https://doi.org/10.1038/s44387-025-00057-z

A self-correcting multi-agent LLM framework for language-based physics simulation and explanation

Donggeun Park[1,2,3], Hyeonbin Moon[1,2,3] & Seunghwa Ryu[1,2] ✉

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. Across twelve tasks of increasing complexity, the framework solved all benchmark cases and improved convergence efficiency relative to GPT-based and human-in-the-loop baselines, under the specific metrics defined in this study. 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. While these results indicate strong robustness on the tested suite, performance in specialized domains and under distributions beyond our benchmark remains an area for future validation.

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[6–8]. Real-world simulation problems often lack complete specifications, including governing equations, boundary conditions, or material properties. Existing approaches to language-based simulation have notable limitations. One-shot LLM code generation methods[9–11] can produce code from text but often fail on underspecified problems due to a lack of iterative refinement. Recent multi-agent LLM frameworks have been explored in specialized contexts[12,13] – for example, a two-agent system was shown to self-correct code for solving elasticity problems[14] – but these systems remain domain-specific and require agents to manually correct each other. Physics-informed neural networks[15–18] provide another approach for solving PDEs, yet they demand fully specified equations and cannot interpret ambiguous natural language prompts.

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 Memory-Coordinated Physics-Aware Simulation, a fully autonomous simulation framework that leverages a multi-agent architecture[19–21] 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.

[1]Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea. [2]KAIST InnoCORE PRISM-AI Center, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea. [3]These authors contributed equally: Donggeun Park, Hyeonbin Moon. ✉e-mail: ryush@kaist.ac.kr

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 success on all twelve benchmark tasks under the defined metrics and exhibited robust generalization across domains, including 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 (Fig. 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" (Fig. 1, ①)—the *Input Clarifier Agent* (Fig. 1, 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* (Fig. 1, 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.

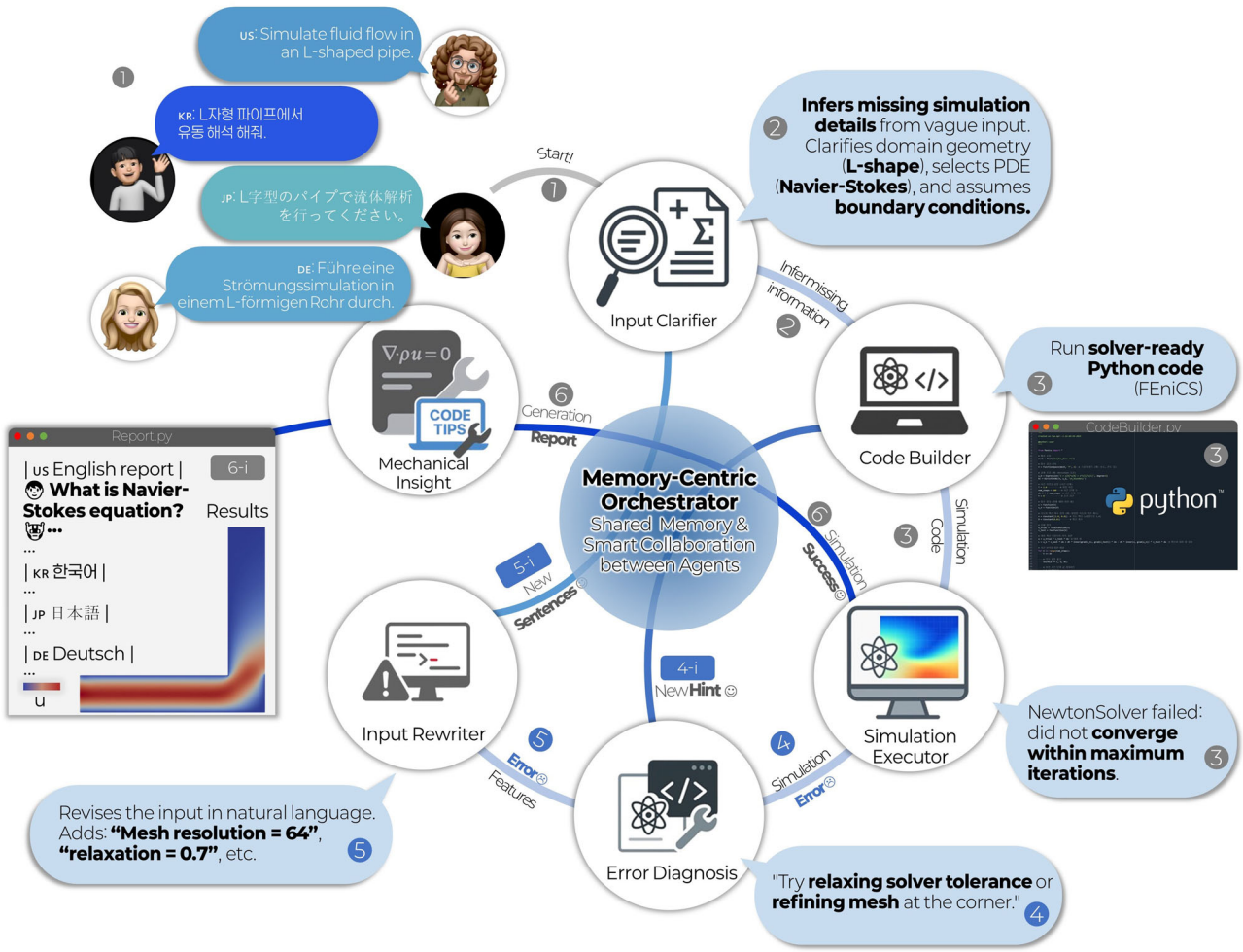

Fig. 1 | MCP-SIM workflow and agent collaboration. The system starts with a user's natural language prompt (①), which is clarified by the *Input Clarifier Agent* (②) to identify missing details (geometry, PDE type, boundary conditions). The *Input Clarifier* turns the user prompt into a canonical paragraph and a structured JSON for code generation (parsing module within the Clarifier). The *Code Builder Agent* (③) then generates solver-ready Python code, which is executed by the *Simulation Executor Agent* (④) while monitoring for errors or physical anomalies. If an error occurs, the *Error Diagnosis Agent* (⑤) identifies and corrects it; if the initial instructions are ambiguous, the *Input Rewriter Agent* (⑤-i) refines the prompt and the cycle repeats. Finally, the Mechanical Insight Agent (⑥) produces a multilingual report explaining the simulation's physical principles and results, demonstrating how MCP-SIM yields accurate, executable, and educational outputs through coordinated agent interaction.

All assumptions, numerical strategies, and intermediate artifacts are persistently logged, ensuring reproducibility and traceability.

The resulting code is executed by the *Simulation Executor Agent* (Fig. 1, 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* (Fig. 1, ④) is invoked. The *Error Diagnosis Agent* (Fig. 1, 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 (Fig. 1, ④-i) for subsequent revision. If the issue stems from high-level ambiguity in the user prompt, the *Input Rewriter Agent* (Fig. 1, ⑤-i) semantically revises the instruction (e.g., adding "mesh resolution = 64" or "relaxation = 0.7"), and the control loop restarts from clarification (Fig. 1, ②).

Upon successful simulation, the *Mechanical Insight Agent* (Fig. 1, 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 (Fig. 1, 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 computations 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 Figs. 1–5. For a demonstration of MCP-SIM in action, please refer to Supplementary Video 1.

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 (Fig. 2A; Table 1). The prompt design for our benchmark tasks follows a gradually increasing complexity (from fully specified to highly ambiguous problems), akin to a curriculum. This approach was inspired in part by conventional finite element tutorials[22], ensuring that the system is challenged across a spectrum from basic to advanced scenarios. 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 methodologies–a standard approach for evaluating the impact of system components–across the benchmark[23,24]. We compare three representative baselines in addition to our proposed MCP-SIM: (i) B1. *One-shot GPT*, which generates code once from a prompt while leaving specification completion, error inspection, and correction to the user; (ii) *B2. GPT + Automated Clarifier*, where the *Input Clarifier Agent* automatically infers missing specifications but execution monitoring and error handling remain manual; and (iii) *B3. GPT + Clarifier + Human Diagnosis (human-in-the-loop hybrid)*, where specifications are auto-inferred and the *Error Diagnosis Agent* proposes fixes, but the user must review and decide whether to accept these suggestions before re-running. These baselines all rely on user mediation, process code and errors sequentially, and cannot exploit the history of past error–fix attempts, which limits robustness under underspecified prompts.

In this study, we quantified performance using three metrics: (i) success rate, defined as the proportion of tasks solved out of 12; (ii) convergence efficiency, measured by the number of Plan → Act → Reflect → Revise cycles required until success; and (iii) numerical convergence, defined as achieving solver residuals below 10^{-4} while producing physically plausible field distributions. Specifically, once baselines and MCP-SIM successfully generated executable simulation code, we evaluated physical convergence by computing the residual as the total imbalance of each governing equation across all cells for the twelve tasks. This residual was normalized by the initial value at every iteration to yield a dimensionless residual, independent of physical units. Convergence was determined when this normalized residual dropped below the prescribed threshold (10^{-4}). These metrics provide a consistent basis for comparing baselines (B1–B3) and our fully autonomous MCP-SIM (Fig. 2A–C).

As shown in Fig. 2B, B1 solved only 6/12 tasks, B2 improved to 8/12, and B3 reached 10/12. In contrast, MCP-SIM solved all 12/12 tasks. In addition, we evaluated convergence efficiency by measuring the number of Plan–Act–Reflect–Revise cycles required for each task. Unlike B1–B3, which only act on the current run, MCP-SIM leverages persistent memory that stores clarifications, code snapshots, execution traces, and error → fix mappings from previous iterations. Agents consult this record to avoid regressions and propose consolidated, multifaceted corrections in one step. This history-aware mechanism explains MCP-SIM's ability to converge within ≤5 iterations across most benchmark tasks (Fig. 2C).

Building on these performance metrics, Fig. 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 dimensionless residual below 10^{-4}. 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 suggest 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.

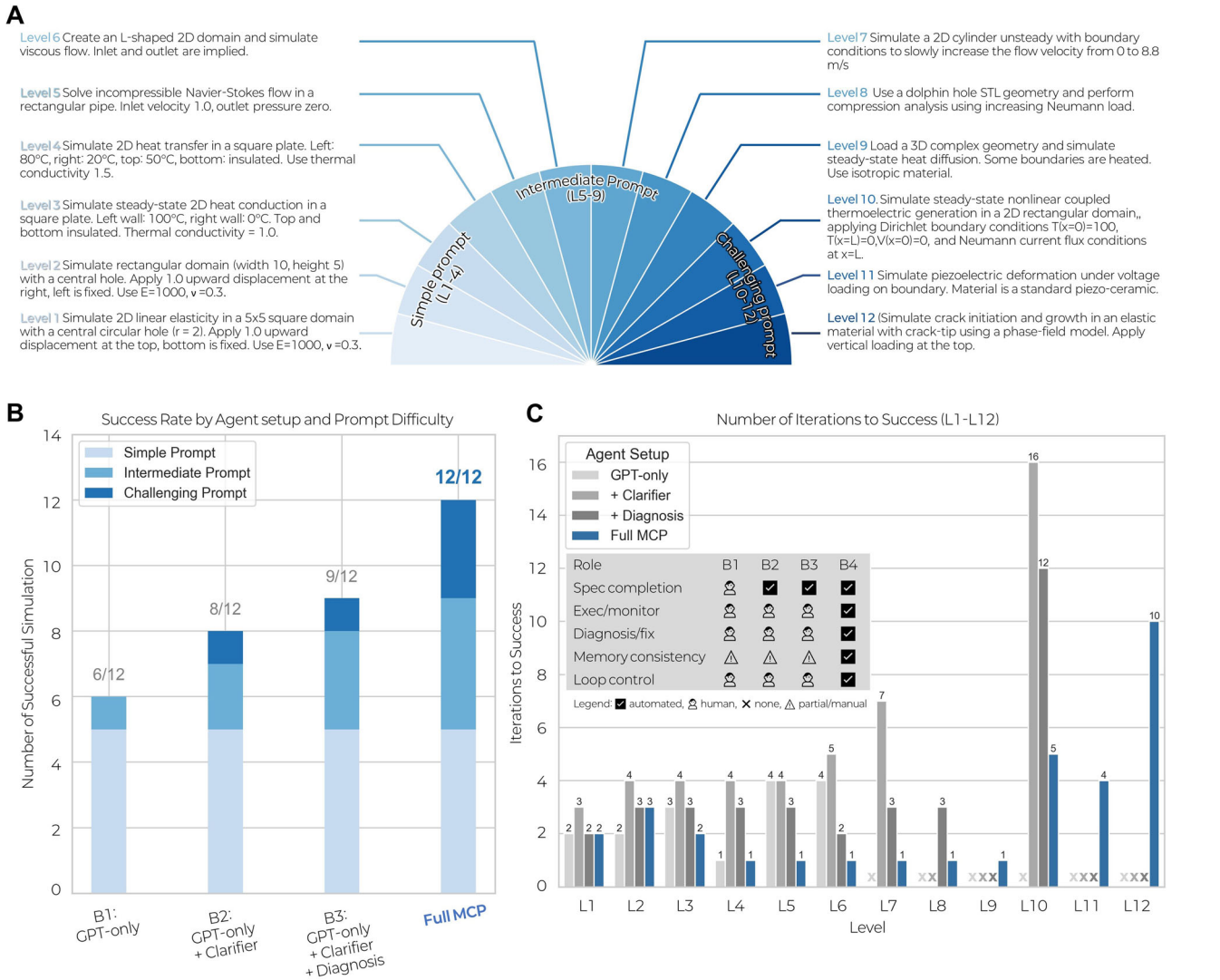

Fig. 2 | Benchmark evaluation of MCP-SIM across varying simulation complexities. A Twelve benchmark tasks categorized by increasing difficulty—from well-posed, single-physics problems (Levels 1–4) to under-specified, multi-physics challenges (Levels 10–12). **B** Success rates across three benchmarks and MCP-SIM: B1. One-shot GPT, B2. GPT + Automated Clarifier, B3. GPT + Clarifier + Human Diagnosis (human-in-the-loop hybrid), and 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. The inset table in (**C**) summarises the automation scope of each benchmark, showing which stages (specification completion, execution/monitoring, diagnosis/fix, memory consistency, loop control) are automated (✓), human-mediated (👤), or partially manual (⚠). MCP-SIM uniquely integrates all stages into a fully automated, memory-centric loop.

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[25] 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.

Figure 4B illustrates the problem setup, featuring a central circular void in a square domain subjected to quasi-static uniaxial tension, with the bottom boundary fixed and the top boundary displaced upward by 5%. After ten self-correcting cycles, the crack propagation patterns simulated by ABAQUS and the MCP framework show close agreement in crack initiation and growth direction, thereby validating the effectiveness of the autonomous solver (Fig. 4C). 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 Figs 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 (Fig. 5), as well as Japanese and German (Supplementary Fig. 9), and follow a pedagogically

Table 1 | Overview of MCP-SIM test levels and objectives

Level	Problem Type	Challenge	Key Parameters/Omissions	Objective
1	2D Linear Elasticity	Simple case, fully specified	Geometry (5×5 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

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.

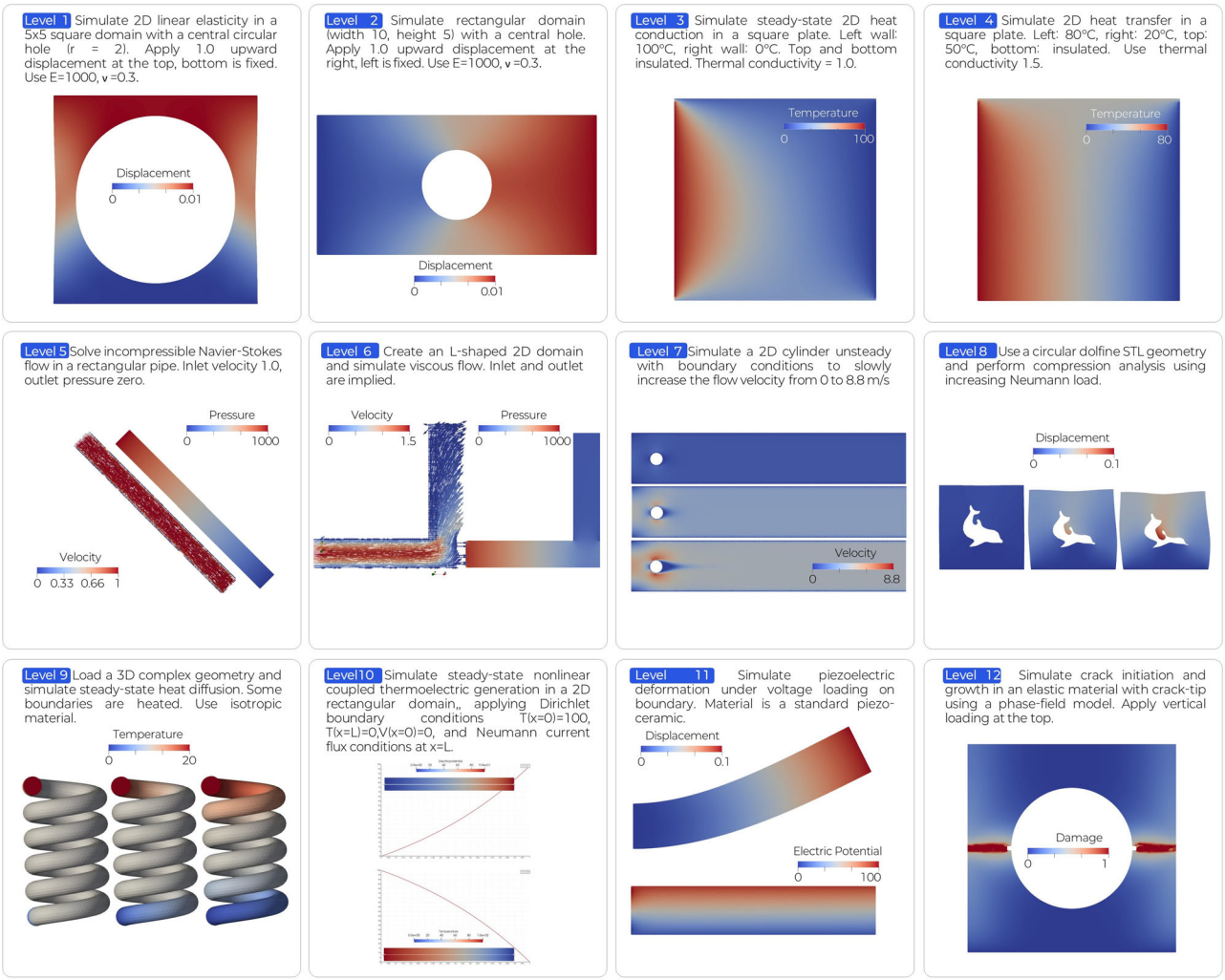

Fig. 3 | Simulation results for the 12 benchmark problems used to validate MCP-SIM across different physics domains. (Top-left to bottom-right): This presents the final simulation outputs from MCP-SIM for each of the 12 levels of complexity, ranging from basic linear elasticity (Level 1) to advanced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

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[26].

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—where all generated code for simulations satisfied the numerical convergence criterion (dimensionless residual $<10-4$)—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 promising results, several limitations remain. First, MCP-SIM is based on a general-purpose LLM (GPT-4o) without domain-specific fine-tuning; performance may degrade in specialized physical settings, such as rare material models or advanced multiphysics couplings. Second, the reflective loop introduces computational latency compared to one-shot generation, which may restrict applicability in real-time or resource-constrained scenarios. Third, our evaluation is limited to a curated twelve-task benchmark with synthetic prompts; extension to experimental or safety-critical contexts will require domain-specific validation and human oversight. Finally, while

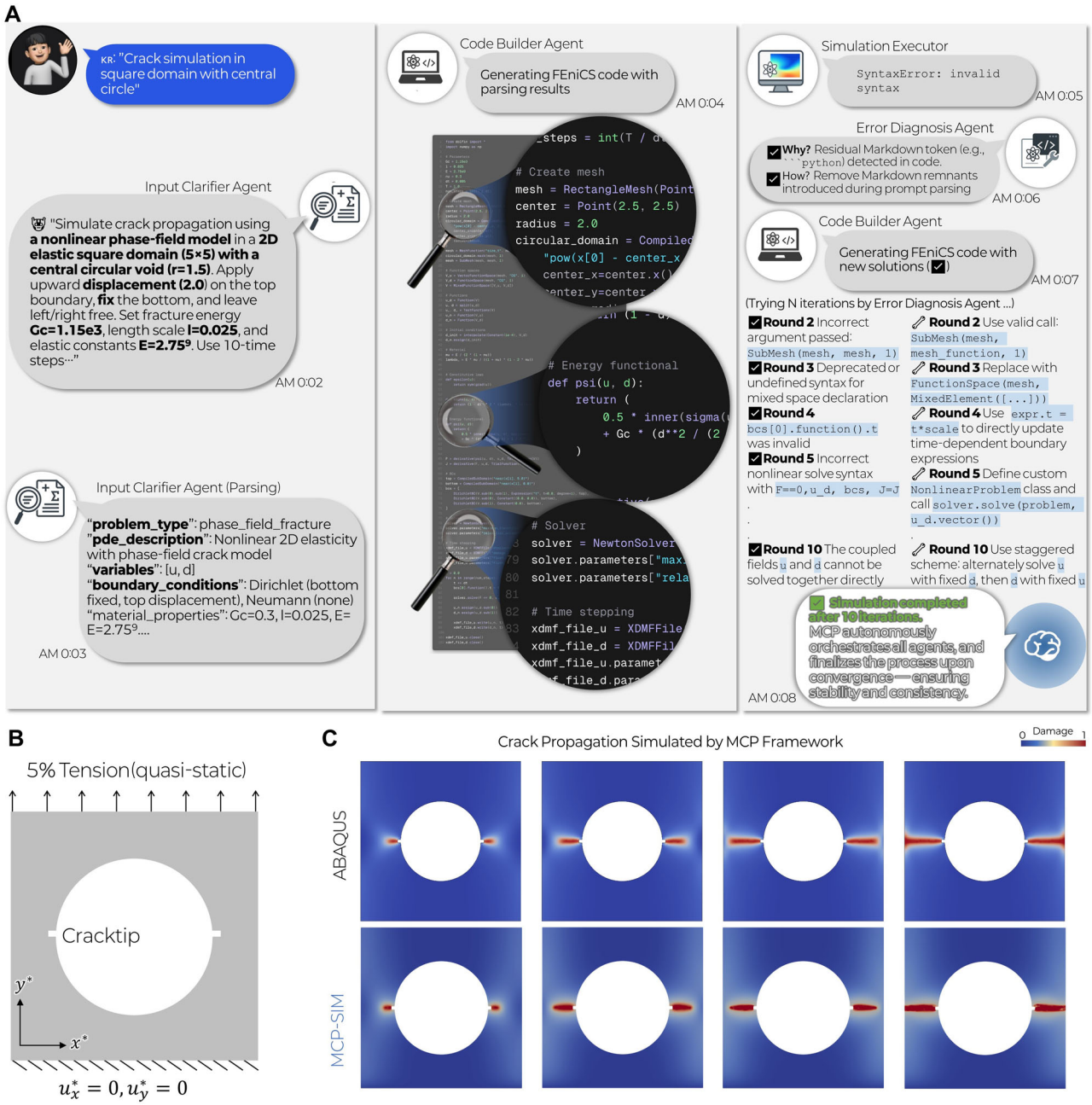

Fig. 4 | Autonomous reproduction of expert-designed fracture simulations from minimal prompts. A Agent reasoning process for Level 12: From a single-sentence prompt, MCP-SIM infers geometry, governing equations, and solver setup through iterative agent collaboration. **B** Problem setup: a square domain 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.

persistent memory reduces regressions in our setup, long-horizon stability under noisy or adversarial inputs remains an open research direction. 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, it exemplifies a new paradigm for AI-driven scientific modeling that blends computational accuracy with 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, making simulation not only more powerful but also more adaptive and scientifically grounded.

Methods

Agent framework design

Each specialized agent in MCP-SIM follows a well-defined algorithm encoded via a prompt template and executed using GPT-4o (OpenAI,

Fig. 5 | Mechanical insight agent: simulation reports - English and Korean. This 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.

2024). Specifically: *the Input Clarifier Agent* refines vague user requests into a fully specified description and, as an internal parsing module, emits a structured JSON object for downstream code generation (see Supplementary Fig. 2). The *Code Builder Agent* generates solver-ready Python/FEniCS code from the JSON and natural language specification; the *Simulation Executor Agent* executes the code and captures logs or errors; the *Error Diagnosis Agent* analyses error messages and proposes corrected code via JSON-formatted outputs; the *Input Rewriter Agent* revises the original prompt if high-level ambiguities persist, using hints from the diagnosis and prior outputs; and the *Mechanical Insight Agent* produces multilingual, student-friendly reports explaining the physical and numerical aspects of the generated code. All agents are orchestrated by a memory-centric controller that stores clarifications, parsed data, code versions, execution logs, and applied fixes, thereby enabling reproducibility and the iterative Plan → Act → Reflect → Revise loop underpinning MCP-SIM's self-correcting behavior. The detailed prompt templates defining each agent's behaviour are provided in Supplementary Note 1 and Supplementary Figs 1–5 for full transparency.

In contrast to prior baselines (B1–B3), where users remain in the loop for either specification completion or reviewing error corrections, MCP-SIM eliminates manual mediation by embedding all tasks—clarification, code generation, execution, diagnosis, and rewriting—within a memory-centric loop. The orchestrator maintains a JSON-based log of specifications, code versions, error traces, and applied fixes, enabling history-aware corrections and preventing regressions.

Simulation backend and libraries

In MCP-SIM, the 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.

Data availability

No experimental or observational data was utilized in this work.

Code availability

The codes used for this work are available at: https://doi.org/10.5281/zenodo.15645333 and https://github.com/KAIST-M4/MCP-SIM.

Received: 7 August 2025; Accepted: 22 November 2025;
Published online: 20 January 2026

References

1. Buehler, M. J. PRefLexOR: preference-based recursive language modeling for exploratory optimization of reasoning and agentic thinking. *npj Artif. Intell.* **1**, 4 (2025).
2. Xue, C. et al. Omniforce: on human-centered, large model empowered and cloud-edge collaborative AutoML system. *npj Artif. Intell.* **1**, 3 (2025).
3. Romera-Paredes, B. et al. Mathematical discoveries from program search with large language models. *Nature* **625**, 468–475 (2024).
4. Zhang, X., Zheng, X. & Luo, J. The influence of stereotypes and visual features on donation intentions in online medical crowdfunding campaigns: A comparison of survey and large language model-based methods. *npj Artif. Intell.* **1**, 16 (2025).
5. Brown, T. B. et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **33**, 1877–1901 (2020).
6. Steyvers, M. et al. What large language models know and what people think they know. *Nat. Mach. Intell.* **7**, 221–231 (2025).
7. Musslick, S. et al. Automating the practice of science: opportunities, challenges, and implications. *Proc. Natl. Acad. Sci.* **122**, e2401238121 (2025).
8. Krenn, M. et al. On scientific understanding with artificial intelligence. *Nat. Rev. Phys.* **4**, 761–769 (2022).
9. Luo, X. et al. Large language models surpass human experts in predicting neuroscience results. *Nat. Hum. Behav.* **9**, 305–315 (2025).
10. Chen, L., Rao, Z. & Singh, S. Self-planning code generation with large language models: plan-then-implement framework. *ACM Trans. Softw. Eng. Methodol.* **33**, 21:1–21:25 (2024).
11. Le, K. T. & Andrzejak, A. Rethinking AI code generation: a one-shot correction approach based on user feedback. *Automated Softw. Eng.* **31**, 60 (2024).
12. Dong, Z., Lu, Z. & Yang, Y. Fine-tuning a large language model for automating computational fluid dynamics simulations. *Theor. Appl. Mech. Lett.* **15**, 100594 (2025).
13. Kim, S., Yu, Y. & Seo, H. Artificial intelligence orchestration for text-based ultrasonic simulation via self-review by multi-large language model agents. *Sci. Rep.* **15**, 12474 (2025).
14. Ni, B. & Buehler, M. J. MechAgents: large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge. *Extrem. Mech. Lett.* **67**, 102131 (2024).
15. Karniadakis, G. E. et al. Physics-informed machine learning. *Nat. Rev. Phys.* **3**, 422–440 (2021).
16. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **378**, 686–707 (2019).
17. Lu, L., Meng, X., Mao, Z. & Karniadakis, G. E. Scientific machine learning through physics-informed neural networks: trends and frontiers. *J. Sci. Comput.* **92**, 57 (2022).
18. Zhang, C. & de Lillo, F. A physics-informed neural framework for inversion and surrogate modeling in solid mechanics. *Comp. Methods Appl. Mech. Eng.* **379**, 113741 (2022).
19. Vinyals, O. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **575**, 350–354 (2019).
20. Bae, H. J. & Koumoutsakos, P. Scientific multi-agent reinforcement learning for wall-models of turbulent flows. *Nat. Commun.* **13**, 1443 (2022).
21. Ghafarollahi, A. & Buehler, M. J. SciAgents: automating scientific discovery through bioinspired multi-agent intelligent graph reasoning. *Adv. Mater,* **37,** e2413523 (2025).
22. Alnæs, M. S. et al. The FEniCS Project Version 1.5. *Arch. Numer. Softw.* **3**, 9–23 (2025).
23. Meyes, R., Lu, M., De Puiseau, C. W. & Meisen, T. *Ablation studies in artificial neural networks*, (2019). *arXiv preprint* arXiv:1901.08644.
24. Vishnusai, Y., Kulakarni, T. R. & Nag, K. S. Ablation of artificial neural networks. *International Conference on Innovative Data Communication Technologies and Application*, 453–460 (Springer, 2019).
25. Storvik, E. et al. An accelerated staggered scheme for variational phase-field models of brittle fracture. *Computer Methods Appl. Mech. Eng.* **381**, 113822 (2021).
26. Qin, L. et al. A survey of multilingual large language models. *Patterns* **6**, 101118 (2025).

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant (RS-2025-16070951) and by the InnoCORE program funded by Ministry of Science and ICT (N10250154). We also acknowledge the Yoon Young Kim AI Innovation Fund of The Korean Society of Mechanical Engineers (KSME).

Author contributions

D.G.P. developed the Multi-agent LLM systems, prepared the figures, and wrote the initial draft. H.B.M. implemented the model and validated it through simulation. S.H.R. supervised the study, provided conceptual guidance, and reviewed the manuscript. All authors reviewed and approved the final version of the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s44387-025-00057-z.

Correspondence and requests for materials should be addressed to Seunghwa Ryu.