A Self-Correcting Multi-Agent Framework for Language-Based Physics Simulation and Explanation

Donggeun Park[a], Hyeonbin Moon[a], and Seunghwa Ryu[a]*

[a] Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, Republic of Korea

* Corresponding author: ryush@kaist.ac.kr

Abstract

Physics-based simulations are essential in science and engineering, yet creating them typically requires expert knowledge of numerical solvers and governing equations. Large language models (LLMs) offer new possibilities for natural language-based simulation, but they often fail when prompts are vague, incomplete, or multilingual. We present **MCP-SIM** (**M**emory-**C**oordinated **P**hysics-Aware **Sim**ulation), a self-correcting multi-agent framework that transforms underspecified prompts into validated simulations and explanatory reports. The system integrates input clarification, code generation, error diagnosis, and multilingual explanation through structured agent collaboration and persistent memory. Rather than relying on one-shot code generation, MCP-SIM emulates expert-like reasoning via iterative plan–act–reflect–revise cycles. Tested on a twelve-task benchmark across diverse physics domains, MCP-SIM achieved 100% success, significantly outperforming baseline LLMs. In addition to numerical accuracy, the system produces interpretable, language-localized reports that explain each simulation's physical logic. MCP-SIM represents a step toward general-purpose autonomous scientific assistants that simulate, adapt, and teach through natural language.

Introduction

Imagine describing a physical simulation in plain language, such as "*simulate how water flows around an obstacle shaped like a dolphin*," and receiving not only executable code and visualizations, but also a structured, multilingual explanation of the underlying physics. Such a capability would democratize access to computational modeling and open the door to simulation-driven discovery for students, engineers, and researchers without specialized programming or numerical expertise.

Despite recent advances in large language models (LLMs)[1-5], existing systems fall short when tasked with generating reliable simulations from ambiguous or linguistically diverse prompts[6,7,8]. Real-world simulation problems often lack complete specifications, including governing equations, boundary conditions, or material properties. Current approaches, such as one-shot code generators[9,10,11], autonomous simulation agents[12,13,14], and physics-informed neural networks[15,16,17,18], can provide partial solutions. Still, they typically assume well-posed input and offer little capacity for error recovery, refinement, or scientific explanation.

We hypothesize that this gap stems from a fundamental misalignment between one-shot LLM-based generation and the iterative, multi-step reasoning that expert modelers use in practice. To address this, we introduce **MCP-SIM**, short for **M**emory-**C**oordinated **P**hysics-Aware **Sim**ulation, a fully autonomous simulation framework that leverages a multi-agent architecture[19,20,21] to emulate expert-level simulation workflows. Rather than translating prompts into code in a single step, MCP-SIM performs structured cycles of planning, acting, reflecting, and revising through a network of specialized agents—each responsible for tasks such as prompt clarification, code generation, error diagnosis, and multilingual explanation.

Central to MCP-SIM is a shared memory system that enables agents to collaborate coherently and track the evolution of the simulation process. This architecture allows the system to infer missing assumptions, self-correct failed simulations, and generate scientifically grounded explanations—all without manual intervention. In benchmark evaluations across twelve increasingly complex tasks, MCP-SIM demonstrated perfect success rates and robust generalization across domains such as elasticity, heat transfer, multiphysics and fracture mechanics.

Beyond its technical performance, MCP-SIM redefines how simulations can be approached and understood. It serves not only as a solver but also as an autonomous scientific assistant, capable of teaching users the rationale behind its modeling choices through multilingual, interpretable reports. By bridging natural language and computational modeling, MCP-SIM expands the frontier of accessible scientific computing and offers a blueprint for future AI systems that both simulate and explain.

Results

A Physics-Aware Multi-Agent Framework for Simulation Automation

We present MCP-SIM, a memory-coordinated multi-agent framework that collaboratively interprets, constructs, and executes finite element simulations directly from natural language prompts. Designed to support structured reasoning workflows typical of expert-level physics-based modeling, MCP-SIM integrates six specialized agents—the *Input Clarifier Agent*, *Code Builder Agent*, *Simulation Executor Agent*, *Error Diagnosis Agent*, *Input Rewriter Agent*, and *Mechanical Insight Agent*—all orchestrated by a central controller, the *Memory-Centric Orchestrator*, which governs information flow via a persistent shared memory (**Figure 1**).

At the core of MCP-SIM lies a Plan → Act → Reflect → Revise control loop that enables self-correcting and interpretable simulation synthesis. Upon receiving a vague or incomplete user prompt—such as "simulate fluid flow in an L-shaped pipe" (①)—the *Input Clarifier Agent* (②) infers essential simulation details like domain geometry, governing PDE (e.g., Navier–Stokes), and boundary conditions using domain knowledge and language understanding. These inferred attributes are stored in global memory as the canonical problem specification.

Next, the *Code Builder Agent* (③) translates this clarified input into solver-ready Python code via prompt templates infused with physics-aware heuristics for mesh generation, solver configuration, and boundary condition specification, using FEniCS[22] as the simulation backend. These templates embed domain-specific heuristics that guide modeling decisions—such as mesh refinement, solver configuration, and stability control—based on the physical

and numerical context of the problem. All assumptions, numerical strategies, and intermediate artifacts are persistently logged, ensuring reproducibility and traceability.

The resulting code is executed by the *Simulation Executor Agent* (③–④) within a sandboxed environment. In addition to standard execution, the agent monitors physically meaningful indicators—such as conservation violations, residual divergence, or spurious field oscillations—that may signal modeling or discretization issues. If runtime failures occur—such as solver divergence, syntax issues, or physical inconsistencies—the system enters the **Reflect** phase, where the *Error Diagnosis Agent* (④) is invoked. The *Error Diagnosis Agent* (④) interprets failures in physical terms, such as insufficient mesh resolution or solver-mismatch. It proposes targeted, physics-aware corrections—such as adjusting mesh density, reducing the time step, or modifying solver parameters—based on all execution logs and memory history. These corrections are communicated to other agents as structured hints (④-i) for subsequent revision. If the issue stems from high-level ambiguity in the user prompt, the *Input Rewriter Agent* (⑤−i) semantically revises the instruction (e.g., adding "mesh resolution = 64" or "relaxation = 0.7"), and the control loop restarts from clarification (②)

Upon successful simulation, the *Mechanical Insight Agent* (⑥) automatically generates an interpretive report. This includes symbolic PDE summaries, physical reasoning behind the model, and code-level annotations—all rendered in accessible formats and multiple languages to support education and transparency (⑥−i). Critically, the *Memory-Centric Orchestrator* acts as the central coordination substrate, capturing the full trajectory of the simulation: from prompt history and problem clarifications to code iterations, diagnostic logs, and final results. This persistent trace enables context-aware decision making across agents and prevents redundant computation or regressions.

In contrast to reactive prompt-to-code systems, MCP-SIM embodies memory-centric simulation automation with interpretable physical reasoning, capable of resolving complex physics problems through modular reasoning, targeted revisions, and cumulative reflection. Prompt templates for each agent—such as PDE clarification, code generation, and diagnostic hinting—were modularly designed based on domain heuristics. Their constructions are described in **Supplementary Note 1** and **Supplementary Figure 1-5**.

Benchmark Performance: Accuracy and Efficiency

To assess the robustness of MCP-SIM, we evaluated the system on a twelve-task benchmark suite encompassing a range of physics domains—including linear elasticity, heat conduction, fluid flow, thermo-mechanical coupling, piezoelectric deformation, and phase-field fracture mechanics. These tasks were intentionally designed to reflect real-world challenges faced by students and non-experts, such as vague boundary conditions, missing material properties, and ambiguous geometry descriptions (**Figure 2a**; **Table 1**). Based on prompt completeness and modeling complexity, we categorized the 12 tasks into:

- **Simple (Levels 1–4)**: Fully specified, single-physics problems.
- **Intermediate (Levels 5–9)**: Incomplete inputs requiring inference (e.g., geometry, material, BCs).
- **Challenging (Levels 10–12)**: Multi-physics with missing assumptions, inspired by published problems without code.

To quantify how each component of MCP-SIM contributes to final performance, we conducted an ablation study across the benchmark. A baseline GPT-4 model using one-shot prompt-to-code generation solved only 6 out of 12 problems, often failing to infer missing problem details or produce physically consistent outputs. Incorporating the *Input Clarifier Agent*, which explicitly infers geometry, PDEs, and boundary conditions, increased the success

rate to 8 out of 12. Adding the *Error Diagnosis Agent*, which analyzes and corrects solver failures, further improved performance to 10 out of 12. In contrast, the full MCP-SIM system—with shared memory and orchestrator-driven agent collaboration—achieved a 100% success rate, successfully solving all twelve tasks by combining clarification, repair, and reflection in a structured control loop (**Figure 2b**).

In addition, we evaluated convergence efficiency by measuring the number of Plan–Act–Reflect–Revise cycles required for each task. As shown in **Figure 2c**, MCP-SIM typically converged in fewer than five iterations, even for the most complex problems. This is a substantial improvement over the baseline, which often failed to converge or required more than a dozen iterations. These gains are attributed to structured memory, agent-level specialization, and iterative self-healing.

Building on these performance metrics, **Figure 3** visualizes the final outputs for the 12 benchmark tasks—demonstrating MCP-SIM's ability to generate physically meaningful results, even in the presence of ambiguity or missing information. The prompts were deliberately designed to span a wide range of physical domains—including elasticity, fluid dynamics, heat transfer, and coupled-field phenomena—and to vary in geometric and numerical complexity. This progression ensures that MCP-SIM is evaluated not only on solvable cases but also on how it responds to the kinds of ambiguity common in real-world practice.

Despite variations in governing equations and modeling complexity, MCP-SIM consistently produced executable and physically plausible simulations without human intervention. Through shared memory, the *Input Clarifier* and *Error Diagnosis agent* dynamically inferred missing quantities such as inlet velocities, thermal conductivities, and

solver parameters. When failures occurred, the system recovered autonomously using physics-aware refinement strategies, guided by the Plan → Act → Reflect → Revise loop.

Each simulation met physical convergence criteria, with residuals below 10^{-6}. Importantly, the solutions were not only numerically stable but also physically interpretable. For instance, in Level 1 (linear elasticity), the stress and displacement fields reflect equilibrium under tension around a central hole. In Level 9 (3D heat diffusion), the temperature gradients follow expected patterns of thermal conduction in a coil-shaped body—mirroring practical thermal management setups.

These outcomes confirm that MCP-SIM is not narrowly tuned to templates but robustly generalizes across domains. Even when given vague or under-specified prompts, the system adapts and delivers physically meaningful results. This capability makes it a reliable tool not only for automated simulation but also for educational and engineering applications where interpretability, flexibility, and correctness are essential.

Expert-Level Reasoning from Minimal Prompts

To illustrate how MCP-SIM handles complex physical systems from minimal input, we examine its performance on Level 12—the most challenging task in the benchmark suite. This case involves reproducing a nonlinear phase-field fracture simulation[23] from a single prompt—"*simulate crack propagation in a square domain with a central circle.*"—without numerical values, PDE specifications, or boundary conditions.

MCP-SIM successfully inferred that a phase-field fracture model was appropriate. The *Input Clarifier* proposed a 2D domain with a central void, applied Dirichlet conditions at the base, and assigned traction-free boundaries elsewhere. It also selected appropriate material constants and solver parameters. The *Code Builder Agent* generated corresponding FEniCS code, and upon encountering a syntax error, the *Error Diagnosis Agent* corrected the variational form and mesh resolution.

After ten self-repair cycles, the simulation converged to physically plausible results (**Figure 4b**), matching expected crack path evolution and stress concentration patterns observed in expert-authored studies (**Figure 4a**). This case illustrates MCP-SIM's capacity for domain-aware abstraction, autonomous decision-making, and model justification, starting from minimal human input. Full simulation codes automatically generated by MCP-SIM for all challenging prompts—Level 10 (thermoelectric coupling), Level 11 (piezoelectric deformation), and Level 12 (phase-field fracture)—are provided in **Supplementary Figures 6–8**, demonstrating the system's ability to generalize across diverse and complex physics domains.

Educational Insight and Multilingual Explanation

Beyond execution, MCP-SIM enhances interpretability and promotes simulation literacy by automatically generating structured simulation reports. After each successful run, the *Mechanical Insight Agent* produces multilingual explanations that describe the simulation's physical principles, governing equations, boundary conditions, and solver strategies.

These reports are available in English and Korean (**Figure 5**), as well as Japanese and German (**Supplementary Figure 9**), and follow a pedagogically consistent structure. They combine code-level annotations with high-level rationales—for example, explaining why Dirichlet rather than Neumann boundaries were applied, or why certain solvers were selected for stability.

Importantly, these reports go beyond explanation—they scaffold student learning. They clearly define simulation goals, unpack physical models and assumptions, and guide users through critical implementation choices. By including localized language support and conceptual reasoning, MCP-SIM acts as a virtual teaching assistant—helping learners not just run simulations, but understand them. This reduces barriers to simulation-based education and broadens access across languages, disciplines, and levels of expertise[24].

Discussion

The results presented above highlight MCP-SIM as a step change in autonomous simulation, demonstrating how LLMs, when structured through multi-agent collaboration and persistent memory, can move beyond static code generation toward self-correcting, interpretable scientific computation. Unlike prior systems that rely on single-step prompt-to-code translation, MCP-SIM emulates the iterative reasoning loop of human experts—planning, acting, reflecting, and revising—through modular agents that perform distinct tasks such as input clarification, code repair, and physical explanation. This architecture allows the system to recover from incomplete or ambiguous inputs and deliver physically plausible outputs across diverse domains including elasticity, heat conduction, and fracture mechanics.

Importantly, MCP-SIM is not a black-box tool. It generates multilingual educational reports that explain the governing equations, boundary conditions, solver strategies, and assumptions underlying each simulation. This interpretability distinguishes it from conventional LLM-based tools, positioning it as an autonomous scientific assistant that can both compute and teach. The ability to explain results in multiple languages also extends accessibility to a broader community of students, educators, and practitioners. The generalization observed across all twelve benchmark tasks suggests that MCP-SIM captures not just surface-level patterns, but domain-aware abstractions that transfer across physical systems and modeling contexts. This capacity for abstraction is particularly evident in Level 12, where the system reproduced a published fracture simulation from a single-sentence prompt without explicit geometry or equations.

Despite these strengths, several limitations remain. MCP-SIM is built on a general-purpose LLM (GPT-4o), which is not fine-tuned for specific physical domains. As a result,

performance may degrade in specialized settings involving rare material models, advanced PDE solvers, or multiphysics coupling beyond the training distribution. In addition, the iterative reasoning loop—while essential for robustness—introduces computational latency that may limit deployment in real-time or resource-constrained scenarios. Future work will focus on integrating domain-specific foundation agents, incorporating symbolic physics engines, and optimizing inference for high-performance computing environments. Expanding the benchmark suite to include real-world data, sensor input, and experimental validation will also be crucial for establishing trust in high-stakes applications. We also envision integrating MCP-SIM into collaborative platforms where human users and AI co-design models, exchange reasoning steps, and accelerate discovery.

At a broader level, MCP-SIM signals a shift in the role of LLMs in scientific computing—from passive generators of syntax to active participants in modeling, interpretation, and explanation. By embedding physical reasoning, structured iteration, and educational transparency into its core, MCP-SIM lays the foundation for a new class of tools that make simulation not only more powerful but also more accessible, adaptive, and scientifically grounded.

Conclusion

We have introduced MCP-SIM, a fully autonomous, multi-agent framework for interpreting natural language prompts and generating physically consistent, validated simulations along with multilingual explanatory reports. By integrating task-specialized agents and structured memory within a reflective reasoning loop, MCP-SIM mimics expert modeling workflows—clarifying incomplete inputs, self-correcting failed simulations, and articulating the physical logic behind modeling decisions.

Through comprehensive benchmarking across twelve tasks of increasing complexity, the system demonstrated strong generalization across physical domains and outperformed baseline LLM-based methods in accuracy, convergence speed, and interpretability. Its ability to explain simulations in multiple languages further expands its role from solver to educator, bridging gaps between disciplines, languages, and expertise levels.

MCP-SIM exemplifies a new paradigm for AI-driven scientific modeling—one that blends computational accuracy with transparency and accessibility. As autonomous agents become increasingly integrated into scientific workflows, systems like MCP-SIM will serve as foundational infrastructure for simulation-based discovery, design, and learning.

Methods

Agent Framework Design

The MCP-SIM system is built atop GPT-4o (OpenAI, 2024 release) and leverages the LangChain framework for orchestrating multi-agent workflows. Each specialized agent—such as the *Input Clarifier Agent*, *Code Builder Agent*, *Simulation Executor Agent*, *Error Diagnosis Agent*, and *Mechanical Insight Agent* —is implemented as an independent callable chain within LangChain's abstraction layer. Agents communicate exclusively through a shared global memory, structured as a JSON-based context log that records all inputs, clarifications, generated code snippets, execution outputs, and error diagnoses at each step. The orchestrator module centrally manages agent invocation according to predefined rules: for example, syntax errors trigger *Error Diagnosis Agent* first, while physical anomalies trigger *Mechanical Insight Agent* analysis. Importantly, agents do not call each other directly; all task planning and transitions are mediated by the orchestrator, ensuring reproducibility and traceability across complex Plan → Act → Reflect → Fix cycles. All models are deployed without task-specific fine-tuning, relying purely on prompt engineering and tool orchestration to achieve autonomous simulation.

Simulation Backend and Libraries

In MCP-SIM, simulation agent is based on the open-source FEniCS computing platform, a widely used library for solving PDEs via finite element methods. Python 3.9 was used as the scripting environment, and all generated codes employed standard FEniCS constructs for mesh generation, function space definition, weak form assembly, and nonlinear solver execution. Simulations were run locally in sandboxed Python environments without manual modification unless otherwise noted.

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

Data availability

Data will be made available on request.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (RS-2023-00222166) and by a grant from the Ministry of Food and Drug Safety (RS-2023-00215667). We also acknowledge the support of the Korean Society of Mechanical Engineers (KSME).

Reference

1. Li, Y., Choi, D., Chung, J., et al. (2022). *Competition-level code generation with AlphaCode.* Science, 378(6624), 1092–1097. DOI: 10.1126/science.abq1158.
2. Bakhtin, A., Brown, N., Dinan, E., et al. (2022). *Human-level play in the game of Diplomacy by combining language models with strategic reasoning.* Science, 378(6624), eade9097. DOI: 10.1126/science.ade9097.
3. Romera-Paredes, B., Barekatain, M., Novikov, A., et al. (2024). *Mathematical discoveries from program search with large language models.* Nature, 625, 468–475. DOI: 10.1038/s41586-023-06924-6.
4. Vaswani, A., Shazeer, N., Parmar, N., *et al.* (2017). *Attention is all you need.* In Advances in Neural Information Processing Systems, 30, 5998–6008.
5. Brown, T. B., Mann, B., Ryder, N., *et al.* (2020). *Language models are few-shot learners.* In Advances in Neural Information Processing Systems, 33, 1877–1901.
6. Steyvers, M., Tejeda, H., Kumar, A., *et al.* (2025). *What large language models know and what people think they know.* Nature Machine Intelligence, 7, 221–231. DOI: 10.1038/s42256-024-00976-7.
7. Musslick, S., Bartlett, L. K., Chandramouli, S. H., *et al.* (2025). *Automating the practice of science: Opportunities, challenges, and implications.* Proceedings of the

National Academy of Sciences, 122(5), e2401238121. DOI: 10.1073/pnas.2401238121.
8. Krenn, M., Pollice, R., Guo, S. Y., *et al.* (2022). *On scientific understanding with artificial intelligence.* Nature Reviews Physics, 4(10), 761–769. DOI: 10.1038/s42254-022-00518-3
9. Luo, X., Rechardt, A., Sun, G., et al. (2025). *Large language models surpass human experts in predicting neuroscience results.* Nature Human Behaviour, 9, 305–315. DOI: 10.1038/s41562-024-02046-9.
10. Chen, L., Rao, Z., & Singh, S. (2024). Self-planning code generation with large language models: plan-then-implement framework. *ACM Transactions on Software Engineering and Methodology*, 33(3), 21:1–21:25. DOI: 10.1145/3672456
11. Le, K. T. & Andrzejak, A. (2024). *Rethinking AI code generation: a one-shot correction approach based on user feedback. Automated Software Engineering*, 31, 60. DOI: 10.1007/s10515-024-00451-y
12. Dong, Z., Lu, Z., & Yang, Y. (2025). *Fine-tuning a large language model for automating computational fluid dynamics simulations.* Theoretical & Applied Mechanics Letters, 15, 100594. DOI: 10.1016/j.taml.2025.100594.
13. Kim, S., Yu, Y., & Seo, H. (2025). *Artificial intelligence orchestration for text-based ultrasonic simulation via self-review by multi-large language model agents*. Scientific Reports, 15, 12474. DOI: 10.1038/s41598-025-97498-y
14. Ni, B. & Buehler, M. J. (2024). *MechAgents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge*. Extreme Mechanics Letters, *67*, 102131. DOI: 10.1016/j.eml.2024.102131
15. Karniadakis, G. E., Kevrekidis, I. G., Lu, L., *et al.* (2021). *Physics-informed machine learning.* Nature Reviews Physics, 3(6), 422–440. DOI: 10.1038/s42254-021-00314-5.
16. Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*. Journal of Computational Physics, 378, 686–707. DOI: 10.1016/j.jcp.2018.10.045
17. Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2022). *Scientific machine learning through physics-informed neural networks: trends and frontiers*. Journal of Scientific Computing, 92, 57. DOI: 10.1007/s10915-022-01939-z
18. Zhang, C. & de Lillo, F. (2022). *A physics-informed neural framework for inversion and surrogate modeling in solid mechanics*. Computer Methods in Applied Mechanics and Engineering, 379, 113741. DOI: 10.1016/j.cma.2021.113741
19. Vinyals, O., Babuschkin, I., Czarnecki, W. M., et al. (2019). *Grandmaster level in StarCraft II using multi-agent reinforcement learning.* Nature, 575(7782), 350–354. DOI: 10.1038/s41586-019-1724-z.
20. Bae, H. J., & Koumoutsakos, P. (2022). *Scientific multi-agent reinforcement learning for wall-models of turbulent flows.* Nature Communications, 13, 1443. DOI: 10.1038/s41467-022-28957-7.
21. Ghafarollahi, A., & Buehler, M. J. (2024). *SciAgents: Automating scientific discovery through bioinspired multi-agent intelligent graph reasoning.* Advanced Materials, e2413523. DOI: 10.1002/adma.202413523.
22. Alnæs, M. S., Blechta, J., Hake, J., *et al.* (2015). *The FEniCS Project Version 1.5.* Archive of Numerical Software, 3(100), 9–23. DOI: 10.11588/ans.2015.100.20553.

23. Storvik, E., Both, J., *et al.* (2021). *An accelerated staggered scheme for variational phase-field models of brittle fracture*. Computer Methods in Applied Mechanics and Engineering, 381, 113822. DOI: 10.1016/j.cma.2021.113822.
24. Qin, L., Chen, Q., Zhou, Y., *et al.* (2025). *A survey of multilingual large language models.* Patterns, 6(1), 101118. DOI: 10.1016/j.patter.2024.101118.

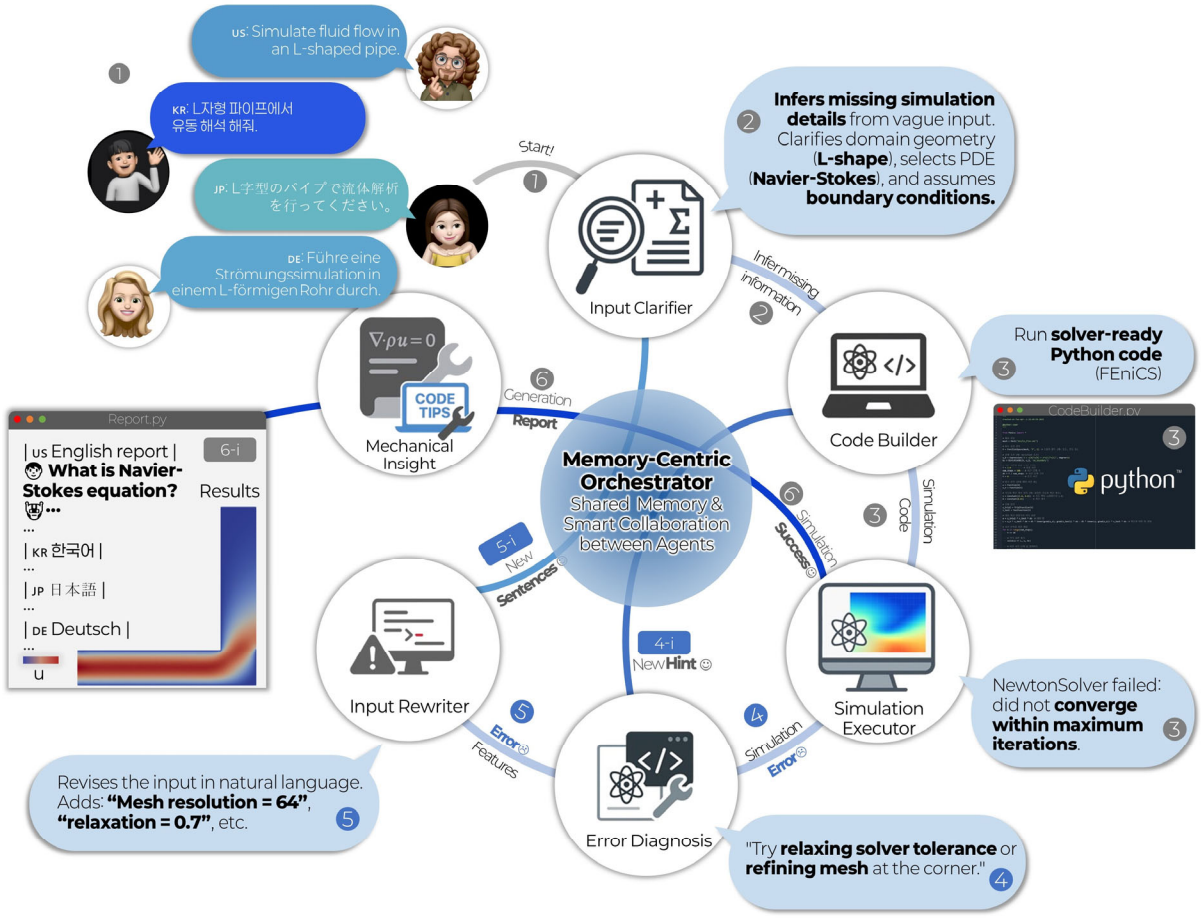


Figure 1 MCP-SIM workflow and agent collaboration. The system starts with a user's natural language prompt, which is clarified by the *Input Clarifier Agent* to identify missing details (geometry, PDE type, boundary conditions). The *Code Builder Agent* then generates solver-ready Python code. If errors occur, the *Error Diagnosis Agent* identifies and corrects them. Finally, the *Mechanical Insight Agent* produces a multilingual report explaining the simulation's physical principles and results, showcasing how MCP-SIM ensures accurate, executable, and educational outputs through agent collaboration.

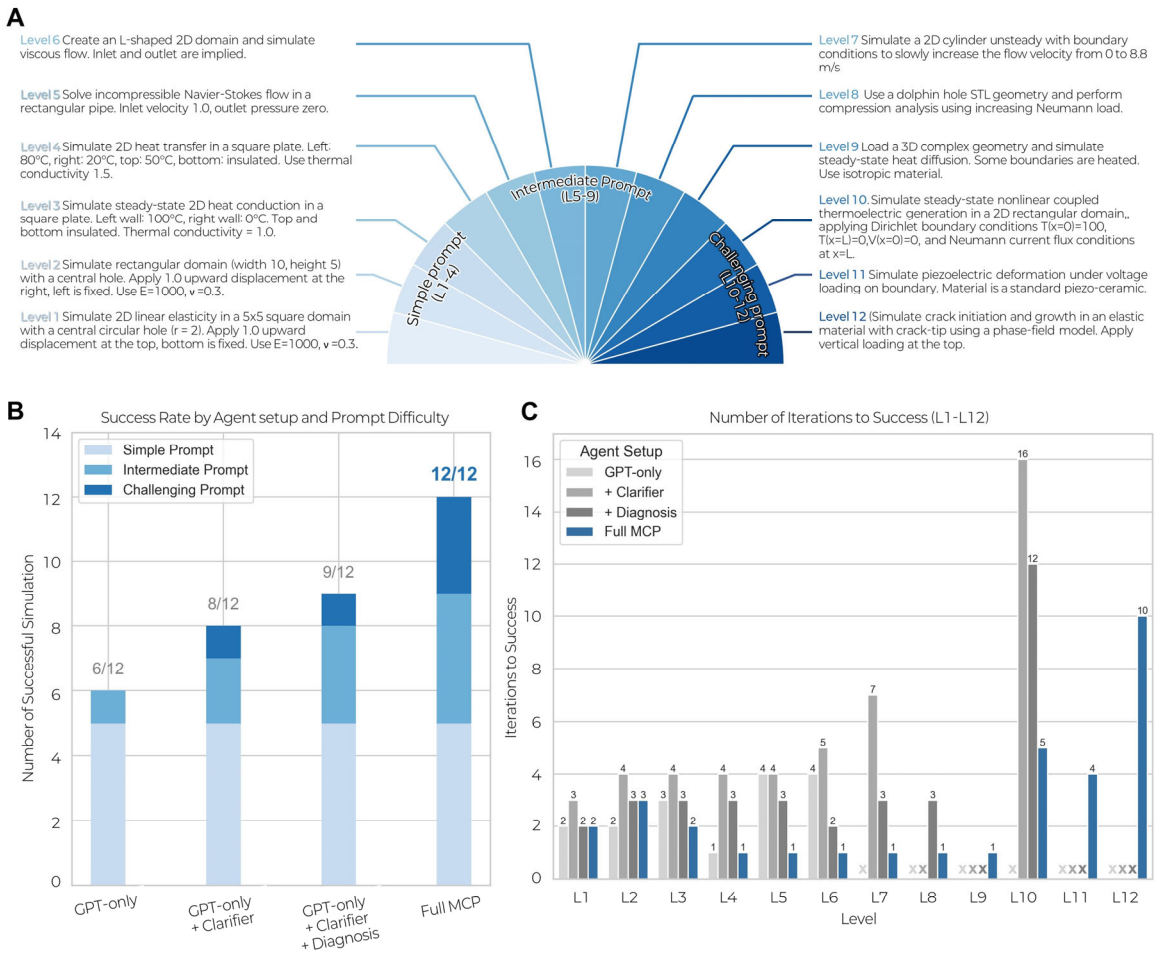


Figure 2 Benchmark evaluation of MCP-SIM across varying simulation complexities. (**A**) Twelve benchmark tasks categorized by increasing difficulty—from well-posed, single-physics problems (Levels 1–4) to abstract, multi-physics challenges (Levels 10–12). (**B**) Success rates across different agent configurations, comparing baseline GPT-4 setups with MCP-SIM. (**C**) Number of Plan–Act–Reflect–Revise cycles required for successful outcomes. MCP-SIM consistently solves all tasks within five iterations, significantly outperforming baseline models.

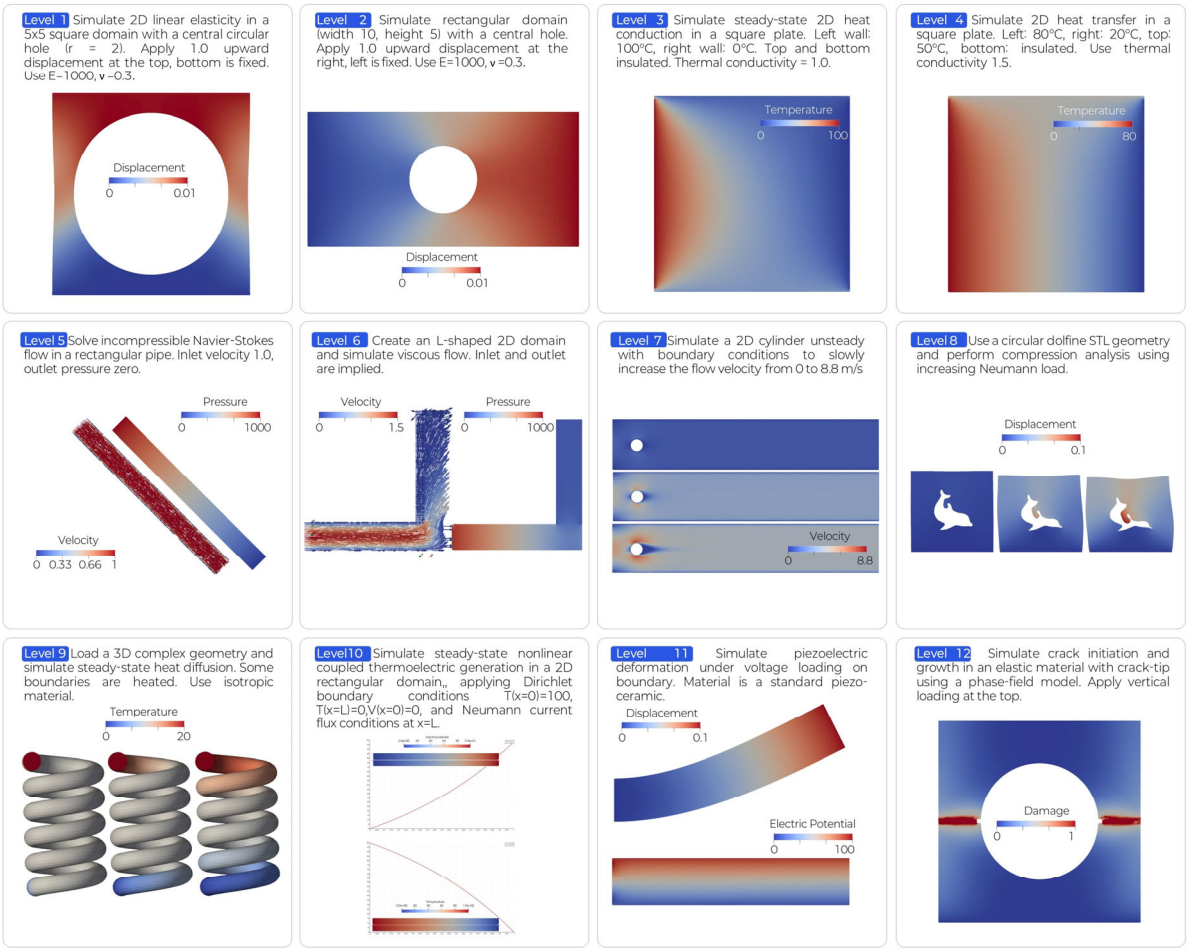


Figure 3 Simulation results for the 12 benchmark problems used to validate MCP-SIM across different physics domains. (Top-left to bottom-right): The figure presents the final simulation outputs from MCP-SIM for each of the 12 levels of complexity, ranging from basic linear elasticity (Level 1) to advanced multi-physics problems such as phase-field crack growth (Level 12).

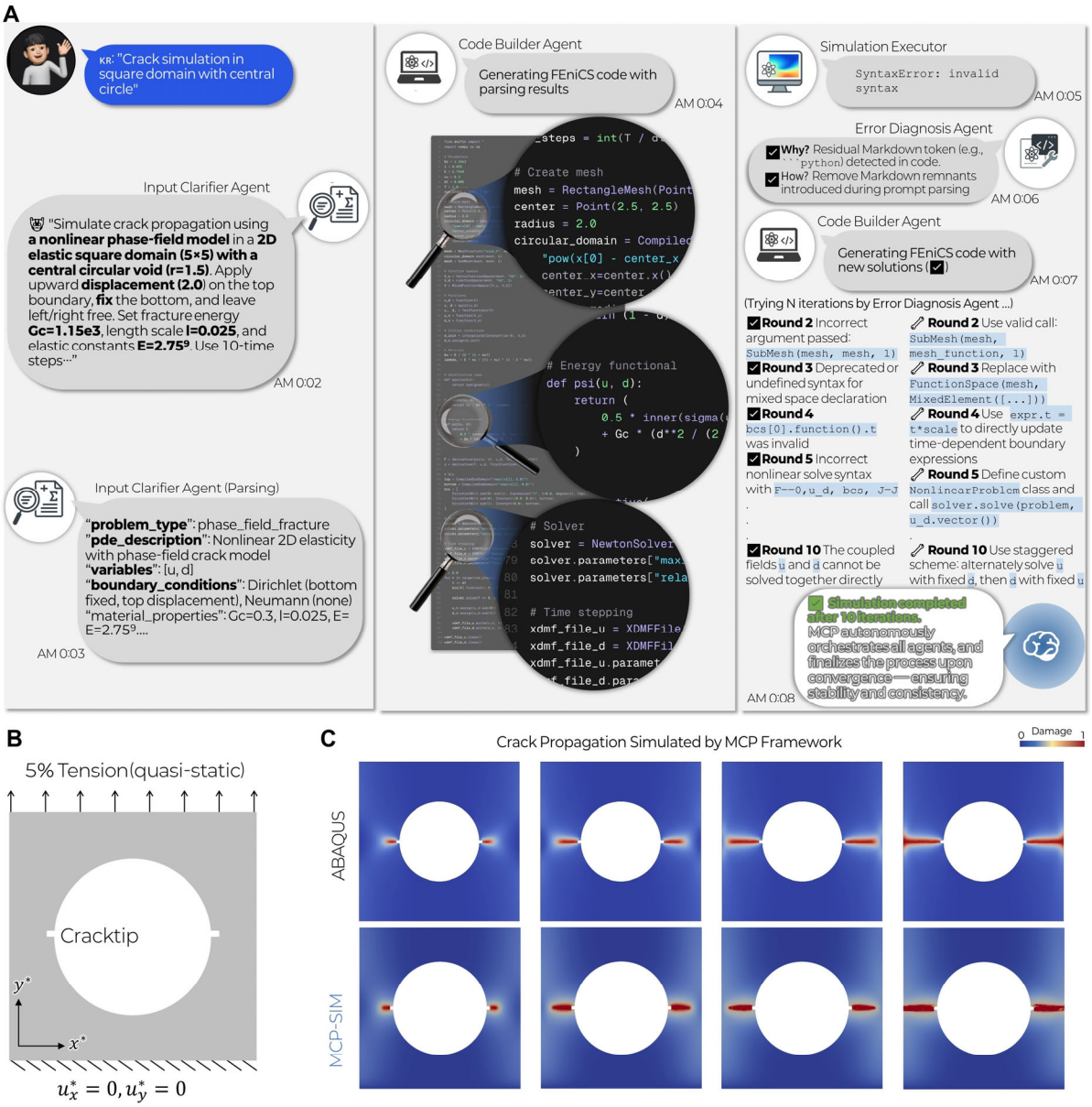


Figure 4 Autonomous reproduction of expert-designed fracture simulations from minimal prompts. (**A**) Agent reasoning process for Level 12: From a single-sentence prompt, MCP-SIM infers geometry, governing equations, and solver setup through iterative agent collaboration. (**B**) Problem setup: a square domain with a central circular crack tip subjected to 5% upward displacement on the top boundary; bottom boundary is fixed, representing a quasi-static tension condition. (**C**) Comparison of crack propagation results obtained using ABAQUS (top row) and the MCP-SIM framework (bottom row). Both methods predict symmetric crack growth from the circular notch, validating the correctness and robustness of the autonomous simulation workflow.

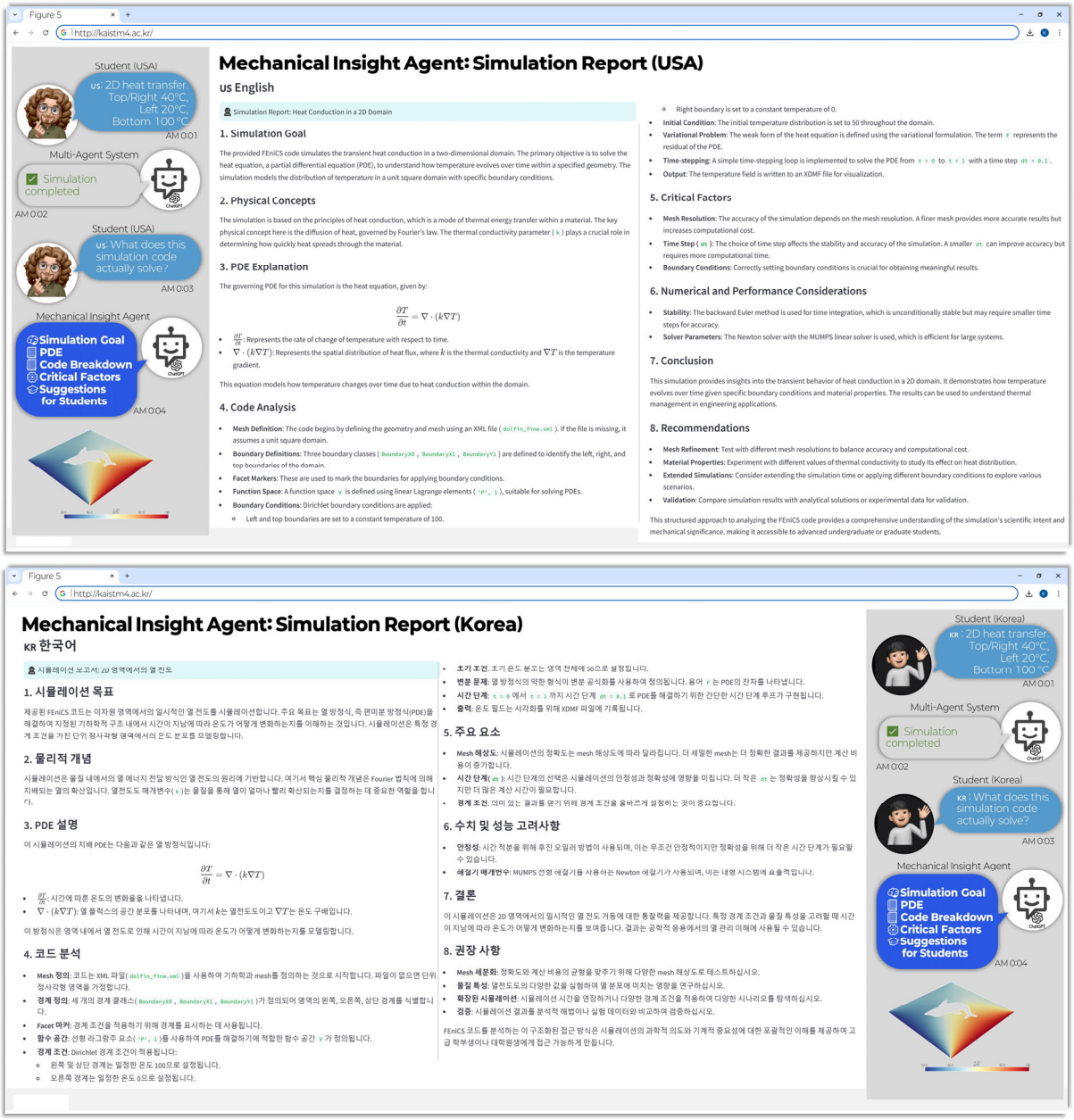


Figure 5 Mechanical insight agent: simulation reports - English and Korean. This figure showcases simulation reports generated by the Mechanical Insight Agent in English (USA) and Korean. The reports provide comprehensive, student-friendly explanations of the simulation, including details on the simulation goal, physical concepts, PDE explanations, and critical factors. The multilingual capability ensures that students from different backgrounds can easily understand the process and results of their simulations.

Level	Problem Type	Challenge	Key Parameters/Omissions	Objective
1	2D Linear Elasticity	Simple case, fully specified	Geometry (5x5 square domain, central circular hole, displacement)	Test basic simulation handling with well-defined inputs
2	2D Linear Elasticity	Slightly more complex geometry and boundary conditions	Central hole geometry, displacement boundaries, material properties	Handle geometric complexity with standard boundary conditions
3	Steady-State 2D Heat Conduction	Thermal conduction simulation with fixed boundary conditions	Temperature boundaries (100°C, 0°C), thermal conductivity	Simulate steady-state heat conduction with fully specified data
4	2D Heat Transfer	Complex temperature boundary conditions	Thermal conductivity (1.5), boundary temperatures	Test ability to handle thermal problems with different boundary conditions
5	Incompressible Navier-Stokes Flow	Simulate viscous flow with inlet and outlet conditions	Inlet velocity, outlet pressure, boundary conditions	Handle fluid dynamics with given velocity and pressure conditions
6	Viscous Flow in L-shaped domain	Inlet and outlet are implied without explicit boundary conditions	Flow conditions not explicitly stated	Handle undefined boundary conditions in L-shaped domain
7	Unsteady Flow in 2D Cylinder	Simulate unsteady fluid flow with varying velocity	Velocity increase over time, boundary conditions	Test fluid dynamics under time-dependent flow conditions
8	2D Compression Analysis	STL geometry for compression under Neumann boundary conditions	STL geometry, applied load	Handle geometric input (STL) for compression analysis
9	3D Heat Diffusion	Complex geometry with heat diffusion	Heated boundaries, isotropic material	Test 3D heat diffusion with complex geometries and varying boundary conditions
10	Steady-State Thermoelectric Generation	Solve coupled temperature and electric potential fields	Multi-physics simulation	Simulate thermoelectric generation with coupled fields
11	Piezoelectric Deformation	Deformation due to voltage loading	Boundary voltage, material properties (piezo-ceramic)	Test piezoelectric deformation with voltage-loaded boundaries
12	Phase-Field Crack Growth	Simulate crack initiation and growth in elastic material	Vertical loading at the top, crack-tip model	Handle complex fracture growth and material behavior in simulations

Table 1 Overview of MCP-SIM test levels and objectives. This table presents the twelve test levels used to evaluate MCP-SIM's performance across a range of simulation challenges. Each level addresses specific problem types, from simple linear elasticity to complex multi-physics scenarios. The table outlines the key challenges, parameters or omissions, and objectives that each level aims to test, demonstrating the system's ability to handle both straightforward and ambiguous inputs in various engineering domains.

Supplementary Information

A Self-Correcting Multi-Agent Framework for Language-Based Physics Simulation and Explanation

Donggeun Park[a], Hyunbin Moon[a], and Seunghwa Ryu[a]*

Affiliations

[1]Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea

[2]Department of Mechanical Engineering, University of California, Berkeley, CA 94720, USA

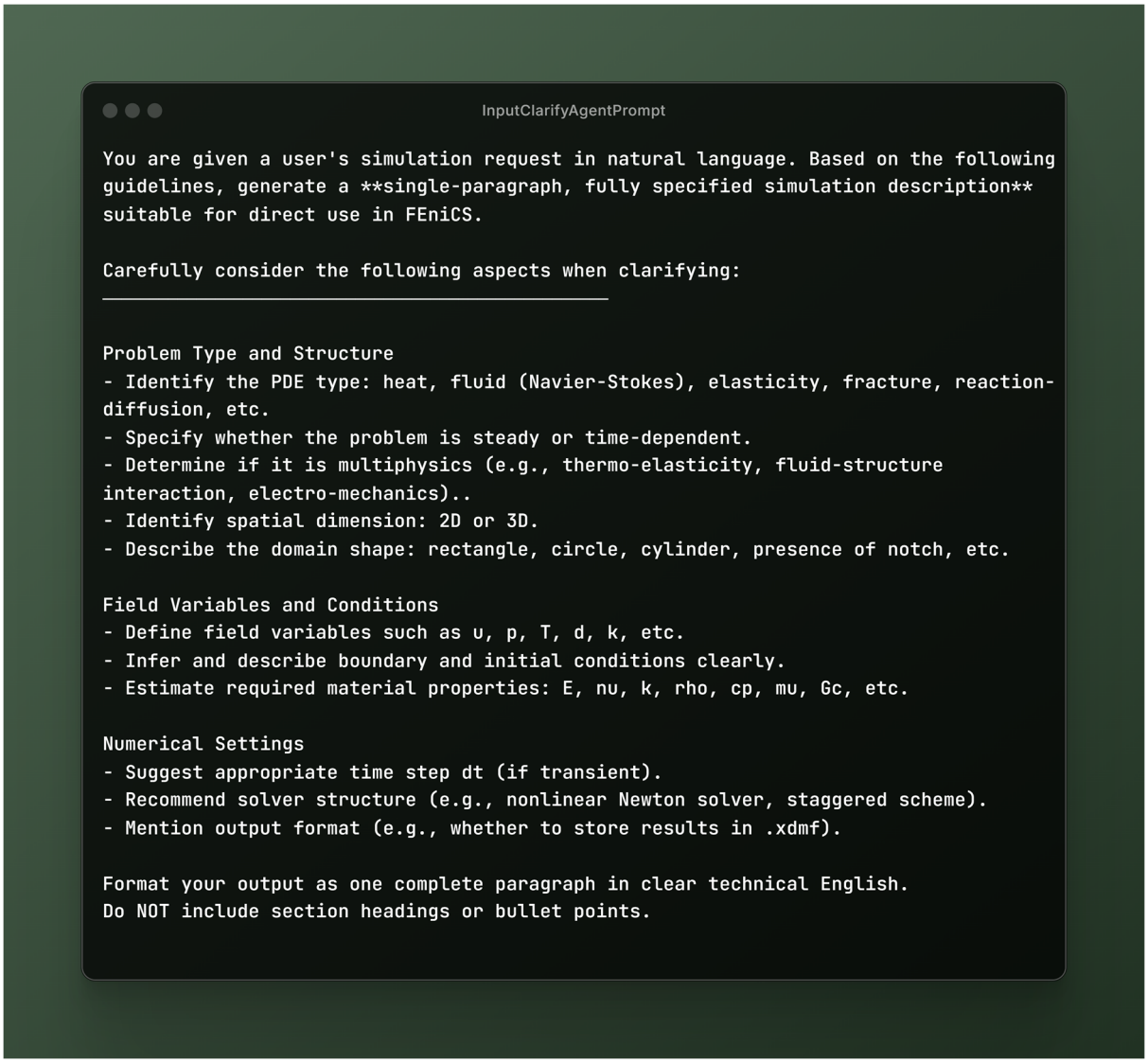
*Corresponding author: ryush@kaist.ac.kr

Supplementary Note 1. Prompt Template Design

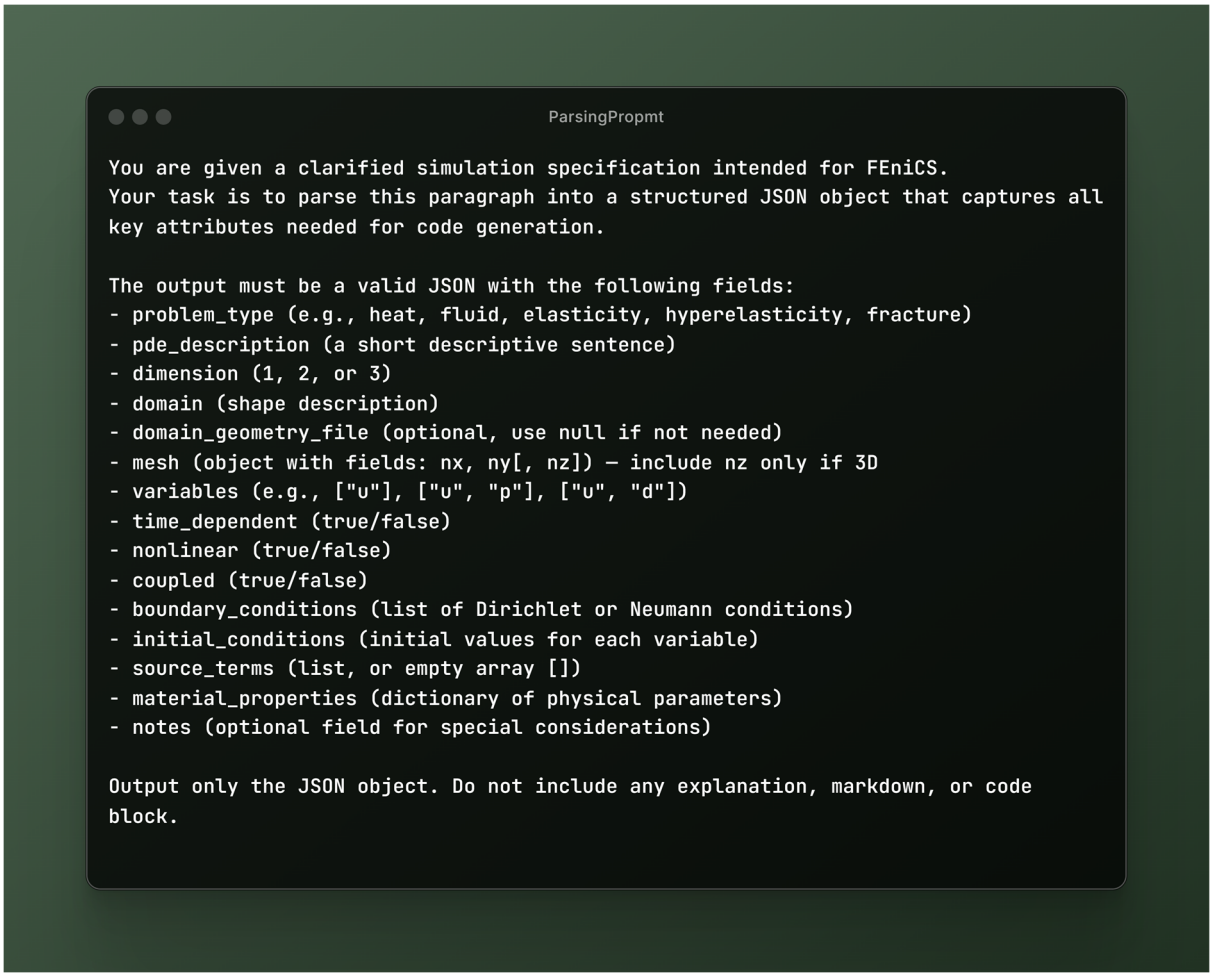
Our prompt templates build upon the structure of standard FEniCS tutorials[1], but extend their capability through a systematic design process that combines modularity and expert knowledge. We began by decomposing existing tutorial examples into reusable functional blocks—such as geometry inference, PDE selection, boundary condition specification, and numerical settings.

To embed domain expertise, we interviewed three computational mechanics specialists and compiled over thirty physics-aware heuristics, including CFL stability limits, mesh refinement ratios, and solver tuning strategies etc. These heuristics were encoded into a dynamic ruleset, which populates specific sections of the prompt templates using context-sensitive calls to GPT-4o.

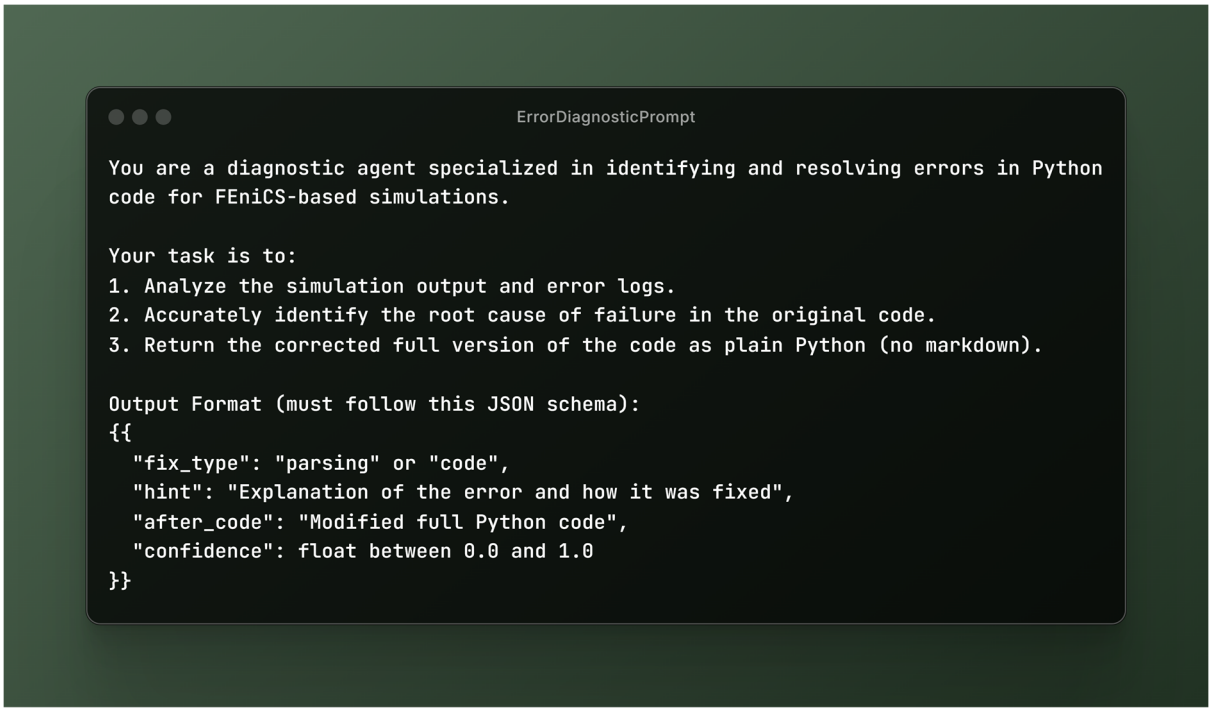
This design ensures that the resulting prompt templates are not only physically grounded and adaptable to a wide range of modeling tasks, but also maintainable, verifiable, and aligned with real-world simulation practices.



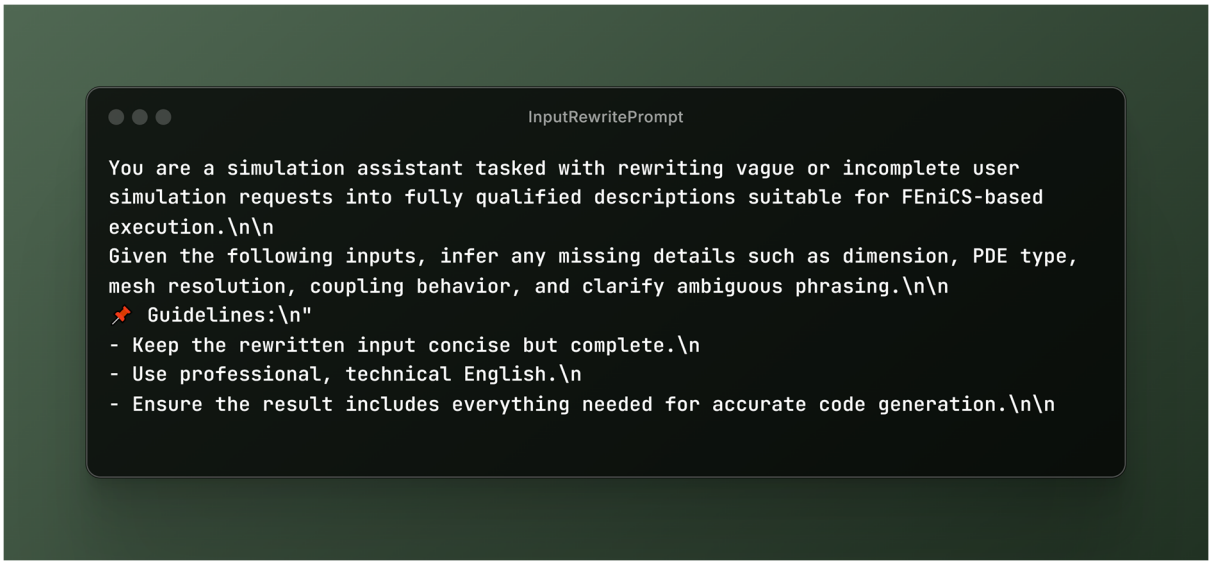
Supplementary Figure 1. Input clarification prompt template used by the Input Clarifier Agent. This prompt guides the Input Clarifier Agent in transforming a vague or underspecified natural language simulation request into a fully specified description suitable for FEniCS-based execution. It outlines a set of physics-aware heuristics to infer PDE types, domain geometry, time-dependency, field variables, boundary conditions, material properties, and solver configurations. The agent is instructed to return a single coherent paragraph in clear technical English, enabling downstream agents to operate on a standardized, well-formed simulation specification. This clarification step plays a critical role in MCP-SIM's Plan → Act → Reflect → Revise loop by providing physically grounded initial conditions from incomplete input.



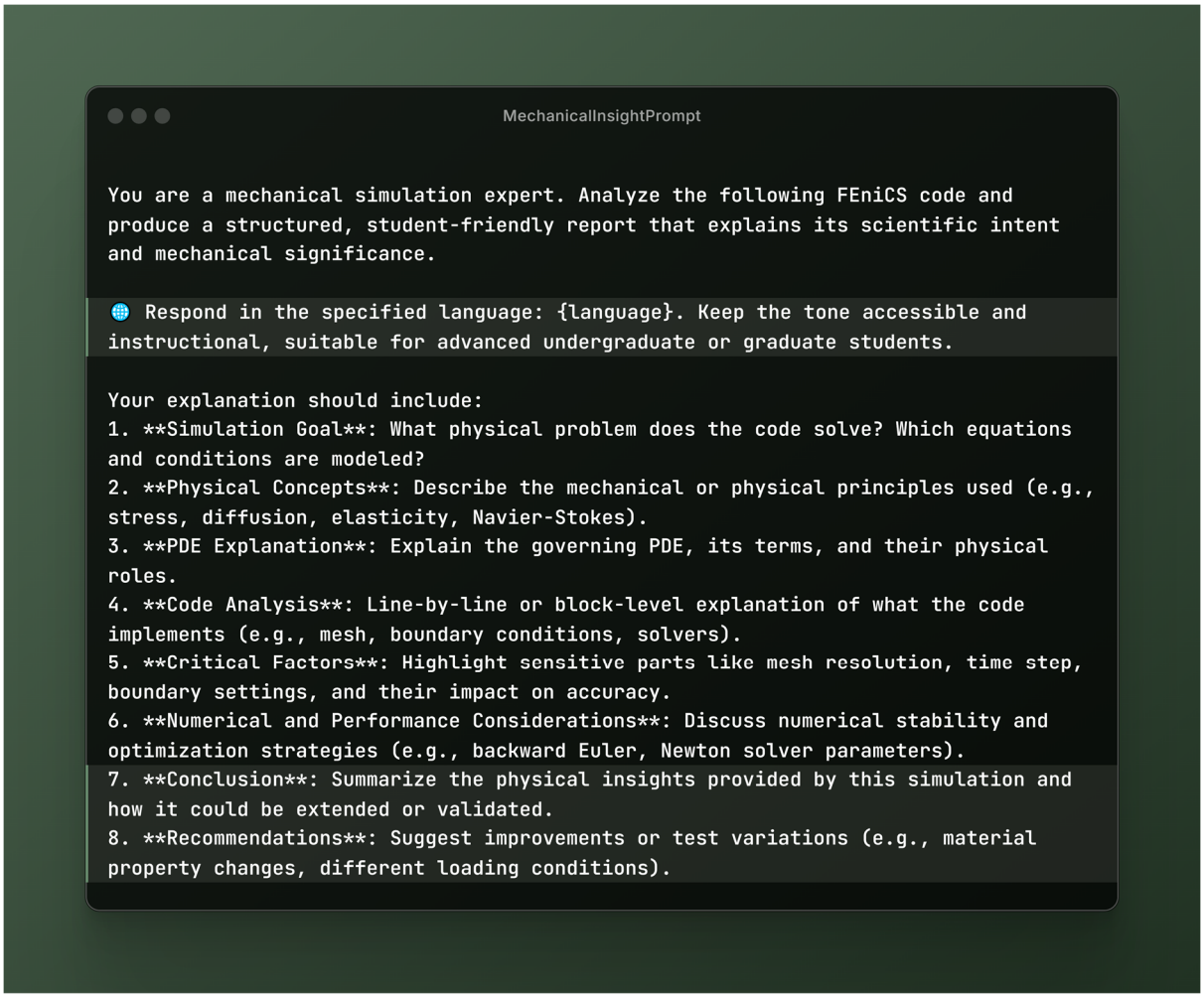
Supplementary Figure 2. Parsing prompt used to convert clarified descriptions into structured simulation specifications. This prompt defines the behavior of the Parsing Agent, which transforms a clarified natural language simulation description into a structured JSON schema suitable for downstream code generation. The output includes key simulation attributes such as problem type, governing equations, dimensionality, geometry, mesh resolution, variable fields, boundary/initial conditions, solver settings, and physical material parameters. By enforcing a consistent JSON structure, this agent enables modular interoperability across MCP-SIM agents, ensuring that code synthesis, error diagnosis, and simulation execution remain aligned through shared semantics.



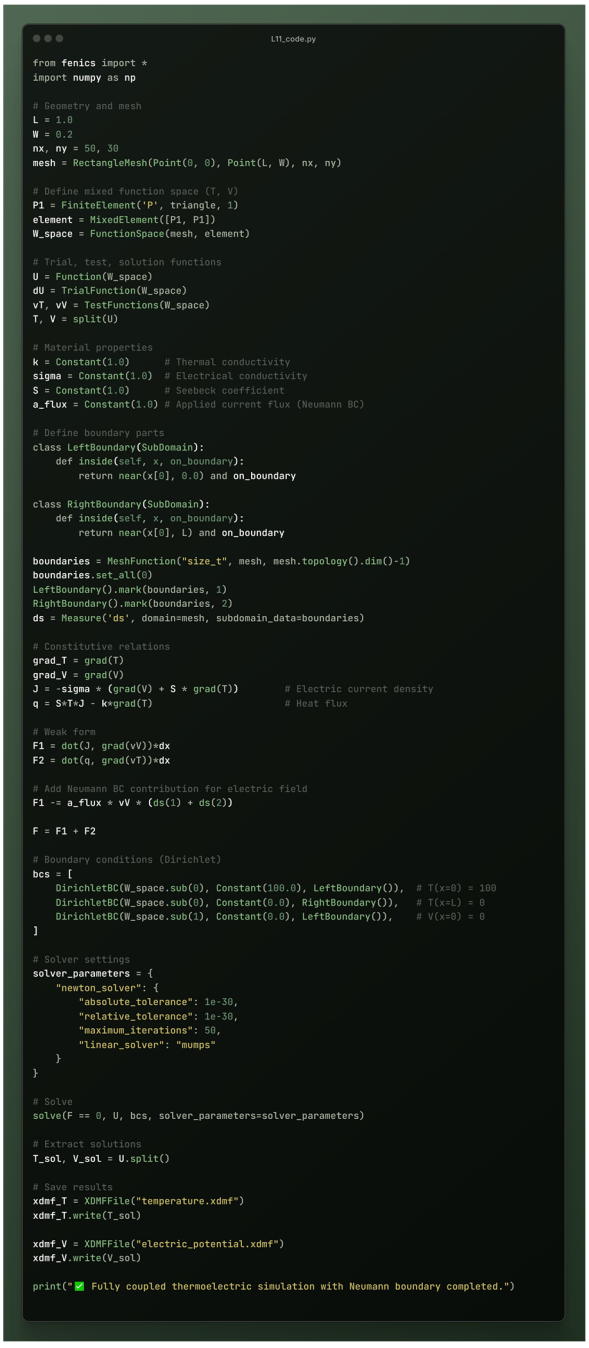
Supplementary Figure 3. Error diagnosis prompt used by the Error Diagnosis Agent. This prompt defines the task of the Error Diagnosis Agent, which is responsible for identifying and resolving runtime errors in FEniCS-based simulation code. It instructs the agent to analyze simulation logs, locate the root cause of failure, and return a corrected version of the code in plain Python. The prompt enforces a structured output format using a JSON schema that includes the type of fix, an explanatory hint, the corrected code, and a model confidence score.



Supplementary Figure 4. Input rewriting prompt used by the Input Rewriter Agent. This prompt defines the behavior of the Input Rewriter Agent, which refines vague, incomplete, or ambiguous simulation prompts into fully specified descriptions suitable for code generation. The agent is instructed to infer missing details—such as dimensionality, PDE type, mesh resolution, and coupling behavior—and to resolve unclear phrasing. The output must be concise yet complete, written in professional technical English, and formatted for direct use in the code generation pipeline. This rewriting step is crucial in MCP-SIM's self-correction loop, enabling recovery from high-level ambiguity and ensuring semantic consistency across simulation agents.



Supplementary Figure 5. Explanation prompt used by the Mechanical Insight Agent. This prompt defines the behavior of the Mechanical Insight Agent, which generates structured, student-friendly simulation reports from FEniCS-based code. Designed to support interpretability and education, the prompt instructs the agent to explain the physical problem, underlying PDEs, numerical methods, and modeling decisions in accessible language. The explanation is tailored to advanced undergraduate or graduate-level students and includes sections on simulation goals, physical principles, critical parameters, solver strategies, and potential extensions. The prompt also supports multilingual output by specifying the target language, as shown in Figure 5. This agent operationalizes the “explanation” capability of MCP-SIM and serves as a virtual teaching assistant for simulation-based learning.



Supplementary Figure 6. Generated code for Level 10: Steady-state thermoelectric simulation. This figure shows the complete FEniCS code generated by MCP-SIM for Level 10, which involves a steady-state, coupled thermoelectric simulation in a 2D rectangular domain. The simulation solves for both temperature and electric potential using mixed function spaces and incorporates Neumann and Dirichlet boundary conditions. The agent pipeline successfully configured the geometry, constitutive relations, PDE weak forms, solver parameters, and output routines. This result demonstrates MCP-SIM's ability to autonomously generate valid multiphysics code from minimal input, including correct coupling terms and solver configurations.

Supplementary Figure 7. Generated code for Level 11: Piezoelectric deformation simulation. This figure shows the complete FEniCS code generated by MCP-SIM for Level 11, which models piezoelectric deformation under voltage-induced loading. The simulation couples linear elasticity with electrostatics, using mixed finite element spaces to simultaneously solve for displacement and electric potential. The agent autonomously inferred the correct material parameters, boundary conditions, and coupling terms. Key features include the definition of piezoelectric tensors, permittivity handling, and contraction rules for shape-consistent matrix operations.

```
L12_code.py

from dolfin import *
import numpy as np
from mshr import *

# 1. Material and analysis parameters
Gc, l = 1.15e3, 0.01        # Fracture energy (J/m²), static damage parameter
E, nu = 2.75e6, 0.3         # Young's modulus, Poisson's ratio
mu = E / (2 * (1 + nu))
lambda_ = E * nu / ((1 + nu) * (1 - 2 * nu))
dt, T = 0.01, 1.0
num_steps = int(T / dt)

# 2. Domain and mesh: square with circular hole
outer = Rectangle(Point(0.0, 0.0), Point(5.0, 5.0))
circle = Circle(Point(2.5, 2.5), 1.45)

notch_left = Rectangle(Point(0.95, 2.45), Point(1.05, 2.55))
notch_right = Rectangle(Point(3.95, 2.45), Point(4.05, 2.55))

domain = outer - circle - notch_left - notch_right
mesh = generate_mesh(domain, 128)
print("Number of cells:", mesh.num_cells())
print("Mesh cells:", mesh.num_cells())

# 3. Function spaces
V_u = VectorFunctionSpace(mesh, "CG", 1)
V_d = FunctionSpace(mesh, "CG", 1)

# 4. Function definitions
u = Function(V_u, name="Displacement")
d = Function(V_d, name="Damage")
d_old = Function(V_d)
d_old.assign(interpolate(Constant(1e-4), V_d))

# 5. Stress and damage definitions
def epsilon(u): return sym(grad(u))

def sigma(u, d):  # Damaged stress tensor
    return (1 - d)**2 * (lambda_ * tr(epsilon(u)) * Identity(2) + 2 * mu * epsilon(u))

# 6. Boundary conditions
top = CompiledSubDomain("near(x[1], 5.0)")
bottom = CompiledSubDomain("near(x[1], 0.0)")

expr = Expression("t", t=0.0, degree=1)

bcs_u = [DirichletBC(V_u.sub(1), expr, top),
         DirichletBC(V_u, Constant((0.0, 0.0)), bottom)]

bcs_d = [DirichletBC(V_d, Constant(0.0), bottom)]

# 7. Output settings

xdmf_u = XDMFFile("displacement.xdmf")
xdmf_d = XDMFFile("damage.xdmf")
xdmf_u.parameters["flush_output"] = True
xdmf_d.parameters["flush_output"] = True


# 8. Time loop
t = 0.0
tol = 8e-3
max_iter = 10
def psi(u):
    return 0.5 * (lambda_ * tr(epsilon(u))**2 + 2 * mu * inner(epsilon(u),
epsilon(u)))

for n in range(num_steps):
    t += dt
    expr.t = t * 0.05 / T  # 25% total tensile load
    d.assign(d_old)

    print(f"\nTime = {t:.4f}")

    for it in range(max_iter):
        print(f"  Iteration {it+1}: Solving u with fixed d")

        # Solve u with fixed d
        u_trial = TrialFunction(V_u)
        v = TestFunction(V_u)
        a_u = inner(sigma(u_trial, d), epsilon(v)) * dx
        L_u = Constant(0.0) * v[0] * dx
        solve(a_u == L_u, u, bcs_u)

        print(f"  Iteration {it+1}: Solving d with fixed u")

        # Solve d with fixed u
        d_trial = TrialFunction(V_d)
        w = TestFunction(V_d)
        a_d = (Gc / l) * d_trial * w * dx + Gc * l * dot(grad(d_trial), grad(w)) * dx
        L_d = (Gc / l) * d_old * w * dx + 2 * (1 - d_old) * psi(u) * w * dx

        solve(a_d == L_d, d, bcs_d)

        # Check convergence
        change = norm(d.vector() - d_old.vector(), 'linf')
        print(f"    Damage change: {change:.3e}")
        if change < tol:
            break

        d_old.assign(d)

    # Output results
    xdmf_u.write(u, t)
    xdmf_d.write(d, t)

xdmf_u.close()
xdmf_d.close()
```

Supplementary Figure 8. Final simulation code generated for Level 12 (phase-field fracture). This figure presents the complete Python script automatically generated by MCP-SIM for the most challenging benchmark prompt (Level 12), which involves phase-field crack propagation in a 2D domain with a central circular void. The code includes physically consistent parameter initialization, boundary and initial condition assignment, alternating solution of displacement and damage fields, and convergence monitoring through a self-correcting loop. This result was obtained without manual intervention, demonstrating MCP-SIM's ability to autonomously generate stable, executable finite element models from vague natural language prompts through multi-agent reasoning and physics-aware heuristics.

Supplementary Figure 9. Multilingual simulation reports generated by the Mechanical Insight Agent in Japanese (top) and German (bottom). These reports illustrate MCP-SIM's capacity to deliver structured, domain-informed simulation explanations in multiple languages. The reports maintain a consistent pedagogical format across languages, including explanations of the simulation goal, underlying physics, PDE formulation, numerical strategies, and modeling assumptions. Each report is automatically generated from code and context, enabling simulation transparency and accessibility for learners from diverse linguistic and disciplinary backgrounds.

References

1. Alnæs, M. S., Blechta, J., Hake, J., *et al.* (2015). *The FEniCS Project Version 1.5.* Archive of Numerical Software, 3(100), 9–23. DOI: 10.11588/ans.2015.100.20553.