MOSAIC: Multi-agent Orchestration for Task-Intelligent Scientific Coding

Siddeshwar Raghavan*
Elmore School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, 47906
raghav12@purdue.edu

Tanwi Mallick
Mathematical and Computer Sciences
Argonne National Laboratory
Lemont, IL, 60439
tmallick@anl.gov

Abstract

We present MOSAIC, a large language model (LLM) based multi-agent framework for tackling complex scientific coding tasks. Unlike general-purpose programming, scientific coding requires rigorous algorithmic reasoning, substantial domain expertise, high numerical precision, extended context management, and the ability to decompose and solve interdependent subproblems under a larger objective. To address these challenges, we design and integrate specialized agents responsible for self-reflection, planning, coding, and debugging inspired by a teacher–student paradigm. This architecture balances open-ended exploration, ensuring executability while maintaining a consolidated context to minimize hallucinations. We evaluate MOSAIC on the scientific coding benchmark SciCode and show that our framework, together with its specialized agents, outperforms existing approaches in accuracy, robustness, and interpretability.

1 Introduction

Large Language Models (LLMs) have recently shown significant advancements, exhibiting capabilities in reasoning, strategic planning, and task execution that are comparable to human cognition. These advancements have led to the development of multi-agent frameworks, where specialized agents coordinate to solve intricate challenges that would be challenging for a single model.

Despite these breakthroughs, scientific coding presents unique challenges. Scientific workflows require precise numerical methods, rigorous error propagation, and strict reproducibility, which often exceed the current capabilities of both monolithic LLMs and simple agent setups. The presence of domain-specific knowledge, intricate data dependencies, and the need for formal validation further complicate code generation and can introduce subtle, difficult-to-detect errors. Scientific coding datasets like SciCode [10] offer no I/O examples for algorithmic validation, and generating them is far from trivial. This fundamental difference makes prior multi-agent frameworks such as MapCoder [7], CodeSIM [8], and AgentCoder [6] unsuitable for scientific settings, as their code generating pipeline rely on the availability of test cases.

*Work done at Argonne National Laboratory as a **Givens Associate**

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: DL4C.

Decomposing intricate scientific problems into coherent, solvable sub-tasks demands both deep domain expertise and careful orchestration. At the same time, as interdependent sub-tasks grow longer, LLMs struggle to retain critical context and are increasingly prone to hallucinations and inconsistencies when operating near the limits of their context windows [11], further undermining their reliability for end-to-end scientific code generation.

Motivated by this research gap, we introduce MOSAIC, a fully autonomous, LLM-agnostic multi-agent framework for scientific code generation that operates without the need for validation I/O test cases. In MOSAIC we design and use four specialized agents: self-reflection agent, inspired by Renze et al. [9], rationale agent to plan the solution, and iterative Coding and Debugger agents that converts these plans into executable, modular code and in-turn focuses on syntactic correctness. To preserve context between interdependent subproblems, MOSAIC also implements a lightweight context window strategy.

Our comprehensive evaluation across three LLM backbones and four optimization baselines (direct synthesis, chain-of-thought prompting, self-planning, analogical reasoning) demonstrates that MOSAIC substantially enhances scientific code generation, outperforming every comparative method achieving **18.64% main problem** solving rate and **39.92% subproblem solving** rate, compared to the **baseline of 10.76%** and **33.01% respectively**. To showcase versatility of MOSAIC, we also evaluate on MBPP [1], HumanEval [3], and APPS [4], where we achieve performance comparable to existing multi-agent frameworks.

2 Method

We present MOSAIC, a modular multi-agent LLM framework designed to decompose complex scientific coding problems into accurate and executable solutions. The framework comprises four core agents: Self-Reflection, Rationale, Coding, and Debugger which collaborate autonomously to generate and refine code, while a Bucketing module directs problems to domain relevant agent groups for targeted problem solving. Inspired by knowledge distillation [5], MOSAIC follows a student–teacher paradigm. The Teacher uses a small subset of ground-truth code from the validation set as in-domain examples to generate detailed rationales. These do not overlap with the test problems, instead, they serve as guidance for decomposing unseen domain problems into logical steps. The Student agents then transform these steps into reliable code with the aid of a consolidated context window, which retains only function signatures and concise summaries to preserve essential context without overloading memory with the full history.

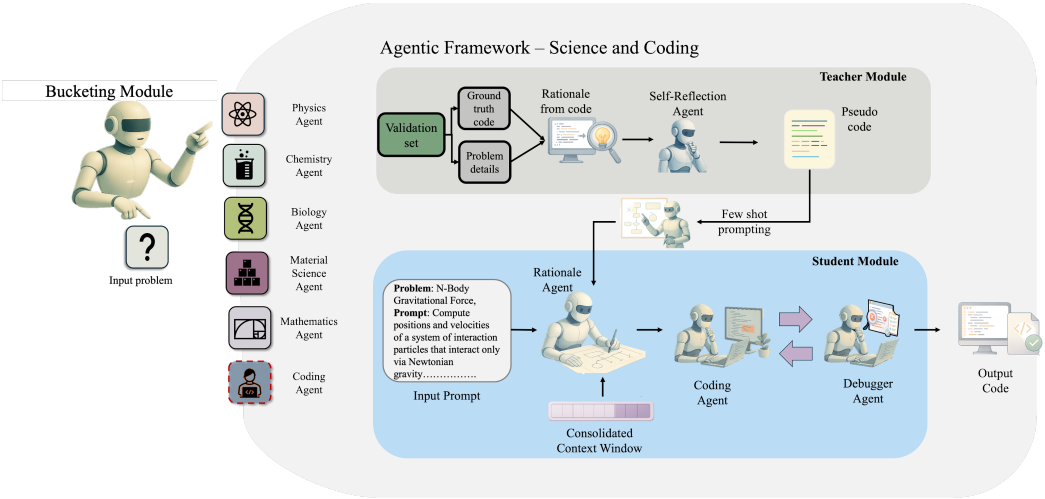

Figure 1: **Overview of MOSAIC**: a four-agent framework inspired by knowledge distillation, where the Teacher module guides the Student agents to generate executable scientific code.

2.0.1 Self-Reflection Agent

The agent receives the ground truth rationale and learns to evaluate its own intermediate reasoning steps. It identifies potential mistakes or omissions and refines its logic iteratively before arriving at the final pseudocode. By verbalizing its thought process and critically analyzing its reasoning path, the **Self-Reflection Agent** enhances output reliability, corrects logical flaws, and improves overall accuracy.

2.0.2 Rationale Agent

The **Rationale Agent** uses the generated pseudocode as few-shot examples to process scientific prompts from the test set (new problems). It then produces a clear, step-by-step reasoning plan similar to the structured guidance offered by the teacher model. Scientific problems typically involve a sequence of interdependent sub-problems that need to be addressed to reach the final solution. To address the risk of hallucination as the context window grows [11, 2], we implement a **Consolidated Context Window (CCW)** to remain focused on the current task. To ensure efficiency, the CCW contains only prior function headers and brief one-sentence summaries, rather than full code history.

2.0.3 Coding Agent

The Coding Agent uses the detailed plan provided by the Rationale Agent to generate the corresponding code block, maintaining awareness of both the subproblem and the broader problem context.

2.0.4 Debugger Agent

The final core agent in the framework is the **Debugger Agent**, which executes the generated code and performs up to k rounds of error correction in collaboration with the Coding Agent. This iterative process resolves syntax and import errors, ensuring the final output is executable. Even in the absence of explicit I/O test cases, the debugger resolves syntax and import errors using examples from related domains.

3 Experiments

We evaluate MOSAIC primarily on scientific benchmark SciCode [10], general-purpose, and competitive coding benchmarks, comparing against strong baselines and current best performing frameworks.

3.1 Datasets and Comparison Approaches

Our main evaluation uses SciCode [10] (283 subproblems across physics, chemistry, biology, mathematics, and materials science) which requires generating executable code without I/O supervision. We also test on MBPP [1] (1k problems), HumanEval [3] (164 problems), and APPS [4] (5k problems) to assess applicability. Baselines include Direct synthesis, Chain-of-Thought prompting, Self-Planning, and Analogical reasoning. On general-purpose datasets, we additionally compare with MapCoder and CodeSIM which has the current best performance.

3.2 Implementation Details

We build on the open-source PyTorch implementation of SciCode [10] for our experiments. Within MOSAIC, we employ LangGraph to orchestrate agent communication and ensure reproducibility. For each problem domain, we instantiate a dedicated agent framework to encapsulate only the domain knowledge and prevent cross-domain interference. Each agent is guided by tailored prompts that constrain it to its specific role and yield the outputs needed to arrive at the final solution. In MOSAIC's teacher module, designed for knowledge distillation via few-shot prompting, we sample twenty problems from the APPS training set, ten MBPP problems and five problems from the HumanEval dataset [3] for few-shot mechanism.

4 Results and Discussion

Table 1 summarizes performance on SciCode across three LLM backbones. MOSAIC consistently outperforms all baselines in both main and subproblem solving using GPT-4o, Claude Sonnet 4, and Google Gemini 2.5 Flash. We attribute these gains to our multi-agent orchestration strategy, inspired by knowledge distillation, which guides both algorithmic planning and execution.

LLM Backbone	Methods	Total		Physics		Chemistry		Biology		Material Science		Mathematics	
GPT-4o	SciCode Baseline	7/65	94/283	3/30	48/145	1/7	13/42	0/7	5/25	2/11	24/50	1/10	4/24
	Analogical	1/65	32/283	1/30	18/145	0/7	2/42	0/7	3/25	0/11	6/50	0/10	3/24
	CoT	2/65	38/283	1/30	21/145	0/7	2/42	0/7	3/25	1/11	8/50	0/10	4/24
	LATS	4/65	49/283	2/30	34/145	0/7	2/42	0/7	3/25	2/11	8/50	0/10	2/24
	MOSAIC (*ours*)	**12/65**	**113/283**	**4/30**	**56/145**	**2/7**	**14/42**	**0/7**	**7/25**	**3/11**	**26/50**	**3/10**	**10/24**
Claude Sonnet 4	SciCode Baseline	9/65	109/283	4/30	71/145	1/7	13/42	1/7	9/25	2/11	8/50	1/10	8/24
	MOSAIC (*ours*)	**13/65**	**118/283**	**4/30**	**77/145**	**2/7**	**17/42**	**1/7**	**8/25**	**3/11**	**8/50**	**3/10**	**8/24**
Gemini 2.5 flash	SciCode Baseline	7/65	112/283	5/30	67/145	1/7	14/42	1/7	11/25	2/11	9/50	1/10	1/24
	MOSAIC (*ours*)	**11/65**	**117/283**	**5/30**	**88/145**	**2/7**	**6/42**	**1/7**	**5/25**	**2/11**	**6/50**	**1/10**	**12/24**

Table 1: Performance comparison between baselines and MOSAIC on scientific datasets with different LLM backbones. **Best** results are highlighted. The SciCode benchmark consists of 65 main problems comprising a total of 283 subproblems spanning physics, chemistry, biology, materials science, and mathematics. A problem is considered solved only when all of its subproblems pass the corresponding test suites.

Table 1 shows performance varies by domain, physics, chemistry, material science and mathematics show improvements, while biology remains the most challenging. Beyond limited in-domain example coverage, we observed incorrect order of steps and oversimplified algorithm logic. Such errors are qualitatively different from those in other domains, where failures are more toward numerical precision rather than conceptual. These findings suggest that future work may have to focus more on integrating external scientific knowledge sources through research papers, coding databases and GitHub repositories alongside agent orchestration. We also observe difficulties with very long problems (>10 subproblems), where maintaining context across multiple steps remains an open challenge despite our consolidated context window.

Despite these challenges, **MOSAIC** achieves substantial gains through a multi-agent design without any domain specific fine tuning. A primary contributor is the use of separate memory for each domain, which prevents interference across domains. With domain scoped memory, the Rationale Agent produces more robust and contextually appropriate plans; for example, an effective strategy for a physics problem can differ significantly from that for a mathematics problem.

To showcase applicability and performance on general purpose coding datasets, we also evaluated MOSAIC on MBPP, HumanEval, and APPS. Results in Table 2 show competitive performance with existing multi-agent frameworks (MapCoder, CodeSIM), indicating that MOSAIC performs well, beyond scientific coding tasks.

Method	HumanEval (30 val / 134 test)	MBPP (10 val / 500 test)	APPS
Direct	89.63	48.80	12.70
CoT	87.20	54.92	11.30
Self-Planning	89.63	49.64	14.70
Analogical	90.85	49.81	12.00
MapCoder	90.26	77.92	22.37
CodeSIM	**93.60**	**80.49**	**22.56**
MOSAIC (*ours*)	**92.53**	**84.90**	**24.71**

Table 2: Performance comparison between current SoTA and our method on general purpose coding and competitive coding datasets. **Best** and **Second Best** results are highlighted.

5 Conclusion and Future Work

In this paper, we introduced MOSAIC, an intelligent multi-agent orchestration framework for tackling complex scientific coding challenges. Drawing inspiration from knowledge distillation, MOSAIC employs a teacher–student paradigm in which the teacher guides the student through few-shot prompting to generate pseudocode. The framework mitigates hallucinations via a consolidated context window and performs iterative debugging without the need for sample test sets, thereby achieving substantial improvements over baseline performance. Experiments validate the effectiveness of the proposed approach. For future work, we intend to investigate fine-tuning strategies that incorporate domain-specific knowledge to further enhance performance. Moreover, we plan to explore heterogeneous agent configurations in which multiple LLM backbones, each specialized for a particular role, operate collaboratively within the same framework.

6 Acknowledgement

This research was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Computing Research, through the SciDAC-RAPIDS2 institute under Contract DE-AC02-06CH11357.

References

[1] J. Austin, A. Odena, M. I. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. V. Le, and C. Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL `https://arxiv.org/abs/2108.07732`.

[2] S. Banerjee, A. Agarwal, and S. Singla. Llms will always hallucinate, and we need to live with this. *arXiv preprint arXiv:2409.05746*, 2024. URL `https://arxiv.org/abs/2409.05746`.

[3] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[4] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, and J. Steinhardt. Measuring coding challenge competence with apps. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[5] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. URL `https://arxiv.org/abs/1503.02531`.

[6] D. Huang, J. M. Zhang, M. Luck, Q. Bu, Y. Qing, and H. Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2024. URL `https://arxiv.org/abs/2312.13010`.

[7] M. A. Islam, M. E. Ali, and M. R. Parvez. Mapcoder: Multi-agent code generation for competitive problem solving. *arXiv preprint arXiv:2405.11403*, 2024. URL `https://arxiv.org/abs/2405.11403`.

[8] M. A. Islam, M. E. Ali, and M. R. Parvez. Codesim: Multi-agent code generation and problem solving through simulation-driven planning and debugging. *arXiv preprint arXiv:2502.05664*, 2025. URL `https://arxiv.org/abs/2502.05664`.

[9] M. Renze and E. Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.

[10] M. Tian, L. Gao, S. D. Zhang, X. Chen, C. Fan, X. Guo, R. Haas, P. Ji, K. Krongchon, Y. Li, S. Liu, D. Luo, Y. Ma, H. Tong, K. Trinh, C. Tian, Z. Wang, B. Wu, Y. Xiong, S. Yin, M. Zhu, K. Lieret, Y. Lu, G. Liu, Y. Du, T. Tao, O. Press, J. Callan, E. Huerta, and H. Peng. Scicode: A research coding benchmark curated by scientists. *arXiv preprint arXiv:2407.13168*, 2024. URL `https://arxiv.org/abs/2407.13168`.

[11] Z. Zhang, C. Wang, Y. Wang, E. Shi, Y. Ma, W. Zhong, J. Chen, M. Mao, and Z. Zheng. Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation. *Proc. ACM Softw. Eng.*, 2(ISSTA):1–23, June 2025. doi: 10.1145/3728894.