.e., decision variables of the computation process, to improve the objective outcomes.

The system performs parameter optimisation over the computation process in three steps: formulating an optimisation task, constructing the decision space of controllable parameters, and performing numerical optimisation within this space, as illustrated in Fig. 6.

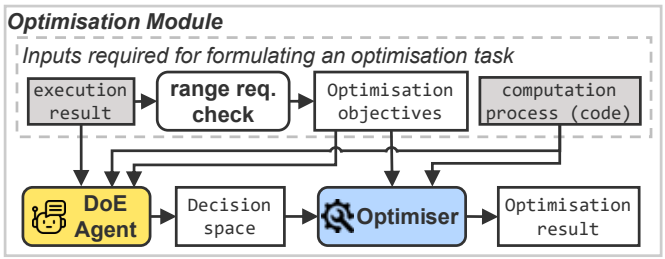


Figure 6: System module for the parameter optimisation procedure.

This module is implemented through the following mathematical formulation:

4.3.1. Forming the Optimisation Task

Let f^* denote the consolidated computation process generated as an executable code script during the previous orchestration. The objective variables produced by this function can be represented as a vector $\mathbf{y}_{obj} = (v_{obj,1}, v_{obj,2}, \ldots, v_{obj,k})$:

$$\mathbf{y}_{obj} = f^*(\mathbf{x}_{dec}, \mathbf{x}_{const}) \tag{7}$$

where $\mathbf{x}_{dec}$ denotes the vector of adjustable decision variables and $\mathbf{x}_{const}$ the vector of fixed input variables.

To measure how far each objective variable v_{obj} deviates from its acceptable range $[L_{v_{obj}}, U_{v_{obj}}]$, we define a normalised distance from the centre of the range interval:

$$dist_{v_{obj}}(\mathbf{x}_{\text{dec}}) = \frac{\left| f^*_{v_{obj}}(\mathbf{x}_{\text{dec}}, \mathbf{x}_{\text{const}}) - \frac{L_{v_{obj}} + U_{v_{obj}}}{2} \right|}{\frac{U_{v_{obj}} - L_{v_{obj}}}{2}}. \tag{8}$$

Based on this deviation distance, an objective loss function is defined that treats in-interval deviations mildly (when $dist_{v_{obj}}(\mathbf{x}_{\text{dec}}) \leq 1$) and applies a linearly increasing penalty to out-of-interval violations for an objective variable v_{obj}:

$$\ell_{v_{obj}}(\mathbf{x}_{dec}) = \begin{cases} dist_{v_{obj}}(\mathbf{x}_{dec})^2, & dist_{v_{obj}}(\mathbf{x}_{dec}) \leq 1, \\ 2\, dist_{v_{obj}}(\mathbf{x}_{dec}) - 1, & dist_{v_{obj}}(\mathbf{x}_{dec}) > 1, \end{cases} \tag{9}$$

The overall optimisation loss aggregates losses across a set of objective variables $\mathbf{V}_{\text{obj}}$ with equal weights by default (i.e., $w_{v_{obj}} = 1$ in Eq. 5):

$$\Phi(\mathbf{x}_{dec}) = \sum_{v_{obj} \in \mathbf{V}_{\text{obj}}} \ell_{v_{obj}}(\mathbf{x}_{dec}). \tag{10}$$

Accordingly, the optimisation problem in Eq. 6 becomes

$$\mathbf{x}^{\star}_{dec} = \arg \min_{\mathbf{x}_{dec} \in \mathcal{X}_{dec}} \sum_{v_{obj} \in \mathbf{V}_{\text{obj}}} \ell_{v_{obj}}(\mathbf{x}_{dec}). \tag{11}$$

In addition to minimising $\Phi(\mathbf{x}_{dec})$, an acceptance indicator is computed as

$$\text{isAccepted}(\mathbf{x}_{dec}) = \bigwedge_{v \in \mathbf{V}_{\text{obj}}} \left(f^*_v(\mathbf{x}_{\text{dec}}, \mathbf{x}_{\text{const}}) \in [L_v, U_v] \right) \tag{12}$$

which indicates whether all acceptable intervals are satisfied simultaneously.

4.3.2. DoE Agent Reasoning on Decision Space

Within the optimisation procedure defined above, a key challenge in engineering use cases is the complex relationship between adjustable variables and objective variables. Determining which variables should be adjusted, and over what feasible ranges, is therefore context-specific and requires significant cognitive reasoning. Consequently, selecting a set of decision variables $\mathbf{V}_{\text{dec}} \subset \mathbf{V}$ and specifying a decision space $\mathbf{x}_{\text{dec}} \in \mathcal{X}_{\text{dec}}$ (in Eq. 7) becomes a nontrivial task.

To automate this step, the system employs a Design-of-Experiment (DoE) agent powered by an LLM. Using the variable semantics and prior reasoning results from the orchestration stage, the DoE agent reasons about how adjustments to computation process inputs may affect the objective variables. It then proposes a set of decision variables $\mathbf{V}_{\text{dec}} \subset \mathbf{V}$ together with hypothetical adjustment ranges $[L_{v_{dec}}, U_{v_{dec}}]$. These intervals span a Cartesian decision space $\mathcal{X}_{dec} = \prod_{v \in \mathbf{V}_{\text{dec}}} [\, L_{v_{dec}},\ U_{v_{dec}}\,]$ (cf. Eq. 4), which represents the DoE agent's hypothesis about which regions of the parameter space are likely to yield improved optimisation outcomes.

4.3.3. Numerical Optimisation in the Decision Space

Once the decision space $\mathcal{X}_{dec}$ is generated, the numerical optimiser treats the consolidated computation process f^* as a black-box evaluation function. For any candidate decision vector $\mathbf{x}^{(t)}_{dec} \in \mathcal{X}_{dec}$ at evaluation step t, the resulting objective variables $\mathbf{y}^{(t)}_{obj} = f^*(\mathbf{x}^{(t)}_{dec}, \mathbf{x}_{const})$ and the overall objective loss $\Phi^{(t)}$ are evaluated using Eq. 7 and Eq. 10. The evaluation result $\mathcal{R}$ is recorded as a set of tuples consisting of evaluated decision variables, resulting objective variables, and corresponding loss:

$$\mathcal{R} = \left\{ \left(\mathbf{x}^{(t)}_{dec}, \mathbf{y}^{(t)}_{obj}, \Phi^{(t)} \right) \right\}_{t=1}^{T}. \tag{13}$$

The system identifies the candidate(s) $\mathbf{x}^*_{dec}$ with the minimum loss Φ^* from records $\mathcal{R}$, and checks optimisation success using the acceptance criterion $\text{isAccepted}(\mathbf{x}^*_{\text{dec}})$ in Eq. 12.

5. Testing Results and Benchmark

The proposed methodology is realised as a software prototype. Its effectiveness is evaluated based on the two types of results it produces: **orchestrated computation results** and **optimisation results**[1]. This section presents the benchmark design and the evaluation results on representative engineering tasks.

5.1. System Outputs and Evaluation Setup

The implemented software incorporates 16 domain-specific simulation functions covering typical engineering tasks in pharmaceutical manufacturing. To evaluate computation orchestration, 20 benchmark test cases are constructed to examine whether the system correctly produces the quantitative outputs[2], while case studies are conducted to assess the effectiveness of the optimisation process, as illustrated in Fig. 7.

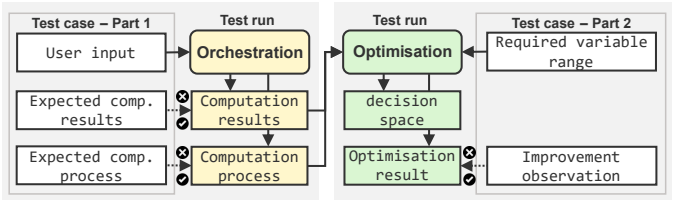


Figure 7: Structure of the two types of test cases. For orchestration evaluation, each test case specifies a user input, and the resulting outputs are compared with the expected outputs. For optimisation evaluation, the optimisation process is examined in terms of range satisfaction and loss reduction.

5.2. Testing Part 1 – Test of Computation Process Orchestration

The 20 benchmark tasks span two specialised engineering domains, and each requires the orchestration of one or more simulation functions to compute quantitative process outcomes:

- **Powder Blending & Compaction (#1 – #10):** tasks involving the computation of powder-mixture properties and the prediction of pharmaceutical tablet properties.
- **Solubility & Crystallisation (#11 – #20):** tasks involving thermodynamic solubility calculation and dynamic crystallisation simulation.

Table 1 summarises the benchmark tasks together with the structural complexity metrics used to characterise task difficulty.

[1] The operation of the software prototype is demonstrated in *Supplementary Materials: Video Abstract*, which illustrates the computation orchestration functions and the optimisation functions.

[2] The complete benchmark test cases and corresponding evaluation results are provided in *Supplementary Material: Benchmark Test Results*.

Table 1: Overview of Benchmark Test Cases and Their Complexity

#	Task	$\lvert V_{\text{tar}}\rvert$	$\lvert F\rvert$	$\lvert D_{\text{max}}\rvert$	B	I
1	Direct Compression Tablet Formulation	3	5	4	–	–
2	Bulk Density Computation	1	1	1	–	–
3	Tablet Porosity and Strength	2	4	4	–	–
4	Filler Family Performance Comparison	4	4	4	✓	–
5	Particle Size Distribution Flow	2	2	2	–	–
6	Minimum Compression Pressure	2	4	4	–	✓
7	Particle Size Distribution and Flowability	1	1	1	–	–
8	Compression Pressure Sensitivity	3	4	4	✓	–
9	Mixture Component Comparison	4	4	4	✓	–
10	Filler Material Replacement Evaluation	3	6	6	–	–
11	Equilibrium Solubility Prediction	1	2	2	–	–
12	Temperature Effect on Solubility	1	2	2	–	–
13	Seed Loading Impact on Crystallisation	6	3	3	✓	–
14	Supersaturation Window Determination	2	2	2	✓	–
15	Crystallisation Temperature Requirement	2	2	2	–	✓
16	Crystal Flowability Evaluation	1	4	4	–	–
17	Crystallisation Yield Estimation	1	2	2	–	–
18	Batch Crystallisation Result Prediction	5	3	3	–	–
19	Dissolution Temperature Determination	1	2	2	–	✓
20	Crystallisation Cooling Rate Comparison	3	3	3	✓	–

$\lvert V_{\text{tar}}\rvert$	number of target variables to be computed
$\lvert F\rvert$	number of required simulation functions to solve the task
$\lvert D_{\text{max}}\rvert$	maximum number of chained dependent functions
B	presence of branching within the reasoning process
I	presence of iteration within the reasoning process

Task complexity is characterised along three dimensions: output and function complexity ($|V_{\text{tar}}|, |F|$), dependency complexity ($|D_{\text{max}}|$), and control-flow complexity (B, I). Larger $|V_{\text{tar}}|$ and $|F|$ generally increase the breadth of required outputs and the difficulty of function selection and composition, while larger $|D_{\text{max}}|$ increases the risk of error propagation along longer dependent function chains. B and I capture additional reasoning complexity arising from branching and repeated function invocation, respectively. These metrics are illustrated schematically in Fig. 8.

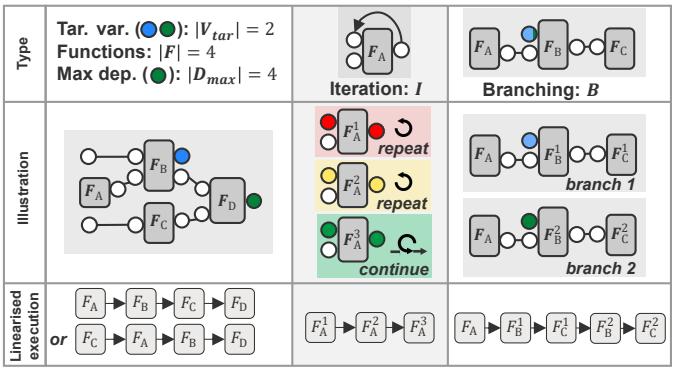


Figure 8: Illustration of structural complexity metrics in benchmark tasks. Top: examples of computation processes illustrating dependency depth ($|D_{\text{max}}|$), branching (B), and iteration (I). Bottom: corresponding linearised execution sequences under operator–executor interaction.

Additional factors of task complexity. Beyond the structural metrics above, task difficulty is also affected by implementation-level factors such as unit conversion, semantic translation and disambiguation, default-value assignment and missing-input inference. Although not treated as primary benchmark complexity dimensions, they remain practical sources of possible failure in end-to-end task solving.

5.3. Testing Part 1 – Benchmark Result

The benchmark evaluation was conducted using 6 different LLMs as the reasoning engine to power the designed agent system. For each model, all 20 benchmark tasks were executed in 5 repeated runs, resulting in 100 executable runs per model and 600 runs in total. Each run was evaluated using three criteria:

- **Executability:** whether the reasoning and execution process completes without runtime or function-execution errors.
- **Computation-process completeness:** whether the system constructs a complete computation process and all required target variables are produced.
- **Result correctness:** whether the produced target values match the expected values (reference answers). A numerical value is considered correct if its relative error is below 1%, allowing for minor rounding differences. For tasks with multiple target variables, correctness is measured as the fraction of the correctly computed variables:

$$P(\text{Correct} \mid \text{Executable}) = \frac{|V_{\text{tar,matched}}|}{|V_{\text{tar}}|}. \tag{14}$$

Thus, correctness is evaluated conditional on successful execution and quantified as the fraction of target variables matching the reference values.

The aggregated benchmark results are summarised in Table 2.

Table 2: Overall Benchmark Performance

Model	Errors(**Executable**)	**Complete**	**Correct**
Deepseek-V3.2-685B-FP4	2 in 102 (**98**%)	**93**%	**81.1**%
GLM-5-435B-FP4	5 in 105 (**95**%)	**95**%	**91.5**%
MiniMax-M2.5-230B-FP4	1 in 101 (**99**%)	**97**%	**89.1**%
Kimi-K2.5-1T-A32B-INT4	1 in 101 (**99**%)	**97**%	**85.0**%
Qwen3.5-397B-A17B	0 in 100 (**100**%)	**97**%	**91.3**%
gpt-oss-120B-FP4	4 in 104 (**96**%)	**98**%	**75.0**%

Overall, all six models achieve relatively high executability across diverse engineering tasks. Successful task solving follows a funnel-like sequence: a run must be executable, produce a complete computation process with all required outputs, and finally yield correct values.

5.4. Testing Part 1 – Benchmark Result Analysis

This subsection analyses benchmark performance in more detail and further examines the main failure modes underlying the errors.

Executability and completeness errors are explicit. Executability failures are mainly caused by nonconforming JSON payloads that lead to parsing or function-execution errors. By contrast, completeness failures occur when the system completes but is not able to return all outputs requested in the user input, for example due to ambiguous concept interpretation, an unproductive reasoning path, or accumulated inconsistencies. Both failure types are clearly detectable and are therefore often recoverable by rerunning the agent system.

Correctness errors are more difficult to detect, because the system may produce plausible but incorrect outputs. For this reason, the more critical metric for evaluating task-solving quality is the conditional correctness $P(\text{Correct} \mid \text{Executable})$, which is determined in the benchmark by comparison with externally provided reference values specified in the test cases.

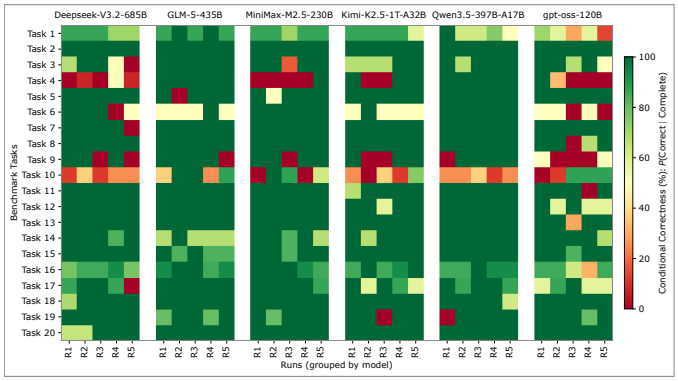


Figure 9: Correctness rate across 20 benchmark tasks and 5 repeated runs for 6 evaluated open-source LLMs. Each cell reports $P(\text{Correct} \mid \text{Executable})$ for a specific task–run

Fig. 9 visualises overall system performance in producing correct quantitative results across benchmark tasks[3]. The results indicate that tasks of varying complexity (cf. Table 1) are generally well handled by the system. The observed failures are attributable to multiple factors and are mainly captured by the failure modes summarised in Table 3.

Implications for Practical Application. Overall, the benchmark results indicate satisfactory system-level performance for computation-process orchestration across diverse engineering tasks. Failures are relatively infrequent, and the consistently high conditional correctness across open-source models suggests that simulation-integrated orchestration can produce accurate, numerically grounded outputs across heterogeneous engineering tasks. Across all evaluated models, conditional correctness exceeds 75%, supporting the suitability of the system as an engineering assistant.

[3]The complete benchmark test cases and corresponding evaluation results are provided in the *Supplementary Material: Benchmark Test Results*.

Table 3: Failure modes in computation process orchestration

Failure mode	**Description**
Incorrect default values	Missing inputs force the system to infer default values that may not reflect the intended engineering conditions.
Function call failure	Malformed payloads generated by the agent may prevent simulation functions from executing.
Ambiguous task reference answers	Some tasks have multiple reasonable solutions, while the benchmark reference represents only one.
Simulation model inaccuracies	Simplified simulation models omit relevant physical mechanisms, leading to prediction deviations.
Semantic similarity confusion	Semantic similarity between functions may introduce ambiguity in reasoning and lead to errors.
Imperfect function descriptions	Incomplete function specifications may lead to incorrect interpretation of function applicability.

5.5. Testing Part 2 – Test of Parameter Optimisation

This subsection evaluates system performance on optimisation tasks and discusses the practical applicability of the method based on the observed outcomes. The evaluation focuses on two aspects:

- **Optimisation effectiveness:** assessed by loss reduction and satisfaction of prescribed variable ranges during repeated evaluations of the consolidated computation process, visualised with loss curves.
- **Runtime performance:** assessed by the wall-clock time of an optimisation run and its decomposition into LLM inference time and numerical optimisation time.

To quantify runtime performance, the wall-clock time T_{Run} of an optimisation task run is decomposed into the LLM inference time required by the DoE agent to formulate the decision space $T_{\text{LLM,DoE}}$, and the numerical optimisation time $T_{\text{Optimiser}}$:

$$T_{\text{Run}} = T_{\text{LLM,DoE}} + \underline{T_{\text{Optimiser}}} \tag{15}$$

For generality, the black-box optimiser used in this study is based on random grid sampling. Its runtime is dominated by repeated simulation evaluations $N_{\text{Eval,Sim}}$, while the optimiser overhead is represented by a negligible term ε:

$$\underline{T_{\text{Optimiser}}} = N_{\text{Eval,Sim}} \cdot T_{\text{Eval,Sim}} + \varepsilon. \tag{16}$$

We then consider two representative scenarios that differ in simulation runtime cost and domain.

5.5.1. Fast simulation scenario: powder mixing and drug formulation process

In this scenario, the computation process is obtained from orchestration task #1 in Table 1, which predicts the outcome of a tableting process after mixing different powder ingredients. In this scenario, the simulations are represented by analytical equations in polynomial form with fitted parameters, allowing

each candidate to be evaluated within milliseconds. The optimiser can therefore explore a large number (10^5) of candidates within a practical runtime budget.

The optimisation objective is to identify material composition configurations such that the resulting tablet satisfies the following customised, product-specific quality requirements:

- tensile strength $T_{\mathrm{MPa}} \in [1.5, 1.7]$;
- porosity $\varepsilon \in [0.17, 0.20]$.

Figure 10 shows the optimisation processes for 10 independent runs. Across runs, the DoE agent constructs different decision spaces as a result of variability in LLM reasoning, while the subsequent numerical optimiser explores these spaces to reduce the objective loss and identify improved parameter settings.

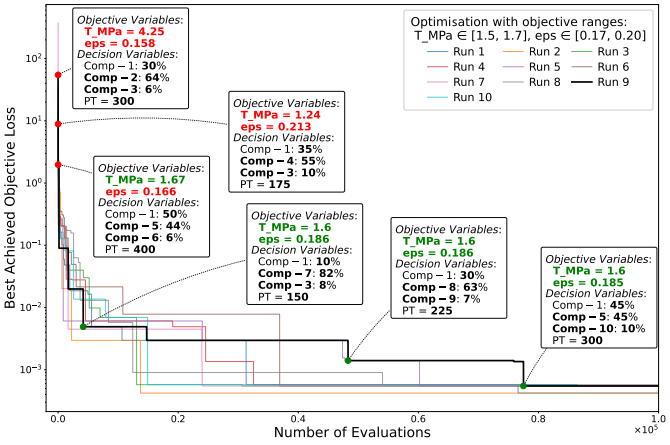


Figure 10: **Optimisation performance across runs in the fast-simulation formulation scenario.** The figure plots the best achieved objective loss against the number of evaluations for 10 optimisation runs. The two objective variables are required to satisfy $T_{\mathrm{MPa}} \in [1.5, 1.7]$ and $\varepsilon \in [0.17, 0.20]$. In each colour-coded run, the DoE agent may generate a different decision space by selecting adjustable variables and their admissible ranges through its reasoning process. The optimiser then searches within this decision space and records the best achieved candidate. Annotated callouts show representative best achieved candidates discovered during the optimisation process of Run 9, including the resulting objective-variable values and the associated decision-variable configurations.

Across the ten runs, the DoE agent required on average $T_{\mathrm{LLM,DoE}} = 32.0\,\mathrm{s}$ ($\sigma_{std} = 4.3\,\mathrm{s}$, range : 27.0–41.5 s) to formulate the decision space, while the optimiser required about $T_{\mathrm{Optimiser}} = 24.9\,\mathrm{s}$ ($\sigma_{std} = 2.7\,\mathrm{s}$, range : 20.5–29.2 s) to perform the evaluation. The overall optimisation process therefore completes in $T_{\mathrm{Run}} = 56.9\,\mathrm{s}$ on average, indicating practical applicability as a software tool.

5.5.2. *Slow simulation scenario: kinetic crystallisation model*

The second scenario considers a batch crystallisation process simulated by numerical integration of coupled differential equations describing nucleation and crystal growth. The computation process is produced from orchestration task #18 in Table 1. In contrast to the first scenario, each evaluation in this scenario requires executing a computationally expensive simulation and takes approximately 4 s. Under this condition, only a limited number of evaluations can be performed.

Figure 11 shows the optimisation processes for 10 independent runs. The optimisation objective is to adjust the parameterised cooling profile so as to:

- maximise the yield, subject to a required yield $\geq 94\%$;
- regulate moderate supersaturation during the crystallisation process, quantified by the time-averaged squared deviation of supersaturation $S(t)$ from the reference value 1.2, calculated with $\frac{1}{T}\int_0^T (S(t) - 1.2)^2\,dt$. This supersaturation-deviation metric is required to be ≤ 0.016.

Compared with the fast-simulation scenario, the slow-simulation scenario permits a much smaller repeated evaluation budget (50) within limited time, leading to more pronounced variation across runs, as visualised in Figure 11. The variation indicates that, when only a limited number of simulation evaluations can be performed, optimisation performance depends strongly on whether the decision variables and their admissible ranges have been chosen appropriately.

In particular, Run 10 reaches favourable low-loss configurations earlier than the other runs, as the decision space proposed in that run was better positioned around the optimal solution region and allowed for efficient search. By contrast, the other runs either improve more slowly or fail to reach similarly favourable solutions within the same evaluation budget. These results highlight the importance of LLM reasoning quality in decision-space formulation when simulation evaluations are computationally expensive.

Across the 10 crystallisation runs, the DoE agent required on average $T_{\mathrm{LLM,DoE}} = 32.5\,\mathrm{s}$ ($\sigma_{\mathrm{std}} = 3.3\,\mathrm{s}$, range : 27.9–38.4 s) to formulate the decision space, comparable to the fast-simulation scenario. By contrast, the numerical search required $T_{\mathrm{Optimiser}} = 211.8\,\mathrm{s}$ ($\sigma_{\mathrm{std}} = 23.1\,\mathrm{s}$, range : 183.2–249.6 s), leading to an average total runtime of $T_{\mathrm{Run}} = 244.3\,\mathrm{s}$ per run, which remains practical for interactive software use.

5.5.3. *Discussion on the convergence behaviour across the two scenarios*

Together, the two scenarios show that the evaluation cost of the orchestrated computation process governs the sensitivity of optimisation performance to decision-space quality. In the fast-simulation scenario, the large evaluation budget allows the optimiser to partly compensate for decision-space variability. In the slow-simulation scenario, the limited evaluation budget makes convergence more dependent on whether the DoE agent proposes suitable decision variables and admissible ranges. This effect is reflected in the slower convergence of the loss curves in Fig. 10 and Fig. 11. Under limited evaluation budgets, how simulation evaluations are allocated becomes a decisive factor in optimisation performance [16, 17]. In the present framework, this allocation is implicitly shaped by the agent's decision-space formulation, which determines the region explored by the numerical optimiser.

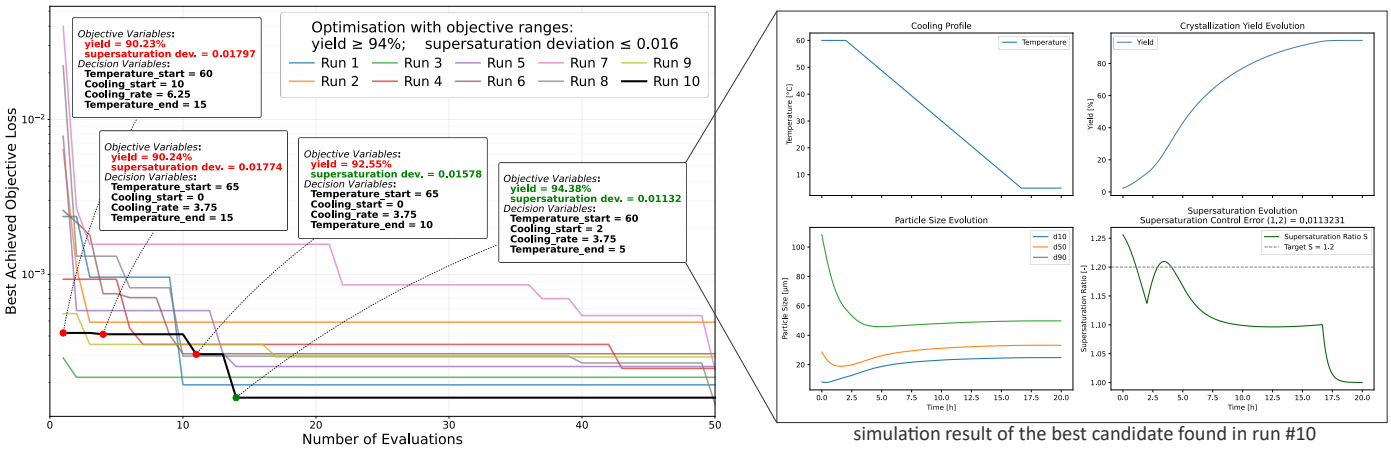


Figure 11: **Optimisation performance across runs in the slow-simulation crystallisation scenario.** Left: convergence behaviour of ten optimisation runs, where the best achieved objective loss is plotted against the number of simulation evaluations. Each run corresponds to a different decision space proposed by the DoE agent. Annotated callouts highlight representative candidate solutions discovered during the optimisation process of the best-performing run, including the resulting objective-variable values and the associated decision-variable settings. Right: simulated process trajectory corresponding to the best-performing candidate configuration. The optimised cooling policy (top-left) produces a steadily increasing crystallisation yield (top-right), smooth particle-size evolution characterised by d_{10}, d_{50}, and d_{90} (bottom-left), and a supersaturation trajectory that has the minimum deviation from the target level $S_{\text{target}} = 1.2$ during the whole batch process time (bottom-right).

5.5.4. Implications for Practical Application

The results further prove that the framework remains applicable across scenarios with substantially different simulation costs and domains. In fast-simulation settings, larger evaluation budgets make optimisation more tolerant of the quality variation in the LLM-formulated decision space. In slow-simulation settings, however, performance becomes much more sensitive to decision-space quality, placing greater demands on the LLM agent's reasoning results.

The two case studies demonstrate that the proposed framework can automate simulation-driven optimisation in practical engineering workflows. In particular, the agent system can automatically identify adjustable variables and formulate an optimisation experiment, thereby reducing the manual effort typically required to set up optimisation tasks.

6. Threats to Validity

This section discusses critical aspects of internal validity, i.e., whether the observed performance can be attributed to the proposed system itself, and of external validity, i.e., whether the findings generalise beyond the evaluated setting.

6.1. Internal validity.

The reported task-solving and optimisation results depend on the fidelity of the simulation functions used in the prototype. The 16 incorporated functions are representative of functions used in the target engineering domain, but they remain simplified computational representations rather than complete physical models. Consequently, system outputs should be interpreted in light of the assumptions, abstractions, and numerical precision of the adopted simulation models. Future extensions could incorporate uncertainty quantification, such as confidence intervals, to better characterise output reliability.

The system may also overlook unmodelled effects. For example, optimisation targeting one objective (e.g., tablet tensile strength) may adversely affect other unmodelled properties (e.g., dissolution behaviour), limiting the interpretation of the results as system-level optimality.

6.2. External validity.

The system's task-solving capability is constrained by the coverage and types of simulation functions available to the agent system. Therefore, the reported performance may not directly generalise to tasks requiring substantially different simulation models. However, the main methodological contributions are independent of the specific function set and can be transferred to other domains if suitable simulation models are integrated in a modular manner for agent use.

6.3. Reproducibility.

The system reproduces comparable performance results across a large number of test runs (600) using different models. The software results presented in this paper are subject to industrial intellectual property protections; therefore, the simulation functions and certain implementation details used in the prototype cannot be publicly released. Nevertheless, the academic contribution of this work remains intact: the problem abstraction, methodology, system design, and experiment results are provided to support independent reimplementation.

7. Related Work

Related work relevant to this paper spans three directions: LLM applications in scientific and engineering domains, LLM multi-agent systems with tool and simulation integration, and LLM-assisted optimisation.

7.1. LLM applications in scientific and engineering domains

Recent studies have demonstrated the potential of LLMs in scientific and engineering domains, often by incorporating domain knowledge through retrieval-augmented generation (RAG) [18, 19]. In chemistry and materials science, LLMs have been applied to predictive tasks, retrosynthesis, drug discovery, and modular task-execution frameworks [20, 21, 22, 23, 24]. In engineering, they have been used for CAD generation, process model construction, code generation, and industrial decision support [25, 26, 27, 28, 29, 30, 31].

However, these approaches mainly leverage LLMs for textual knowledge interpretation and language-level task execution, while leaving the mechanistic knowledge embedded in simulation models underutilised. They generally do not construct quantitative computation processes with well-defined variable associations and composable function units, and therefore provide limited support for verifiable quantitative problem solving and parameter optimisation.

7.2. LLM multi-agent systems with tool and simulation integration

Multi-agent architectures have been proposed to extend LLM capabilities beyond standalone text reasoning. Representative systems such as Coscientist and ChemCrow demonstrate complex experimental planning and tool use [32, 33], while related frameworks support chemical synthesis development, CFD configuration, autonomous simulation workflows, robotics integration, and simulation-model generation [34, 35, 36, 37, 38, 39, 40, 41]. More general agent frameworks further improve cooperative tool use, structured planning, and tool descriptions [42, 43, 44, 45].

Despite these advances, simulations are typically treated as external tools within loosely structured workflows. As a result, existing systems generally do not capture simulation functions as composable computational units with explicit variable-level dependencies, and therefore lack a principled mechanism for constructing reusable computation processes for quantitative engineering problem solving.

7.3. LLMs for solving optimisation problems

Related work on LLM-assisted optimisation shows that LLMs can serve as high-level components in optimisation pipelines, for example as search operators, heuristic generators, or problem formulators [46, 47]. Systems such as OptiMUS further demonstrate that LLMs can translate natural-language descriptions into formal optimisation models and solver code [48].

However, these approaches are primarily developed in generic optimisation settings and are not grounded in concrete engineering application scenarios.

8. Conclusion and Outlook

This paper presented a simulation-integrated LLM-agent framework for engineering problem solving and process parameter optimisation. The framework treats simulation models as composable computational units and transforms natural-language engineering queries into executable, reproducible computation processes, providing a foundation for simulation-grounded LLM systems for quantitative engineering analysis and decision making.

Experimental results demonstrate high quantitative correctness across diverse engineering tasks. Optimisation case studies further show that the system can formulate decision spaces and obtain satisfactory parameter configurations within practical runtime constraints. Future work will focus on extending simulation coverage, improving implementation maturity, and enhancing system robustness for industrial deployment.

CRediT authorship contribution statement

Yuchen Xia: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Resources, Visualization, Writing – original draft, Writing – review & editing.

Michael Weyrich: Conceptualization, Validation, Investigation, Resources, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Nasser Jazdi: Conceptualization, Visualization, Writing – review & editing, Supervision, Project administration.

Johannes Stümpfle: Conceptualization, Software, Writing – review & editing.

Johannes Sigel: Conceptualization, Software, Writing – review & editing.

Akshay Narla: Conceptualization, Software, Visualization, Writing – review & editing.

Gavin K. Reynolds: Conceptualization, Software, Validation, Investigation, Data curation, Resources, Writing – review & editing, Supervision, Project administration, Funding acquisition, Validation.

Anna Jawor-Baczynska: Conceptualization, Software, Data curation, Resources, Validation.

Matthew Molloy: Conceptualization, Resources, Validation.

Pol Llopart: Conceptualization, Software, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] H. Zhang, W. Yan, H. Hu, et al., An llm-based knowledge and function-augmented approach for optimal design of remanufacturing process, Advanced Engineering Informatics 65 (2025) 103206. doi:https://doi.org/10.1016/j.aei.2025.103206.

[2] Y. Wan, Z. Chen, Y. Liu, et al., Empowering llms by hybrid retrieval-augmented generation for domain-centric qa

in smart manufacturing, Advanced Engineering Informatics 65 (2025) 103212. doi:https://doi.org/10.1016/j.aei.2025.103212.

[3] C. Liu, J. Song, D. Tang, et al., Probing a novel machine tool fault reasoning and maintenance service recommendation approach through data-knowledge empowered llms integrated with ar-assisted maintenance guidance, Advanced Engineering Informatics 66 (2025) 103460. doi:https://doi.org/10.1016/j.aei.2025.103460.

[4] H. Cui, P. Zheng, M. Ren, et al., An llm-based cross-domain knowledge retrieval augmented generation method for bio-inspired solution design, Advanced Engineering Informatics 69 (2026) 104017. doi:https://doi.org/10.1016/j.aei.2025.104017.

[5] J. Kaplan, S. McCandlish, T. Henighan, et al., Scaling laws for neural language models, arXiv preprint arXiv:2001.08361 (2020). doi:10.48550/arXiv.2001.08361.

[6] A. Chowdhery, S. Narang, J. Devlin, et al., Palm: scaling language modeling with pathways, J. Mach. Learn. Res. 24 (1) (jan 2023). doi:10.5555/3648699.3648939.

[7] I. Sutskever, D. Patel, We're moving from the age of scaling to the age of research, dwarkesh Podcast interview transcript, URL: https://www.dwarkesh.com/p/ilya-sutskever-2 (2025).

[8] E. Dohmatob, Y. Feng, P. Yang, et al., A tale of tails: model collapse as a change of scaling laws, in: Proceedings of the 41st International Conference on Machine Learning, 2024. doi:10.5555/3692070.3692515.

[9] L. Zhou, W. Schellaert, F. Martínez-Plumed, et al., Larger and more instructable language models become less reliable, Nature 634 (8032) (2024) 61–68. doi:10.1038/s41586-024-07930-y.

[10] B. Wen, J. Yao, S. Feng, et al., Know your limits: A survey of abstention in large language models, Transactions of the Association for Computational Linguistics 13 (2025) 529–556. doi:10.1162/tacl_a_00754.

[11] P. Kirichenko, M. Ibrahim, K. Chaudhuri, et al., Abstentionbench: Reasoning LLMs fail on unanswerable questions, in: The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2025. doi:10.48550/arXiv.2506.09038.

[12] J. Mädler, C. G. Serrano, I. Viedt, et al., Simulation model exchange in process industry: Requirements, solutions, and open challenges, Chemical Engineering & Technology 48 (3) (2025) e202400331. doi:https://doi.org/10.1002/ceat.202400331.

[13] W. Fang, Y. Xu, T. Zhang, et al., Towards large language model for cognitive industrial mixed reality: A survey, Advanced Engineering Informatics 71 (2026) 104276. doi:https://doi.org/10.1016/j.aei.2025.104276.

[14] M. Raza, Z. Jahangir, M. B. Riaz, et al., Industrial applications of large language models, Scientific Reports 15 (1) (2025) 13755. doi:10.1038/s41598-025-98483-1.

[15] C. Qu, S. Dai, X. Wei, et al., Tool learning with large language models: A survey, Frontiers of Computer Science 19 (8) (2025) 198343. doi:10.1007/s11704-024-40678-2.

[16] M. Perry, J. Xu, E. Huang, et al., Robust sampling budget allocation under deep uncertainty, IEEE Transactions on Systems, Man, and Cybernetics: Systems 52 (10) (2022) 6339–6347. doi:10.1109/TSMC.2022.3144363.

[17] M. Zou, E. Huang, B. Vogel-Heuser, et al., Efficiently learning a distributed control policy in cyber-physical production systems via simulation optimization, in: 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), 2020, pp. 645–651. doi:10.1109/CASE48305.2020.9249228.

[18] P. Lewis, E. Perez, A. Piktus, et al., Retrieval-augmented generation for knowledge-intensive nlp tasks, in: Proceedings of the 34th International Conference on Neural Information Processing Systems, 2020. doi:10.5555/3495724.3496517.

[19] W. Fan, Y. Ding, L. Ning, et al., A survey on rag meeting llms: Towards retrieval-augmented large language models, in: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2024, pp. 6491–6501. doi:10.1145/3637528.3671470.

[20] K. M. Jablonka, P. Schwaller, A. Ortega-Guerrero, et al., Leveraging large language models for predictive chemistry, Nature Machine Intelligence 6 (2) (2024) 161–169. doi:10.1038/s42256-023-00788-1.

[21] Q. Ma, Y. Zhou, J. Li, Automated retrosynthesis planning of macromolecules using large language models and knowledge graphs, Macromolecular Rapid Communications (2025) 2500065doi:10.1002/marc.202500065.

[22] Y. Zheng, H. Y. Koh, J. Ju, et al., Large language models for drug discovery and development, Patterns 6 (10) (2025) 101346. doi:10.1016/j.patter.2025.101346.

[23] H. Guo, X. Xing, Y. Zhou, et al., A survey of large language model for drug research and development, IEEE Access 13 (2025) 51110–51129. doi:10.1109/ACCESS.2025.3552256.

[24] J. Ock, R. S. Meda, S. Badrinarayanan, et al., Large language model agent for modular task execution in drug discovery, Journal of Chemical Information and Modeling 66 (4) (2026) 2055–2068. doi:10.1021/acs.jcim.5c02454.

[25] A. Daareyni, A. Martikkala, H. Mokhtarian, et al., Generative ai meets cad: enhancing engineering design to manufacturing processes with large language models, The International Journal of Advanced Manufacturing Technology (2025). doi:10.1007/s00170-025-15830-2.

[26] H. Kourani, A. Berti, D. Schuster, W. M. P. van der Aalst, Process modeling with large language models, in: Enterprise, Business-Process and Information Systems Modeling, 2024, pp. 229–244. doi:10.1007/978-3-031-61007-3_18.

[27] H. Koziolek, S. Grüner, R. Hark, et al., Llm-based and retrieval-augmented control code generation, in: Proceedings of the 1st International Workshop on Large Language Models for Code, 2024, p. 22–29. doi:10.1145/3643795.3648384.

[28] J. A. Heredia Álvaro, J. G. Barreda, An advanced retrieval-augmented generation system for manufacturing quality control, Advanced Engineering Informatics 64 (2025) 103007. doi:10.1016/j.aei.2024.103007.

[29] J. Lee, J.-Y. Kim, H. Kim, et al., Im-chat: A multi-agent llm framework integrating tool-calling and diffusion modeling for knowledge transfer in injection molding industry, Journal of Manufacturing Systems 83 (2025) 839–855. doi:10.1016/j.jmsy.2025.11.007.

[30] Y. Pan, Q. Xiao, F. Zhao, et al., Chat-microreactor: A large-language-model-based assistant for designing continuous flow systems, Chemical Engineering Science 311 (2025) 121567. doi:10.1016/j.ces.2025.121567.

[31] Y. Bounab, O. Antikainen, M. Sivén, et al., Advancing direct tablet compression with ai: A multi-task framework for quality control, batch acceptance, and causal analysis, European Journal of Pharmaceutical Sciences 212 (2025) 107142. doi:10.1016/j.ejps.2025.107142.

[32] D. A. Boiko, R. MacKnight, B. Kline, et al., Autonomous chemical research with large language models, Nature 624 (7992) (2023) 570–578. doi:10.1038/s41586-023-06792-0.

[33] A. M. Bran, S. Cox, O. Schilter, et al., Augmenting large language models with chemistry tools, Nature Machine Intelligence 6 (5) (2024) 525–535. doi:10.1038/s42256-024-00832-8.

[34] Y. Ruan, C. Lu, N. Xu, et al., An automatic end-to-end chemical synthesis development platform powered by large language models, Nature Communications 15 (1) (2024) 10160. doi:10.1038/s41467-024-54457-x.

[35] S. Pandey, R. Xu, W. Wang, et al., Openfoamgpt: A retrieval-augmented large language model (llm) agent for openfoam-based computational fluid dynamics, Physics of Fluids 37 (3) (2025) 035120. doi:10.1063/5.0257555.

[36] Z. Dong, Z. Lu, Y. Yang, Fine-tuning a large language model for automating computational fluid dynamics simulations, Theoretical and Applied Mechanics Letters 15 (3) (2025) 100594. doi:10.1016/j.taml.2025.100594.

[37] Z. Liu, Y. Chai, J. Li, Toward automated simulation research workflow through llm prompt engineering design, Journal of Chemical Information and Modeling 65 (1) (2025) 114–124, pMID: 39801288. doi:10.1021/acs.jcim.4c01653.

[38] B. Ni, M. J. Buehler, Mechagents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge, Extreme Mechanics Letters 67 (2024) 102131. doi:10.1016/j.eml.2024.102131.

[39] A. Ghafarollahi, M. J. Buehler, Protagents: protein discovery via large language model multi-agent collaborations combining physics and machine learning, Digital Discovery 3 (7) (2024) 1389–1409. doi:10.1039/D4DD00013G.

[40] N. Yoshikawa, M. Skreta, K. Darvish, et al., Large language models for chemistry robotics, Autonomous Robots 47 (2023) 1057–1086. doi:10.1007/s10514-023-10136-2.

[41] X. Ren, Q. Zang, Z. Guo, Simugen: Multi-modal agentic framework for constructing block diagram-based simulation models, in: Workshop on Scaling Environments for Agents, 2025. doi:10.48550/arXiv.2506.15695.

[42] Z. Shi, S. Gao, X. Chen, et al., Learning to use tools via cooperative and interactive agents, in: Findings of the Association for Computational Linguistics: EMNLP 2024, 2024, pp. 10642–10657. doi:10.18653/v1/2024.findings-emnlp.624.

[43] L. E. Erdogan, N. Lee, S. Kim, et al., PLAN-AND-ACT: improving planning of agents for long-horizon tasks, in: Proceedings of the 42nd International Conference on Machine Learning, 2025. doi:10.5555/3780338.3780932.

[44] S. Yuan, K. Song, J. Chen, 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 (Volume 1: Long Papers), 2025, pp. 951–972. doi:10.18653/v1/2025.naacl-long.44.

[45] J. Lu, T. Holleis, Y. Zhang, B. Aumayer, F. Nan, H. Bai, S. Ma, S. Ma, M. Li, G. Yin, Z. Wang, R. Pang, ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities, in: Findings of the Association for Computational Linguistics: NAACL 2025, 2025, pp. 1160–1183. doi:10.18653/v1/2025.findings-naacl.65.

[46] S. Huang, K. Yang, S. Qi, et al., When large language model meets optimization, Swarm and Evolutionary Computation 90 (2024) 101663. doi:10.1016/j.swevo.2024.101663.

[47] R. Zhong, A. G. Hussien, J. Yu, et al., Llmoa: A novel large language model assisted hyper-heuristic optimization algorithm, Advanced Engineering Informatics 64 (2025) 103042. doi:10.1016/j.aei.2024.103042.

[48] A. AhmadiTeshnizi, W. Gao, M. Udell, Optimus: scalable optimization modeling with (mi)lp solvers and large language models, in: Proceedings of the 41st International Conference on Machine Learning, 2024. doi:10.5555/3692070.3692094.