



AI agents for automating materials research: a case study of crystal plasticity simulations

Jiyi Yang, Yoshinao Kobayashi & Masahiko Demura

To cite this article: Jiyi Yang, Yoshinao Kobayashi & Masahiko Demura (2026) AI agents for automating materials research: a case study of crystal plasticity simulations, Science and Technology of Advanced Materials: Methods, 6:1, 2630445, DOI: [10.1080/27660400.2026.2630445](https://doi.org/10.1080/27660400.2026.2630445)

To link to this article: <https://doi.org/10.1080/27660400.2026.2630445>



© 2026 The Author(s). Published by National Institute for Materials Science in partnership with Taylor & Francis Group



Published online: 02 Apr 2026.



Submit your article to this journal [↗](#)



Article views: 1064






View related articles [↗](#)



View Crossmark data [↗](#)

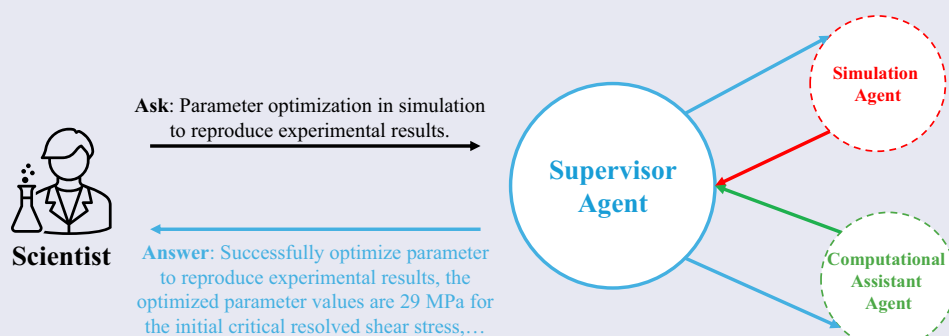
AI agents for automating materials research: a case study of crystal plasticity simulations

Jiyi Yang ^{a,b}, Yoshinao Kobayashi ^a and Masahiko Demura ^b

^aDepartment of Materials Science and Engineering, Institute of Science Tokyo, Tokyo, Japan; ^bCenter for Basic Research on Materials, National Institute for Materials Science (NIMS), Tsukuba, Japan

ABSTRACT

This work introduces a multi-agent system, CrystalPlasticitySim, which leverages large language models (LLMs) to automate complex workflows involved in crystal plasticity simulations. The system substantially reduces the human effort required to learn, prepare, and execute crystal plasticity simulations, enabling workflows that previously required months of manual effort to be completed within hours of autonomous execution. Traditionally, using crystal plasticity simulations requires not only deep expertise in materials science but also technical knowledge of a specific simulation package and computational skills – forcing users to manually configure parameters, prepare input files, and troubleshoot simulations through laborious trial-and-error processes. These technical hurdles limit accessibility and slow scientific progress. To overcome these challenges, CrystalPlasticitySim employs three collaborating AI agents – the Supervisor Agent, Simulation Agent, and Computational Assistant Agent – that autonomously generate input files, execute simulations, extract results, and even perform parameter optimization. Using a case study on the anisotropic behavior of Ni₃Al single crystals during cold rolling, we demonstrate that the system can autonomously solve well-defined tasks, thereby lowering the barrier to crystal plasticity modeling and improving accessibility to advanced simulation tools to researchers across disciplines.



IMPACT STATEMENT

This work introduces a multi-agent LLM framework that autonomously executes crystal plasticity simulations, significantly reducing manual effort and improving reproducibility in materials research workflows.

ARTICLE HISTORY

Received 6 November 2025
Revised 13 January 2026
Accepted 8 February 2026

KEYWORDS

AI agent; large language model; crystal plasticity simulation; simulation automation; data-driven approach

1. Introduction

Crystal plasticity simulation has become an essential tool for investigating the mechanical behavior and microstructural evolution of crystalline materials, providing insight into deformation mechanisms at multiple length scales. By explicitly accounting for crystallographic orientation and slip system activity, and – when available – incorporating microstructural evolution (e.g. dislocation density, internal variable dynamics), these models enable prediction of texture

evolution and mechanical responses under complex loading paths [1]. Despite these advantages, the adoption of crystal plasticity simulation remains limited by several barriers that make it inaccessible to non-experts and time-consuming even for experienced researchers.

First, simulation packages – such as DAMASK (Düsseldorf Advanced Material Simulation Kit) [2,3] – require users to manually create configuration files that define materials, load cases, and meshing schemes. This process demands fluency in package-

specific syntax and detailed knowledge of the simulation software. Even small errors in file structure can result in failed runs or unreliable results.

Second, parameter specification presents another major challenge. Defining material-specific properties such as slip system activity, critical resolved shear stresses, hardening laws, and constitutive behavior requires deep domain expertise. These quantities are often not directly measurable and must instead be inferred from experiments or estimated through iterative calibration. Such optimization is computationally intensive, highly sensitive to initial assumptions, and demands knowledge of computer and mathematical sciences.

Third, post-simulation analysis adds another layer of complexity. Extracting and visualizing result data such as stress – strain curves, slip system activity, or pole figures from output files requires specialized scripting and visualization tools. Researchers must often manually align the simulated and experimental textures to ensure consistent reference frames and write custom scripts to extract specific quantities such as local stress tensors and slip activities from output files. Visualization of pole figures and orientation distribution functions typically involves additional software and manual parameter tuning. Together, these steps introduce subjectivity, increase dependence on user expertise, and reduce reproducibility.

Moreover, crystal plasticity simulations are frequently performed on workstations, where users must also manage software installation, environment configuration, and dependencies. The large volume of data generated by high-fidelity simulations raises additional concerns about storage, provenance, and reproducibility. Collectively, these challenges create a fragmented and labor-intensive workflow that hinders broader adoption of crystal plasticity simulation.

Recent advances in LLMs, particularly the emergence of systems such as GPT, offer an opportunity to address these challenges. LLMs have demonstrated capabilities in scientific reasoning, code generation, tool invocation, and contextual memory [4,5]. Beyond these general abilities, the ReAct (Reason + Act) agent paradigm has been proposed as a framework in which LLMs interleave step-by-step reasoning with concrete actions in the environment [6]. This approach allows an LLM to not only generate solutions but also to execute tasks, interpret feedback, and iteratively refine its outputs. When embedded within multi-agent architectures, ReAct-style agents can coordinate and manage complex workflows in a closed-loop, adaptive fashion [7–10].

Motivated by these developments, in this study, we develop CrystalPlasticitySim, a multi-agent system designed to automate the entire crystal plasticity simulation pipeline. Three specialized agents – the Supervisor Agent, Simulation Agent, and Computational Assistant

Agent – collaborate to autonomously generate input files, execute simulations, optimize material parameters, recover from errors, and interpret results. The system is implemented using LangGraph [8], which provides a directed execution graph connecting agents through structured communication channels. By combining role-specific prompts, shared memory, and domain-specific tool interfaces, this framework achieves robust, self-correcting behavior that reduces the need for manual configuration, enhances reproducibility, and lowers the barrier to entry for high-fidelity crystal plasticity simulations.

We demonstrate the efficacy of CrystalPlasticitySim through a case study on the cold rolling of Ni₃Al single crystals using the DAMASK crystal plasticity simulation software. We previously investigated the deformation behavior of Ni₃Al [11] where we analyzed slip system activation, and slip-slip interactions during cold rolling. That study revealed the strong orientation dependence of mechanical responses in this intermetallic alloy, underscoring the need for accurate crystal plasticity simulation. In the present case study, we demonstrate that CrystalPlasticitySim not only generates input files, runs simulations, and post-processes result files, but also autonomously optimizes material parameters and boundary conditions to replicate the experimental observations with high fidelity. These results highlight the potential of integrating AI agents with domain-specific tools to enable more intelligent, reproducible, and accessible materials simulation workflows.

2. Methods

2.1. System architecture

CrystalPlasticitySim is a multi-agent system powered by LLMs, designed to automate DAMASK version 3.0 simulation workflows for crystal plasticity. As illustrated in Figure 1, the system architecture features three specialized agents: the Supervisor Agent, Simulation Agent, and Computational Assistant Agent. Together, these agents form a closed-loop control system capable of autonomously setting up, executing, and refining crystal plasticity simulations. The role and core functions of each agent are summarized in Table 1 and are described in detail in the following subsections.

2.1.1. Supervisor agent

The Supervisor Agent serves as the coordinator and decision-maker of the system. It receives high-level objectives from the user (e.g. ‘*Optimize the deformation parameters to match experimental results*’), decomposes these into actionable subtasks, and dispatches them to the appropriate agents as summarized in Table 2. It maintains a persistent memory of the

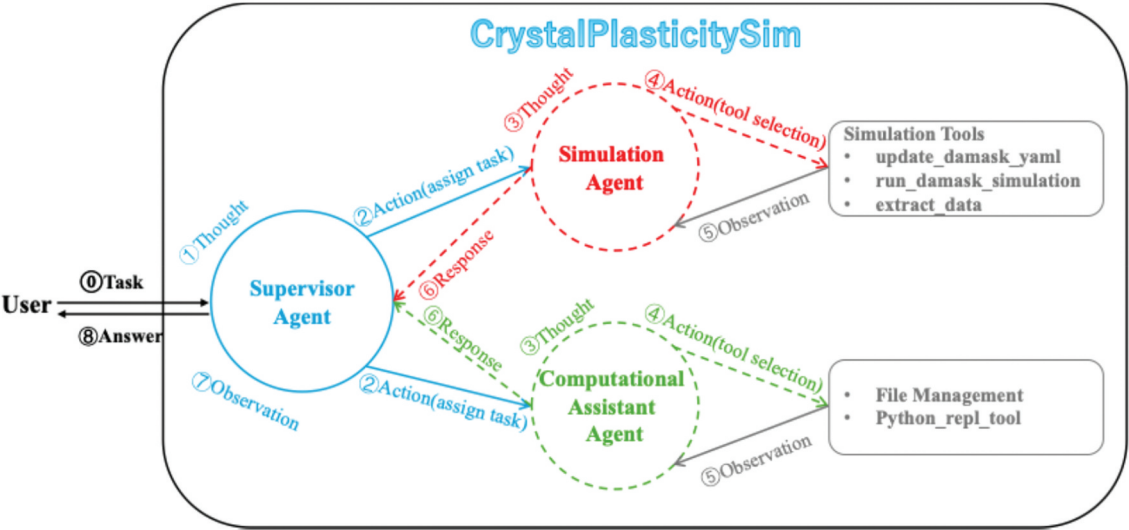


Figure 1. Architecture of the CrystalPlasticitySim multi-agent system.

Table 1. Overview of agent responsibilities and key functions in CrystalPlasticitySim.

Agent	Role	Core Functions
Supervisor Agent	Orchestrates the overall workflow by interpreting user queries, decomposing tasks, and delegating responsibilities to other agents.	<ul style="list-style-type: none">– Goal decomposition– Task delegation– Workflow coordination– Progress tracking and context memory
Simulation Agent	Executes simulations using external tools (e.g. DAMASK), retrieves outputs, and reports results.	<ul style="list-style-type: none">– Run simulation scripts– Parse and extract outputs– Return structured results
Computational Assistant Agent	Generates and modifies code or configuration files to optimize parameters based on real-time feedback.	<ul style="list-style-type: none">– Script generation– Input file editing– Parameter tuning and optimization– Runtime error correction

workflow’s state, tracking completed tasks, intermediate results, and convergence status. When the criteria are met, the Supervisor Agent halts the loop or initiates further iterations.

2.1.2. Simulation agent

The Simulation Agent is responsible for managing the core simulation workflow, focusing on three main tasks: pre-processing, execution, and post-processing as summarized in Table 2. In our implementation, it interfaces with DAMASK 3.0 to run crystal plasticity simulations.

During pre-processing, the Simulation Agent prepares the required input data, such as parameterized material configuration or load conditions YAML files, based on instructions provided by the Supervisor Agent. During execution, the Simulation Agent launches DAMASK simulations entirely within the local computational environment, continuously monitoring their completion. No intermediate files or sensitive data are transmitted outside the controlled environment – including to the LLM itself – thereby ensuring that proprietary simulation inputs and results remain secure. This design allows CrystalPlasticitySim to benefit from LLM-driven orchestration without

exposing material data or computational results to external servers. Finally, in the post-processing case, the agent extracts and organizes the relevant outputs – including crystal orientations, Cauchy stress values, and slip rates for each slip system and so on – into structured formats that can be interpreted by the Supervisor Agent.

By automating these three steps, the Simulation Agent eliminates the need for manual handling of input and output files and data extraction, ensuring that simulation results are consistently prepared for subsequent analysis and decision-making.

2.1.3. Computational assistant agent

The Computational Assistant Agent is responsible for code generation, program execution, and maintaining the local computational environment. As summarized in Table 2, this agent performs four core tasks: package installation, file management, code execution, and automated debugging.

The agent first ensures that the required computational environment is correctly configured. When instructed by the Supervisor Agent, it installs any missing Python packages or dependencies within a designated temporary workspace. This sandboxed

Table 2. Role-specific prompt design for the agents in CrystalPlasticitySim.

Agent	Prompt
Supervisor Agent	You are a supervisor tasked with managing a conversation between the following workers: {members}. Given the following user request, respond with the worker to act next. Each worker will perform a task and respond with their results and status. When finished, respond with FINISH.
Simulation Agent	You are an expert in materials science specializing in crystal plasticity modeling. Your role is to analyze and simulate material behaviors using the following tools: ... Given a user request, select the most appropriate tool(s) to process the task. Provide detailed and structured results based on scientific best practices.
Computational Assistant Agent	You are an expert Python programmer specializing in numerical optimization. Your role is to generate, modify, and execute scripts efficiently within a specified 'workdir' directory. The 'workdir' directory contains all necessary input files, and all operations should be performed using absolute paths ...

environment guarantees that all operations remain secure and reproducible while preventing conflicts between simulations. The installation process is logged for full auditability.

Within its assigned workspace, the agent handles all file-related operations, including creating, reading, modifying, and organizing Python scripts, YAML configuration files, and log files. Each file is tracked by version, and naming conventions encode relevant parameters for traceability. This ensures data provenance and facilitates reproducible reruns of the same simulation task.

When a new computational task is assigned, the agent generates Python scripts that implement the required operations (e.g. parameter optimization or result parsing). It executes these scripts and continuously monitors their output to detect runtime errors. The agent maintains absolute file paths for all inputs and outputs, ensuring consistent references across iterations.

If an error occurs during execution, the Computational Assistant Agent analyzes the logs or stack traces, identifies the cause, and automatically applies targeted fixes – such as adding missing imports, correcting file paths, or updating syntax. The revised script is saved as a new version (e.g. version_1.py, version_2.py) to preserve the debugging history. This self-healing process continues until each script runs successfully, enabling uninterrupted progress toward convergence without human intervention.

In essence, the Computational Assistant Agent functions as a hybrid programmer – debugger, dynamically maintaining a functional execution environment and automating the entire software lifecycle – from setup to recovery. This design ensures secure, reproducible, and traceable execution of all simulation tasks within the CrystalPlasticitySim system.

2.2. Implementation by LangGraph

The CrystalPlasticitySim system builds on recent advancements in LLM frameworks. Each agent is guided by role-specific prompts and toolkits. In this study, we used LangGraph [8] to implement the interaction system for CrystalPlasticitySim. To support transparency and reproducibility, the source code, prompts, and toolkits are provided in our GitHub

repository: <https://github.com/ForeverYoungJay/CrystalPlasticitySim>

2.2.1. Prompt design

Prompt engineering [12–15] was a key step in implementing CrystalPlasticitySim, as it determined how the agents coordinated tasks, handled errors, and produced reproducible outputs. In this study, we treated prompts not as casual queries but as carefully constructed research protocols that encode agent roles, goals, and actions in natural language. Well-designed prompts ensured that the agents performed domain-specific tasks with precision, avoided redundant work, and produced structured results that could be compared across multiple runs.

We first defined each agent's role and translated workflow requirements into precise natural-language instructions. For the Supervisor Agent, we emphasized orchestration and termination logic:

You are a supervisor tasked with managing a conversation between the following workers: Simulation Agent and Computational Assistant Agent ... When finished, respond with FINISH.

This design allowed the Supervisor Agent to decompose objectives, delegate subtasks in order, and terminate workflows once the objectives were achieved.

The Simulation Agent was framed as a domain expert in crystal plasticity modeling, tasked with preparing input files, running DAMASK simulations, and reporting results in a standardized format:

Your role is to analyze and simulate material behaviors ... Given a user request, select the most appropriate tool(s) to process the task. Provide detailed and structured results based on scientific best practices.

Finally, the Computational Assistant Agent was given explicit instructions to behave like a professional Python programmer:

You are an expert Python programmer specializing in numerical optimization. Your role is to generate, modify, and execute scripts efficiently within a specified workdir directory. All input and output files should be handled using absolute paths ... If errors occur during execution, generate a new version of the script with names version_1, version_2, etc., instead of overwriting the original script.

Our prompt design process followed an iterative approach. We began with baseline prompts derived from the desired workflow, then executed controlled test runs to evaluate behavior. Early iterations revealed issues such as incomplete YAML formatting, missing imports, and duplicated actions between agents. We revised the prompts to clarify task descriptions, add structured output requirements, and introduce error-handling instructions. For example, we added a step instructing the Computational Assistant Agent to identify and reuse existing Python functions within the work directory rather than redundantly regenerating them. We also required automatic installation of missing dependencies, which improved robustness during large batch simulations.

Excerpts of these prompts are listed in Table 2, allowing each agent to behave deterministically, work within well-defined boundaries, and collaborate seamlessly.

2.2.2. Toolkits

While prompt engineering defined the cognitive roles of each agent, the next step was to build role-specific toolkits that would enable these agents to perform concrete actions. Rather than relying solely on generic utilities provided by LangGraph, we designed and implemented a collection of modular Python functions tailored to the DAMASK simulation workflow. This ensured that every operation was auditable, reproducible, and directly linked to our research objectives.

Each agent was provided with tools carefully matched to its functional responsibilities, as listed in Table 3. The Supervisor Agent did not require domain-specific tools but was endowed with coordination and memory utilities to manage the flow of information and track convergence status. The Simulation Agent was equipped with three categories of tools – pre-processing, execution, and post-processing – allowing it to prepare input configurations, run DAMASK simulations, and extract results into structured formats. The Computational Assistant Agent was given file management and Python code execution capabilities, enabling it to create, modify, and debug scripts within the specified working directory.

By aligning toolkits with agent roles, we ensured that each agent had a unique and non-overlapping set

of capabilities, which reduced redundant work and increased determinism in the workflow. Taking the Simulation Agent as an example, we developed three core Python functions that were iteratively tested and refined to ensure stable integration with DAMASK:

damask_yaml.py: Provides utilities to modify DAMASK YAML inputs, including material configuration and load cases. Functions validate schema elements, update parameters (e.g. slip-related properties, deformation gradient components), and write out revised files with systematic filenames that encode key parameter values for traceability. All changes are logged.

damask_simulation.py: Implements a Python wrapper for DAMASK execution. The wrapper invokes DAMASK_grid, sets the working directory, and captures stdout/stderr. It writes a run log to damask_monitor.log, enforces unique result prefixes, and ensures each run produces a uniquely named.hdf5 output (e.g. timestamp or hash-based suffix). Return codes and run metadata (input file paths, seed, package version) are recorded for auditability.

damask_results.py: Parses DAMASK.hdf5 outputs and exports analysis-ready data.

Each agent was equipped with a set of Python functions to operationalize its role. To enhance reproducibility, domain knowledge was embedded directly into the docstrings of these functions. This practice ensured that usage requirements, parameter ranges, and output formats were explicitly documented within the code base, reducing ambiguity and enabling transparent audit trails. For example, in *damask_results.py*:

```
def calculate_deviation_angle(json_input: str) -> dict:
    """
    Calculate the deviation angle between simulated and
    experimental orientations.
    Requires a JSON input containing:
    - simulated_file: Name of the result file.
    - experimental_quaternion: A list representing the
    experimental quaternion [w, x, y, z].
    Returns a dictionary with the deviation angle (degrees)
    and simulated quaternion.
    """
```

In this example, the docstring specifies input requirements (JSON fields), expected output (dictionary with deviation angle and quaternion), and the physical context (comparison of simulated and experimental orientations). Similar documentation was added to

Table 3. Toolkits assigned to each agent in CrystalPlasticitySim.

Agent	Toolkit
Supervisor Agent	N/A
Simulation Agent	coordination and memory Pre-Processing Running the Simulation Post-Processing
Computational Assistant Agent	FileManagement (File I/O) Python_repl_tool (run code)

functions handling slip-related parameter ranges, optimization thresholds, and file-naming conventions. Embedding this information in natural language within the code reduced the need for external instructions and ensured that the reasoning process remained transparent, auditable, and reproducible.

Each function was manually and independently verified before integration into the agent workflow. We ran pilot simulations to confirm that YAML files were DAMASK-compatible, result files were correctly generated, and computed deviation angles matched expected values.

3. A four-Level taxonomy of AI agents in materials simulation

Artificial intelligence (AI) agents are increasingly integrated into materials simulation workflows, offering varying degrees of autonomy, reasoning, and interaction with computational tools.

To systematize this development, we define a four-level taxonomy (Figure 2), that captures progressive agentic capabilities – from basic language-based utilities to aspirational autonomous scientific agents.

This taxonomy focuses specifically on AI agents within materials simulation, emphasizing how they perceive goals, plan and execute simulations, and reason about results.

3.1. Level 1: tool-level agent

Level 1 agents act as assistants that respond to human prompts without memory or planning. They assist in isolated tasks such as generating simulation input scripts, explaining outputs, or summarizing literature. These agents lack any persistent state, workflow control, or capacity for reasoning beyond direct

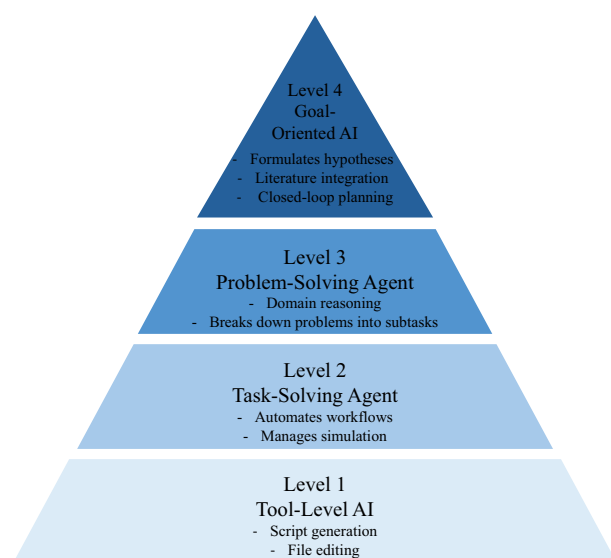


Figure 2. Taxonomy of AI agent levels in materials simulation workflows.

instruction. Examples include language-based utilities such as MechGPT [16], which generates mechanics code and explanations, and AutoFLUKA [17], which generates input files, runs simulations, and post-processes output files for Monte Carlo simulations in nuclear engineering. These models function as enhanced tools – useful for accelerating human work but not autonomous in any sense.

3.2. Level 2: task-solving agent

Level 2 agents can autonomously execute a single, well-defined simulation task. They use state memory, file handling, or APIs to plan and complete a bounded workflow, such as running a molecular-dynamics tensile test or optimizing a unit cell parameter. The Autonomous Intelligent Agents for Accelerated Materials Discovery system [18] automates data generation and analysis loops for predefined materials discovery problems.

3.3. Level 3: problem-solving agent

Level 3 agents reason beyond individual tasks by decomposing a human-provided hypothesis into subtasks, orchestrating multiple simulations, and integrating the results. They possess memory, multi-step planning, and adaptive feedback, but their high-level goals still originate from the human scientist. Several recent systems in materials simulation exemplify this level of autonomy: DREAMS [19] implements an agentic research engine for exploring design spaces via density-functional-theory simulations. MAPPS [20] integrates planning, physics-based simulation, and human-in-the-loop oversight for coordinated materials discovery. ToPolyAgent [21] uses multi-agent collaboration for coarse-grained polymer simulations with autonomous execution loops. Multi-Agent Alloy Design [22] employs LLM – GNN cooperation for alloy optimization and design reasoning. These studies are all very recent, and Level 3 agents are still in their infancy. So far, reported systems have been designed for specific types of simulations. In particular, to the best of our knowledge, no Level 3 agent has yet been developed for crystal plasticity simulations such as those targeted in this study.

3.4. Level 4: goal-oriented AI

Level 4 agents would possess full scientific autonomy – the ability to formulate their own hypotheses, design and execute simulations, interpret results, and revise research directions without human intervention. Such an agent would complete the entire scientific cycle: literature → hypothesis → simulation → interpretation → new hypothesis. No published work has yet demonstrated a true Level 4 agent in materials simulation. Closest are

adjacent efforts: AutoLabs [23], a cognitive multi-agent system for autonomous chemical experimentation, integrates self-correction and planning but relies on user-provided objectives. The Agentic Science Survey [24] outlines conceptual blueprints for goal-setting scientific agents yet acknowledges that full autonomy remains an open challenge in materials domains.

In this study, we position CrystalPlasticitySim as a Level 2 \rightarrow early Level 3 agent within the proposed taxonomy. The system autonomously orchestrates crystal plasticity simulations using DAMASK and related computational tools. It performs bounded workflow automation – including task planning, parameter generation, simulation execution, result parsing, and self-correction – without direct user intervention once a specific objective is defined. Beyond Level 2 behavior, CrystalPlasticitySim incorporates state memory, error recovery, and adaptive control, allowing iterative exploration of simulation conditions. However, its objectives and hypotheses are still provided by the human researcher; the agent does not independently generate or reformulate research questions. Accordingly, while it exceeds single-task automation (Level 2) by exhibiting limited reasoning and self-reflection characteristic of Level 3 systems, it remains goal-conditioned by human hypotheses rather than autonomously generating new ones. This capability places CrystalPlasticitySim at the transition between task-solving and problem-solving agents, marking an important step toward more autonomous materials-simulation frameworks.

4. Case study: autonomous crystal plasticity simulation of Ni₃Al

4.1. Background and motivation

Unlike typical intermetallic compounds, nickel aluminide (Ni₃Al) can be heavily cold rolled into foils in the form of single crystals [25,26], however it exhibits pronounced anisotropic rolling behavior probably due to its ordered L1₂ crystal structure. During cold rolling, Ni₃Al demonstrates orientation-dependent deformation [27], necessitating crystal plasticity modeling to capture the evolution of slip activity and lattice orientation. This case study illustrates how CrystalPlasticitySim autonomously performs crystal plasticity simulations through a closed-loop workflow. In this case study, all simulations were performed using the phenomenological crystal plasticity model implemented in DAMASK. Detailed constitutive definitions and numerical integration settings are provided in Appendix A. The research process for anisotropic behavior comprises four key steps:

- (1) Optimize material parameters, as listed in Table 4, to reproduce experimental tensile stress – strain curves of single crystals.

- (2) Optimize boundary conditions to replicate experimentally observed orientation during cold rolling as shown in Table 5.
- (3) Identify activated or restrained slip systems by comparing with ideal plane strain condition.
- (4) Elucidate slip system interactions that lead to anisotropic behavior.

In this study, CrystalPlasticitySim effectively automates and accelerates the first two steps. The detailed materials science insights obtained from the subsequent steps of this research, such as the interpretation of slip system activity, are beyond the scope of the present paper and will be reported separately in a dedicated publication.

4.2. Case #1: optimize material parameters

4.2.1. Subtask objective and background

To ensure accurate representation of the crystal plasticity behavior of Ni₃Al single crystals, calibration of slip-related material parameters was carried out. The objective was to minimize the deviation between the simulated and experimentally measured uniaxial tensile stress – strain response by optimizing key parameters in the crystal plasticity model. Specifically, the experimental stress – strain dataset was taken from Demura and Hirano [28], which reports tensile stress responses of binary stoichiometric Ni₃Al single crystals tested over a range of temperatures and strain-rate changes. These parameters govern slip system kinetics and hardening behavior, both of which are critical in capturing the deformation characteristics of Ni₃Al. The targeted parameters and their corresponding bounds, as listed in Table 4, included the initial critical resolved shear stress, the saturation shear stress, and the initial hardening modulus.

4.2.2. Query setting

The query was designed as follows:

Optimize the slip-related material parameters in the DAMASK material file for the Ni₃Al17-A1 alloy to reduce the mean absolute percentage error between the experimental and simulated stress – strain curves, despite the noncontinuous nature of the experimental data. Focus on tuning the following parameters: the initial critical shear stress for slip (range: 27–90 MPa), the maximum critical shear stress for slip (range: 1000–5000 MPa), and the initial hardening modulus

Table 4. Targeted material parameters and their physically meaningful ranges for slip-related calibration in the DAMASK crystal plasticity model.

Parameter	Range (MPa)
Initial critical resolved shear stress (S_0)	27–90
Saturation shear stress (S_∞^a)	1000–5000
Initial hardening modulus (h_0)	100–500

for slip – slip interactions (range: 100–500 MPa). The relevant DAMASK simulation files and experimental data are located in the `workdir` folder. Apply an efficient optimization algorithm to identify the parameter set that minimizes the error. During the optimization process, log the results of each iteration in a file named `optimization_results.csv`, including the current values of all parameters and the corresponding computed error.

4.2.3. Results

The Supervisor Agent decomposed the overall optimization task and coordinated the overall workflow. Following its instructions, the Computational Assistant Agent autonomously selected an optimization strategy – Differential Evolution (DE) – from the available library functions, configured its hyperparameters (population size, iteration limit, and tolerance), generated the corresponding Python scripts, and executed the optimization loop; the full configuration is provided in [Appendix B](#) (B.2). During each iteration, it updated material parameters in the material configuration YAML files, launched DAMASK simulations, collected results, and logged all data for later analysis. After completion, the Computational Assistant Agent returned a summary report to the Supervisor, which instructed the Simulation Agent to verify the optimized parameters by running a validation simulation, parsing the outputs, and evaluating the final error. The Supervisor Agent then confirmed convergence and terminated the workflow.

Table 6 summarizes the optimized slip-related parameters, and Figure 3 shows the stress – strain curve obtained using these parameters. The computed curve (red, dashed line) closely reproduces the experimental response (blue, solid line), achieving a Mean Absolute Percentage Error (MAPE) of less than 2%. The iteration process plotted in Figure 4 indicates convergence within approximately 50 iterations.

4.3. Case #2: optimize boundary conditions

4.3.1. Subtask objective and background

The second case of this study focused on reproducing the final crystal orientation observed experimentally after cold rolling of Ni_3Al single crystals. During rolling, the imposed deformation gradient – particularly its shear components – strongly governs lattice rotation and slip activity. However, unlike macroscopic shape change or thickness reduction, these shear components denoted as F_{12} , F_{13} , F_{23} cannot be directly measured in experiments, as they represent internal deformation modes rather than externally observable geometry. Consequently, the precise deformation path that leads from the initial to the final orientation remains uncertain and must be inferred through simulation.

To resolve this, the optimization task was formulated to determine the most appropriate shear components of the deformation gradient that reproduce the experimentally measured final orientation, as shown

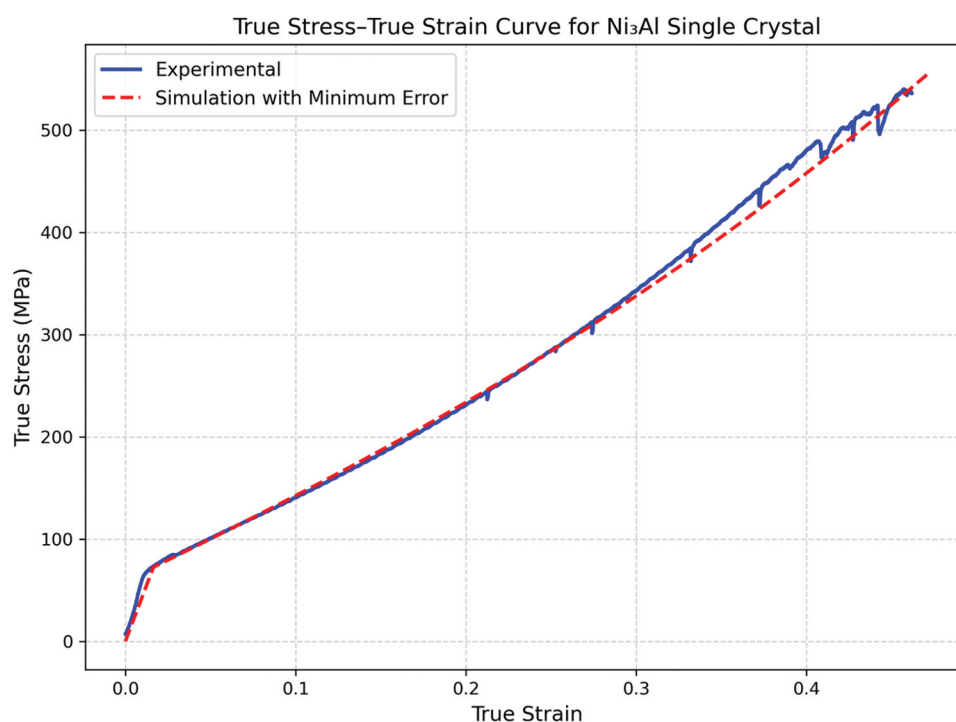


Figure 3. Comparison of experimental true stress – true strain curve (blue) and the simulation result obtained with the optimized parameters (red).

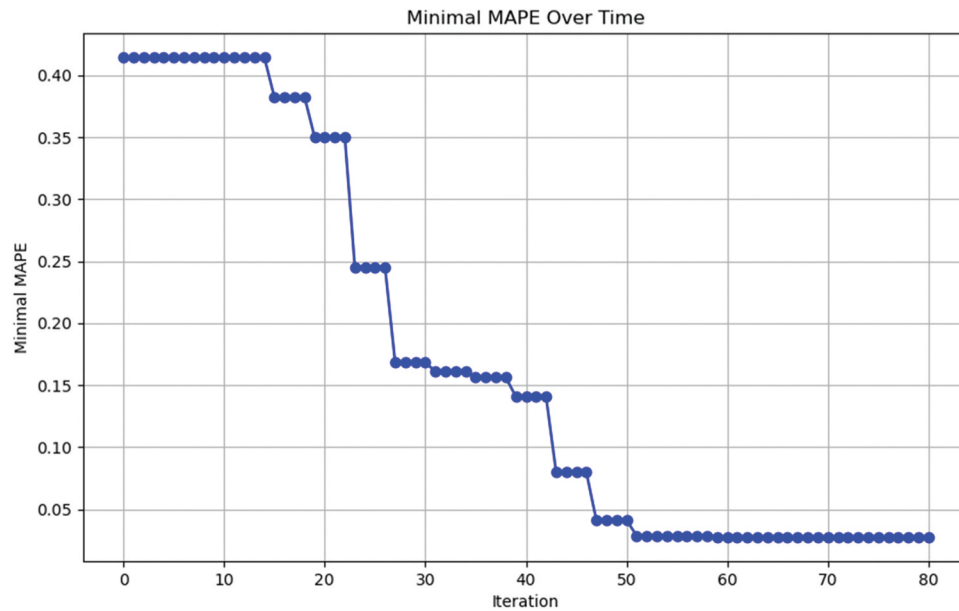


Figure 4. Evolution of minimal mean absolute percentage error (MAPE) during optimization of slip-related parameters.

in Table 5. The simulation was initialized using the experimentally measured starting orientation, and the optimization sought deformation conditions under which the final simulated quaternion orientation matched the experimental result with minimal deviation. Here, crystallographic orientations are represented using unit quaternions. A quaternion $q = (q_0, q_1, q_2, q_3)$ represents a rotation by an angle θ about a unit axis $u = (u_x, u_y, u_z)$ through the relation:

$$q = \left(\cos \frac{\theta}{2}, u_x \sin \frac{\theta}{2}, u_y \sin \frac{\theta}{2}, u_z \sin \frac{\theta}{2} \right) \quad (1)$$

4.3.2. Query setting

The query was designed as follows:

*Optimize the three independent deformation gradient components (F_{12} , F_{13} , F_{23}) in the load file within the range of -0.00300 to 0.00300 . The objective is to minimize the deviation angle between the simulated orientation quaternion obtained from the DAMASK simulation and the experimental target quaternion $[0.03451538, 0.56773038, 0.38495706, 0.72684178]$, with a threshold of 1 degree. The initial DAMASK configuration files are located in the workdir folder. The optimization should proceed iteratively by adjusting F_{12} , F_{13} , and F_{23} , executing the DAMASK simulation, and calculating the resulting deviation angle. An efficient optimization algorithm should be applied to determine the optimal deformation gradient values. During each iteration, log the values of F_{12} , F_{13} , and F_{23} , the resulting simulated quaternion, and the corresponding deviation angle in a CSV file named *optimization_results.csv*. Additionally, generate and save a figure illustrating the evolution of the best (minimum) deviation angle throughout the optimization process.*

4.3.3. Results

The Supervisor Agent decomposed the orientation-matching objective into an iterative optimization task and coordinated the workflow. Following its instructions, the Computational Assistant Agent autonomously selected the optimization algorithm – L-BFGS-B – configured its hyperparameters, generated the Python optimization script, and executed the iterative process; the full configuration is provided in Appendix B (B.3). In each cycle, it updated the deformation-gradient components in the load YAML file, ran the DAMASK simulation, calculated the deviation angle between the simulated and experimental orientations, and recorded all intermediate data. After completing the iterations, the Computational Assistant Agent reported the results to the Supervisor Agent, which instructed the Simulation Agent to verify the optimized deformation gradients by running a final validation simulation, parsing the outputs, and computing the deviation angle. The Supervisor Agent then assessed the evaluation and terminated the workflow.

Table 7 lists the optimized deformation-gradient components, and Figure 5 shows that the simulated final quaternion exactly reproduced the target experimental

Table 5. Experimental initial and final quaternions during cold-rolling.

Experimental initial quaternion	Experimental final quaternion
$[0.040, -0.503, -0.363, -0.782]$	$[0.034, 0.567, 0.384, 0.726]$

Table 6. Optimized slip-related material parameters.

Parameter	Optimized Value (MPa)
Initial critical resolved shear stress (S_0)	29.0
Saturation shear stress (S_∞)	3500.0
Initial hardening modulus (h_0)	208.0

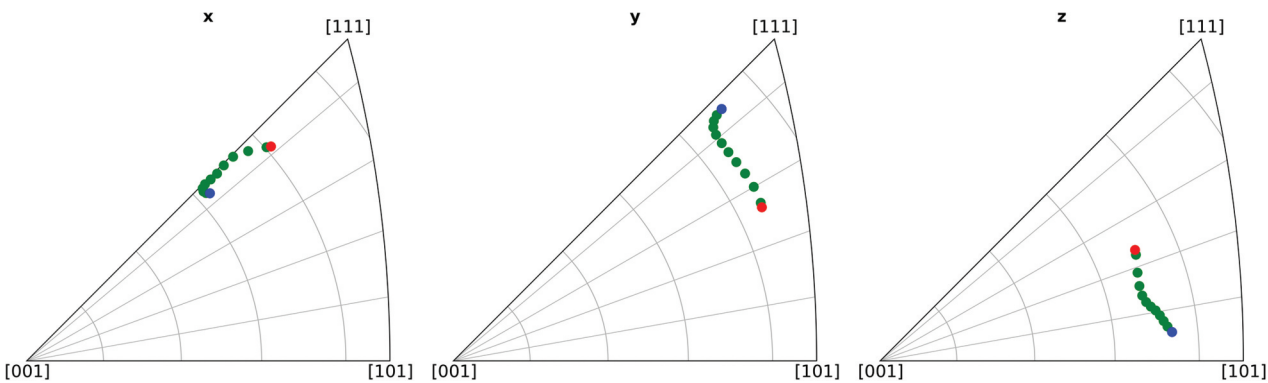


Figure 5. Inverse Pole figures showing orientation evolution during optimization and the final match with the experimental target.

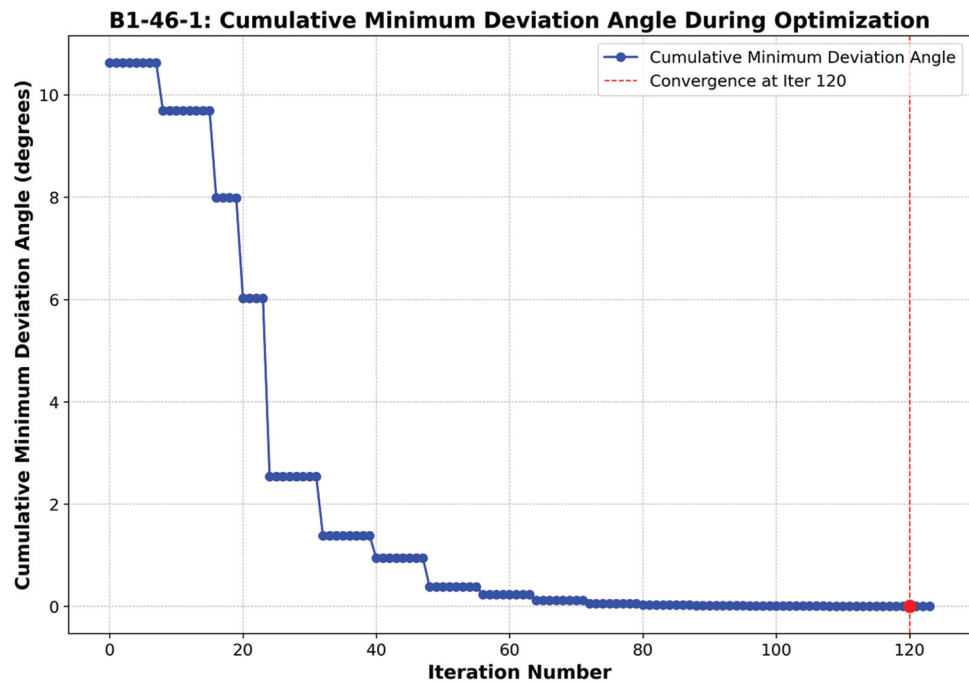


Figure 6. Convergence curve showing cumulative minimum misorientation angle approaching zero.

orientation with a 0.0° deviation angle. The optimization trajectory in Figure 6 illustrates that the deviation angle decreased from approximately 13° at the initial guess to 0.0° at convergence within 120 iterations.

5. Discussion

5.1. The workflow analysis

In CrystalPlasticitySim, the user provides initial simulation files and a high-level optimization objective. In detail, the user provides a partially specified DAMASK simulation setup, including: a material configuration file in which the value for each parameter to be optimized are not given or initialized; a load file defining the deformation mode, in which selected deformation-

gradient components are treated as optimization variables; a geometry file specifying the regular grid domain and material IDs for the DAMASK grid solver. Target outputs and optimization goals are given in the natural-language query. Beyond these inputs, all remaining steps – including optimization script generation, file modification, simulation execution, post-processing, and convergence evaluation – are carried out autonomously by the agents.

The workflow of CrystalPlasticitySim during autonomous optimization is illustrated in Figure 7, which summarizes how the system executed the tasks in both case studies. The diagram was reconstructed from the agents’ interaction logs and depicts three color-coded panels— (a) Supervisor Agent (blue), (b) Computational Assistant Agent (green), and (c) Simulation Agent (orange) –

Table 7. Optimized deformation gradient components (F_{12} , F_{13} , F_{23}) and orientation deviation.

Simulation final quaternion	F_{12}	F_{23}	F_{13}	Deviation angle
[0.034, 0.567, 0.384, 0.726]	1.410×10^{-3}	0.653×10^{-3}	0.354×10^{-3}	0.000

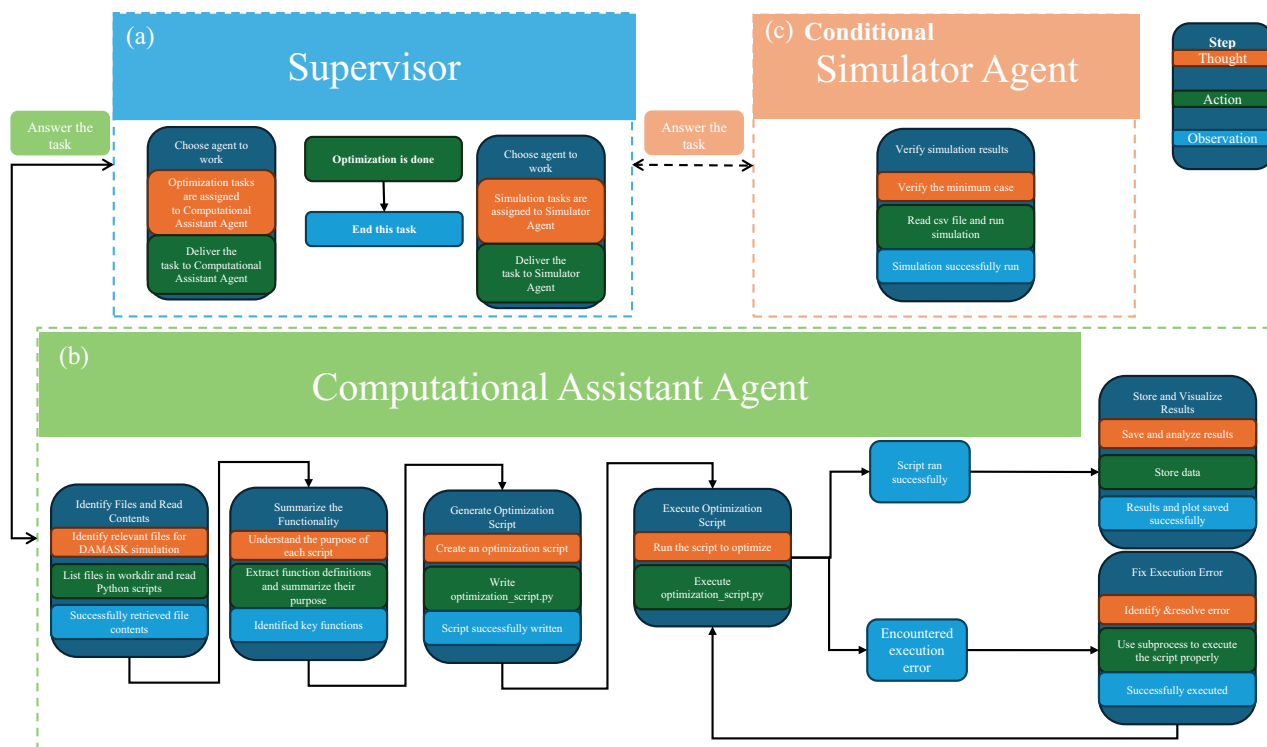


Figure 7. Workflow diagram of the CrystalPlasticitySim multi-agent system during autonomous optimization tasks. The figure illustrates the interaction between three agents: (a) Supervisor Agent (blue), (b) Computational Assistant Agent (green), and (c) simulation Agent (orange). Each rounded rectangle represents a discrete process node following a thought → action → observation reasoning structure, with color-coded labels indicating the corresponding cognitive step. Solid arrows denote routine execution paths, while dashed arrows indicate conditional control flow.

connected by arrows indicating control and data flow. Solid arrows represent routine execution paths that occurred in every workflow cycle, whereas the dashed arrows between panels (a) and (c) denoted conditional execution, invoked only when physical simulation validation was required. Each panel follows the same underlying reasoning structure: Thought → Action → Observation, illustrated through stacked color blocks within each process node.

The workflow begins in panel (a) where the Supervisor Agent interpreted a user request – such as ‘optimize material parameters’ or ‘reproduce experimental orientation’. It then decomposed the request into subtasks and delegated them to the appropriate agents. For both the present use cases, the Supervisor Agent delegated the optimization subtasks to the Computational Assistant Agent (panel b) and the workflow within the Computational Assistant Agent was common to both cases. That is, it autonomously identified required input files, read available scripts, extracted key functions, and generated an executable optimization routine. The agent selected a suitable algorithm (e.g. Differential Evolution or L-BFGS-B), set its hyperparameters, and ran the iterative process. When errors occurred, it interpreted the log files, applied corrective modifications, and re-executed until successful completion. All intermediate results were logged, visualized, and version-controlled to ensure reproducibility.

After the optimization subtask was completed, the Supervisor Agent delegated the validation subtask to the Simulation Agent (panel c) in the second case study. It executed simulations under the optimized deformation-gradient conditions, parsed the resulting orientations, and reported deviation angles back to the Supervisor Agent for convergence checking.

However, activation of the Simulation Agent (panel c) was conditional for several reasons. First, the Computational Assistant Agent could complete not only the optimization step but also the simulation step independently because it was able to use all tools in the toolkit by reading the files. Second, in the present study, the case studies involved optimization, where physical simulation validation may not necessarily be required; however, if the request is a simple forward simulation task, the Supervisor Agent will delegate execution directly to the Simulation Agent. Third, since the LLM itself is a probabilistic system, small variations in its reasoning process can lead the Supervisor to occasionally omit or invoke the Simulation Agent differently, even under similar conditions.

5.2. Failure profile and self-healing behavior

A critical requirement for deploying LLM-driven automation in scientific simulation workflows is robustness against common execution-level failures. In the present study, the failure profile of

CrystalPlasticitySim was limited and well characterized, and the system was able to autonomously recover from most practical errors encountered during execution. Notably, no YAML schema or formatting errors were observed during production runs. This can be attributed to the fact that we supplied a template of each configuration YAML file; the template was structurally valid, though it was incomplete. The agents subsequently modified only parameter values while preserving the original schema, preventing common template-breaking errors that frequently occur when LLMs generate YAML from scratch.

The primary observed failure modes included missing software dependencies, incorrect file paths or missing files, and occasional language-model token-limit exhaustion. Dependency-related failures primarily occurred when the workflow was executed in a clean environment lacking required Python modules or DAMASK-related packages. These failures were detected automatically through runtime exceptions and were typically resolved by the Computational Assistant Agent through dependency installation, after which the workflow continued without human intervention. Similarly, file path and I/O errors (e.g. missing experimental data files or mismatched filenames) were detected during execution and commonly resolved through directory re-scanning, reconstruction of absolute paths, and consistent updating of internal references.

In contrast, token-limit exhaustion represents a structural limitation of current LLM context lengths rather than a simulation-level error. This issue occurred rarely but was observed when the system repeatedly processed long experimental stress – strain datasets or verbose logs. Unlike dependency and path-related failures, token-limit exhaustion cannot be corrected through iterative self-healing, because the LLM cannot reliably continue reasoning once context capacity is exceeded. In such cases, the workflow terminates and must be restarted using a reduced-context strategy, such as truncating the dataset, using summarized inputs, or limiting the verbosity of intermediate logs.

During system development, additional failure cases were encountered and mitigated through iterative refinement of agent prompts. Once prompts were revised to correctly handle specific error types, the system consistently avoided the same failure in subsequent runs. This behavior constitutes development-time stabilization rather than run-time self-healing, highlighting that prompt engineering remains an important practical step in building robust agentic scientific workflows.

Overall, the results demonstrate that CrystalPlasticitySim can reliably converge without human intervention for common execution-level errors (dependency and path mismatches) after prompt stabilization, while token-limit exhaustion remains an external limitation of current LLM technology. A systematic

quantitative benchmark of failure rates and recovery cycles was not performed in this study; however, the observed recovery behavior supports the feasibility of self-healing automation in materials simulation pipelines and motivates future work on controlled robustness evaluation and improved context-management strategies.

5.3. Adaptive algorithm selection

CrystalPlasticitySim performs parameter calibration through a closed-loop workflow consisting of (i) updating DAMASK input files, (ii) executing DAMASK simulations, and (iii) evaluating an objective function against experimental data. The optimization algorithm is autonomously selected and configured based on the dimensionality of the search space, expected smoothness of the objective landscape, and the presence of physical bounds.

In Case #1, the system employs Differential Evolution (DE) implemented to minimize the discrepancy between simulated and experimentally measured stress – strain curves. The Computational Assistant Agent updates the target constitutive parameters in the DAMASK material configuration file and evaluates each candidate parameter set by running DAMASK and computing the error metric. In Case #2, the system uses L-BFGS-B to minimize the misorientation angle between simulated and experimentally measured final crystal orientations by optimizing the shear components of the deformation gradient under box constraints. Complete definitions of objective functions, parameter bounds, solver settings, and stopping criteria are provided in [Appendix B](#), enabling full reproducibility of the optimization procedures.

5.4. System performance, limitations, and extensibility

From an operational perspective, the agents collaboratively maintained repeatable, logged execution, version-controlled data management, and structured logging, ensuring full reproducibility and traceability of results. The Computational Assistant's self-healing mechanisms automatically detected and corrected runtime errors, allowing the workflow to continue without human intervention. Based on the author (Y.J.Y.)'s prior experience, conventional crystal plasticity calibration often involves substantial end-to-end human effort, spanning software learning, input preparation and validation, optimization scripting and debugging, and repeated simulation iterations; collectively, this process can take weeks to months. CrystalPlasticitySim reduces this end-to-end burden by automating the major workflow components after initialization, including script generation, iterative file modification, DAMASK execution, and objective evaluation. For example, for the Ni₃Al tensile calibration in Case #1,

the end-to-end automated optimization pipeline completed in approximately 12 minutes of wall-clock time (16:47:55 to 17:00:00, from execution logs), including file updates, DAMASK executions, and objective evaluation. This demonstrates a substantial gain in efficiency and a marked reduction in human effort while maintaining full reproducibility and consistency of results.

Despite these advantages, several limitations remain. The current implementation performs effectively only when provided with a well-defined, quantitative objective – for example, minimizing an error metric or deviation angle. Ambiguous or qualitative goals cannot yet be translated into executable actions, restricting the system to clearly structured optimization tasks. Furthermore, the agents still rely on human assistance for environment preparation and maintenance, including software installation, directory management, and solver compatibility (e.g. DAMASK versions). These dependencies can affect portability and may require expert oversight when deploying the framework across heterogeneous computing environments. Addressing such issues through containerized execution, automated dependency detection, or environment-awareness modules would further enhance robustness and autonomy.

The architecture of CrystalPlasticitySim is inherently extensible beyond crystal plasticity simulations. Its modular agent design and toolkit-based implementation enable seamless integration with other computational frameworks such as phase-field modeling, molecular dynamics, or finite element analysis. Each agent communicates through standardized data schemas and reasoning prompts that define the simulation objectives, required input files, and expected outputs. By modifying only a minimal set of prompt templates and interface toolkits – for instance, adapting file parsers or execution commands – the system can interact with new solvers without altering its core logic. This design minimizes redevelopment effort when transitioning to other materials modeling environments and ensures consistent provenance tracking and error-handling behavior across platforms. Consequently, the multi-agent framework provides a scalable foundation for autonomous coordination of diverse simulation workflows, advancing reproducibility and cross-domain scientific computation.

In summary, the dual-case demonstration illustrated how a multi-agent LLM framework can autonomously manage distinct classes of optimization tasks, adapt algorithm choice to task complexity, and maintain reproducible scientific workflows. While current limitations remain in hypothesis generation and computational scaling, the framework established a foundation for AI-assisted, self-improving materials simulation systems capable of bridging human reasoning and computational experimentation.

6. Conclusion

We have developed and demonstrated CrystalPlasticitySim, a multi-agent system that leverages LLMs to automate and optimize materials simulation workflows. In our case study on cold rolling of Ni₃Al single crystals, the system's Supervisor, Simulation, and Computational Assistant agents worked together to execute DAMASK crystal plasticity simulations and iteratively refine the input parameters, successfully minimizing the difference between simulation results and experimental data. This closed-loop automation required little to no human intervention beyond the initial instructions, highlighting the potential of AI agents to perform complex research tasks in materials science. Future work will extend this framework to other material systems and incorporate reinforcement learning to enable adaptive hypothesis generation.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by Ministry of Education, Culture, Sports, Science and Technology (MEXT) Program: Data Creation and Utilization-Type Material Research and Development Project Grant Number [JPMXP1122684766].

ORCID

Jiyi Yang  <http://orcid.org/0009-0003-0213-1258>

Yoshinao Kobayashi  <http://orcid.org/0000-0002-0908-9377>

Masahiko Demura  <http://orcid.org/0000-0002-7308-3041>

Data availability statement

The data that support the findings of this study are openly available at <https://github.com/ForeverYoungJay/CrystalPlasticitySim>.

References

- [1] Roters F, Eisenlohr P, Hantcherli L, et al. Overview of constitutive laws, kinematics, homogenization and multiscale methods in crystal plasticity finite-element modeling: theory, experiments, applications. *Acta Materialia*. 2010;58(4):1152–1211. doi: [10.1016/j.actamat.2009.10.058](https://doi.org/10.1016/j.actamat.2009.10.058)
- [2] Roters F, Diehl M, Shanthraj P, et al. Damask – the Düsseldorf advanced material simulation kit for modeling multi-physics crystal plasticity, thermal, and damage phenomena from the single crystal up to the component scale. *Comput Mater Sci*. 2019;158:420–478. doi: [10.1016/j.commatsci.2018.04.030](https://doi.org/10.1016/j.commatsci.2018.04.030)

- [3] Roters F, Eisenlohr P, Kords C, et al. Damask: the Düsseldorf advanced material simulation kit for studying crystal plasticity using an FE based or a spectral numerical solver. *Procedia IUTAM*. 2012;3:3–10. doi: [10.1016/j.piutam.2012.03.001](https://doi.org/10.1016/j.piutam.2012.03.001)
- [4] Open AI, Achiam J, Adler S, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774. 2024.
- [5] Lei G, Docherty R, Cooper SJ. Materials science in the era of large language models: a perspective. *Digit Discov*. 2024;3(7):1257–1272. doi: [10.1039/D4DD00074A](https://doi.org/10.1039/D4DD00074A)
- [6] Yao S, Zhao J, Yu D, et al. React: synergizing reasoning and acting in language models. arXiv:2210.03629. 2023. doi: [10.48550/arXiv.2210.03629](https://doi.org/10.48550/arXiv.2210.03629)
- [7] Wu Q, Bansal G, Zhang J, et al. AutoGen: enabling next-gen LLM applications via multi-agent conversation. arXiv:2308.08155. 2023. doi: [10.48550/arXiv.2308.08155](https://doi.org/10.48550/arXiv.2308.08155)
- [8] LangGraph [Internet]. [cited 2025 Oct 6]. Available from: <https://www.langchain.com/langgraph>
- [9] Talebirad Y, Nadiri A. Multi-agent collaboration: harnessing the power of intelligent LLM agents. arXiv:2306.03314. 2023. doi: [10.48550/arXiv.2306.03314](https://doi.org/10.48550/arXiv.2306.03314).
- [10] Li X, Wang S, Zeng S, et al. A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*. 2024;1(1):9. doi: [10.1007/s44336-024-00009-2](https://doi.org/10.1007/s44336-024-00009-2)
- [11] Demura M, Raabe D, Roters F, et al. Slip system analysis in the cold rolling of a Ni3Al single crystal. *Mater Sci Forum*. 2014;783–786:1111–1116. doi: [10.4028/www.scientific.net/MSF.783-786.1111](https://doi.org/10.4028/www.scientific.net/MSF.783-786.1111)
- [12] White J, Fu Q, Hays S, et al. A prompt pattern catalog to enhance prompt engineering with ChatGPT. In: *Proceedings of the 30th Conference on Pattern Languages of Programs; USA*. The Hillside Group; 2023. p. 1–31.
- [13] Giray L. Prompt engineering with ChatGPT: a guide for academic writers. *Ann Biomed Eng*. 2023;51(12):2629–2633. doi: [10.1007/s10439-023-03272-4](https://doi.org/10.1007/s10439-023-03272-4)
- [14] Marvin G, Hellen N, Jjingo D, et al. Prompt engineering in large language models. In: Jacob IJ, Piramuthu S, Falkowski-Gilski P, editors. *Data intelligence and cognitive informatics*. Singapore: Springer Nature; 2024. p. 387–402. doi: [10.1007/978-981-99-7962-2_30](https://doi.org/10.1007/978-981-99-7962-2_30)
- [15] Prompt engineering for ChatGPT: a quick guide to techniques, tips, and best practices [Internet]. [cited 2025 Oct 6]. Available from: <https://www.techrxiv.org/doi/full/10.36227/techrxiv.22683919>
- [16] Buehler MJ. MechGPT, a language-based strategy for mechanics and materials modeling that connects knowledge across scales, disciplines and modalities. arXiv:2310.10445. 2023. doi: [10.48550/arXiv.2310.10445](https://doi.org/10.48550/arXiv.2310.10445)
- [17] Ndum ZN, Tao J, Ford J, et al. Automating Monte Carlo simulations in nuclear engineering with domain knowledge-embedded large language model agents. *Energy AI*. 2025;21:100555. doi: [10.1016/j.egyai.2025.100555](https://doi.org/10.1016/j.egyai.2025.100555)
- [18] Montoya JH, Winther KT, Flores RA, et al. Autonomous intelligent agents for accelerated materials discovery. *Chem Sci*. 2020;11(32):8517–8532. doi: [10.1039/D0SC01101K](https://doi.org/10.1039/D0SC01101K)
- [19] Wang Z, Huang H, Zhao H, et al. Dreams: density functional theory based research engine for agentic materials simulation. arXiv:2507.14267. 2025. doi: [10.48550/arXiv.2507.14267](https://doi.org/10.48550/arXiv.2507.14267)
- [20] Zhou L, Ling H, Yan K, et al. Toward greater autonomy in materials discovery agents: unifying planning, physics, and scientists. arXiv:2506.05616. 2025. doi: [10.48550/arXiv.2506.05616](https://doi.org/10.48550/arXiv.2506.05616)
- [21] Ding L, Carrillo J-M, Do C. ToPolyAgent: AI agents for coarse-grained topological polymer simulations. arXiv:2510.12091. 2023. doi: [10.48550/arXiv.2510.12091](https://doi.org/10.48550/arXiv.2510.12091)
- [22] Ghafarollahi A, Buehler MJ. Rapid and automated alloy design with graph neural network-powered LLM-driven multi-agent systems. arXiv:2410.13768. 2024. doi: [10.48550/arXiv.2410.13768](https://doi.org/10.48550/arXiv.2410.13768)
- [23] Panapitiya G, Saldanha E, Job H, et al. Autolabs: cognitive multi-agent systems with self-correction for autonomous chemical experimentation. arXiv:2509.25651. 2025. doi: [10.48550/arXiv.2509.25651](https://doi.org/10.48550/arXiv.2509.25651)
- [24] Wei J, Yang Y, Zhang X, et al. From AI for science to agentic science: a survey on autonomous scientific discovery. arXiv:2508.14111. 2025. doi: [10.48550/arXiv.2508.14111](https://doi.org/10.48550/arXiv.2508.14111)
- [25] Demura M, Kishida K, Suga Y, et al. Fabrication of thin Ni3Al foils by cold rolling. *Scr Materialia*. 2002;47(4):267–272. doi: [10.1016/S1359-6462\(02\)00139-2](https://doi.org/10.1016/S1359-6462(02)00139-2)
- [26] Demura M, Suga Y, Umezawa O, et al. Fabrication of Ni3Al thin foil by cold-rolling. *Intermetallics*. 2001;9(2):157–167. doi: [10.1016/S0966-9795\(00\)00121-7](https://doi.org/10.1016/S0966-9795(00)00121-7)
- [27] Kishida K, Demura M, Suga Y, et al. Orientation dependence of texture evolution in cold-rolled Ni3Al single crystals. *Phil Mag*. 2003;83(26):3029–3046. doi: [10.1080/1478643031000149117](https://doi.org/10.1080/1478643031000149117)
- [28] Demura M, Hirano T. Stress response by the strain-rate change in a binary stoichiometric Ni3Al single crystal. *Phil Mag Lett*. 1997;75(3):143–148. doi: [10.1080/095008397179697](https://doi.org/10.1080/095008397179697)

Appendices

Appendix A. Crystal Plasticity Model Details

In this study, Ni₃Al simulations were performed using the phenomenological crystal plasticity model implemented in DAMASK. Plastic deformation is modeled by crystallographic slip. For Ni₃Al, slip was modeled using the 12 FCC {111}(110) systems. The shear rate on slip system α follows a rate-dependent power-law flow rule:

$$\dot{\gamma}^\alpha = \dot{\gamma}_0 \left| \frac{\tau^\alpha}{S^\alpha} \right|^n \text{sign}(\tau^\alpha) \quad (\text{A.1})$$

where $\dot{\gamma}_0$ is the reference shear rate, n is the stress exponent, τ^α is the resolved shear stress, and S^α represents the slip resistance, which is used to model strain hardening. S^α is assumed to increase from its initial value S_0 , with shear deformation. The rate of increase, \dot{S}^α , is described by the following equation:

$$\dot{S}^\alpha = h_0 \left(1 - \frac{S^\alpha}{S_\infty^\alpha} \right)^w \sum_\beta h_{\alpha\beta} |\dot{\gamma}^\beta| \quad (\text{A.2})$$

where S_∞^α is the saturation value, h_0 is the initial hardening modulus, w is a saturation exponent, $\dot{\gamma}^\beta$ is the slip rate on system β . $h_{\alpha\beta}$ is the strength of each slip-slip interaction and was set as 1 and 1.4 for coplanar and non-coplanar interactions, respectively.

Appendix B. Optimization Algorithms and Configuration

Overview of optimization strategy

CrystalPlasticitySim does not rely on a single fixed optimization algorithm. Instead, the system autonomously selects and configures optimization algorithms based on task characteristics such as parameter dimensionality, smoothness of the objective function, and physical bounds.

Case #1: Optimize material parameters

Optimization Algorithm

Material-parameter calibration was performed using the Differential Evolution (DE) algorithm implemented in SciPy. DE is a population-based global optimization method well suited for nonlinear, nonconvex objective functions.

Objective function

The objective function minimized was the mean absolute percentage error (MAPE) between simulated and experimentally measured true stress – true strain curves:

$$E = \text{MAPE}(\sigma^{exp}, \sigma^{sim}) = \frac{1}{N} \sum_{i=1}^N \left| \frac{\sigma_i^{sim} - \sigma_i^{exp}}{\sigma_i^{exp}} \right| \times 100 \quad (\text{B.1})$$

where σ_i^{sim} and σ_i^{exp} denote the simulated and experimental stresses at corresponding strain points. The

experimental stress – strain data were interpolated onto the simulated strain grid prior to evaluation.

Optimized parameters and bounds

The material parameters in the DAMASK material configuration file were optimized in the range shown in Table 4.

Convergence and stopping criteria

The Differential Evolution (DE) solver was configured using the best1bin strategy with a population size of 15, a maximum of 100 generations, and a convergence tolerance of 0.01. The optimization was terminated when either the relative improvement in the population objective value fell below the specified tolerance, the maximum number of generations was reached, or the solver's internal convergence criteria were satisfied.

Case #2

Optimization Algorithm

For orientation matching during cold rolling, the L-BFGS-B algorithm was employed. L-BFGS-B is a gradient-based quasi-Newton method designed for smooth, low-dimensional optimization problems with bound constraints.

Objective function

The objective function minimized was the deviation angle between the simulated and experimental crystal orientations. The simulated and experimental orientations were represented as unit quaternions q_{sim} and q_{exp} , which were converted to rotation matrices R_{sim} and R_{exp} , respectively. The deviation angle was then computed from the relative rotation matrix $R_\Delta = R_{sim} R_{exp}^{-1}$ as

$$\Delta\theta = \cos^{-1} \left(\frac{\text{tr}(R_{sim} R_{exp}^{-1}) - 1}{2} \right) \quad (\text{B.2})$$

Optimized parameters and bounds

The three independent shear components F_{12} , F_{13} , F_{23} of the deformation gradient were bounded within $[-0.003, 0.003]$

Convergence and stopping criteria

The L-BFGS-B optimization employed the default termination criteria provided by SciPy. Optimization was terminated when the projected gradient norm fell below the specified tolerance, when the change in the objective function value became sufficiently small, or when the maximum number of iterations was reached.