RE^4: SCIENTIFIC COMPUTING AGENT WITH REWRITING, RESOLUTION, REVIEW AND REVISION

Ao Cheng[1,2], **Lei Zhang**[1,2*], **Guowei He**[1,2†]
[1]The State Key Laboratory of Nonlinear Mechanics,
Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China
[2]School of Engineering Science,
University of Chinese Academy of Sciences, Beijing 100049, China

ABSTRACT

Large language models (LLMs) serve as an active and promising field of generative artificial intelligence and have demonstrated abilities to perform complex tasks in multiple domains, including mathematical and scientific reasoning. In this work, we construct a novel agent framework for solving representative problems in scientific computing. The proposed agent, incorporating a "rewriting-resolution-review-revision" logical chain via three reasoning LLMs (functioning as the Consultant, Reviewer, and Programmer, respectively), is integrated in a collaborative and interactive manner. The Consultant module endows the agent with knowledge transfer capabilities to link problems to professional domain insights, thereby rewriting problem descriptions through text augmentation. The Programmer module is responsible for generating and executing well-structured code to deliver the problem resolution. The Reviewer module equips the agent with the capacity for self-debugging and self-refinement through interactive feedback with code runtime outputs. By leveraging the end-to-end review mechanism, the executable code provided by the Programmer attains the iterative revision. A comprehensive evaluation is conducted on the performance of the proposed agent framework in solving partial differential equations (PDEs), ill-conditioned linear systems, and data-driven physical analysis problems. Compared to single-model, this collaborative framework significantly improves the bug-free code generation rate and reduces the occurrence of non-physical solutions, thereby establishing a highly reliable framework for autonomous code generation based on natural language descriptions. The review mechanism improved the average execution success (bug-free code and non-NaN solutions) rate of the modern reasoning models, DeepSeek R1, from 59% to 82%, while ChatGPT 4.1-mini improved from 66% to 87% and Gemini-2.5 improved from 60% to 84%. In summary, our agent framework establishes automatic code generation and review as a promising scientific computing paradigm. Our code is available at `https://github.com/ChengAo21/Re4_Sci_Agent`.

1 INTRODUCTION

Scientific computing, as a cornerstone of modern science and engineering, enables the modeling, simulation, and analysis of complex physical phenomena, including fluid dynamics (Brunton et al., 2020), computational mechanics (Zhang et al., 2022), electromagnetism (Oskooi et al., 2010), material science (Ramprasad et al., 2017) and quantum computing (Meng & Yang, 2023). However, solving scientific computing problems typically requires profound domain expertise, sophisticated algorithm design, and rigorous code implementation. In recent years, large language models (LLMs) (Zhao et al., 2023) with advanced reasoning capabilities have emerged as transformative and low-barrier tools in scientific computing: automatically generating code based on a small amount of vague natural language descriptions. Implementing a natural-language-based solving paradigm faces two major challenges: (1) how to autonomously select and implement appropriate methods for specific

*Corresponding author: zhanglei@imech.ac.cn
†Corresponding author: hgw@lnm.imech.ac.cn

problems; and (2) how to generate bug-free code. The first challenge demands the mastery and application of extensive domain-specific knowledge, necessitating that the LLMs complete the entire reasoning chain-from determining the type of equation (e.g., elliptic, parabolic, hyperbolic) to selecting the appropriate numerical method-without human intervention. The second challenge is a critical prerequisite for achieving autonomous scientific computing, requiring the model to translate vague natural language into precise code language.

Since the advent of LLMs like ChatGPT (Achiam et al., 2023), researchers have begun to explore generating code through natural language descriptions (Chen et al., 2021; Wang et al., 2023; Roziere et al., 2023). Early models heavily relied on meticulous prompt engineering (Wei et al., 2022; Kojima et al., 2022), requiring manual specification of numerical methods and model parameters, often resulting in logical and syntactical errors. At that stage, LLMs essentially functioned merely as code generation tools, providing rough program templates based on detailed customization from researchers, with low autonomy and necessitating extensive modification and debugging by users(Jiang et al., 2024). Recently, the excellent reasoning capabilities of reasoning LLMs have shown promise in addressing autonomy issues. Studies by Jiang et al. (Jiang et al., 2025) and Wang et al. (Wang et al., 2025) have tested the capabilities of these reasoning models in scientific computing, indicating that they generally outperform non-reasoning models. However, it was found that even when addressing classical problems with established solutions, the code generated by reasoning models frequently contained errors requiring human correction. Furthermore, in tests conducted by Wang et al., reasoning models such as DeepSeek R1 (Guo et al., 2025) and OpenAI's o3-mini-high (OpenAI, 2025b) proved ineffective in solving challenging problems like the Hilbert linear algebra problem, with DeepSeek R1 proposing potential strategies during thinking but failing to implement them. In addition to low bug-free rates in code generation, reasoning models currently face issues such as strong randomness in the selection of numerical methods, unstable output results, and reasoning hallucinations. These factors can lead to the selection of suboptimal numerical schemes, thereby significantly affecting the reliability of LLMs in scientific computing.

The agent framework based on LLMs provides the capability to interact with external systems. Prompt-engineering-based interactive feedback enables self-refinement without the need for extra training. Madaan et al. (Madaan et al., 2023) leveraged a single non-reasoning LLM to generate an initial output, provided self-feedback, and iteratively optimized subsequent outputs using historical feedback and prior iterative outputs. Such a self-feedback framework, by simulating the iterative process of "draft-reflection-revision" of humans, significantly improves the quality of outputs compared with the traditional one-step generation approach. By encoding Standardized Operating Procedures (SOPs) into prompt sequences, the multi-agent collaboration framework MetaGPT (Hong et al., 2023) empowered LLMs to simulate human professional roles, delivered structured outputs, and collaborated in an assembly-line manner, thereby reducing errors and boosting task efficiency. By assigning unique roles to multiple LLMs, the ChatEval framework constructed a multi-agent referee team that enhanced text evaluation accuracy through collaborative debate, and experimental results underscore the strengths of multi-agent collaboration (Chan et al., 2023). In the field of scientific computing, such as computational fluid dynamics, an LLM-driven framework enabled end-to-end output from natural language input to user-interactive wall-bounded turbulence modeling problems (Yang et al., 2025). Xu et al. (Xu et al., 2025) proposed a zero-shot, natural language-driven multi-agent system called CFDagent, which, coordinated by LLMs, completes the end-to-end process from natural language prompts to fully automated computational fluid dynamics (CFD) simulations. For solving partial differential equations (PDEs), the PINNsAgent (Wuwu et al., 2025), which integrates physics-informed knowledge replay and memory tree reasoning strategies, serves as an LLM-based surrogate framework to automatically construct and optimize physics-informed neural networks (PINNs) architectures, effectively bridging the gap between domain-specific knowledge and deep learning expertise. The task-specific-tuning-free framework CodePDE was validated on a range of representative PDE problems by enabling LLMs to perform iterative debugging (Li et al., 2025). The best solutions achieved by this framework on multiple PDE problems reach or exceed human levels, indicating the potential of LLM-based agents in solving PDEs. However, existing applications of LLMs in scientific computing often lack a structured framework to ensure the designed algorithms adaptable to diverse tasks and code refinement through feedback that interacts with runtime results.

To address the autonomy and reliability issues encountered by LLMs in achieving solutions autonomously, while leveraging the merits of multi-step multi-agent collaboration, we propose a novel agent framework for scientific computing that integrates three collaborative modules: Consultant,

Reviewer, and Programmer, to automate the end-to-end process of problem solving. This framework is designed to incorporate:

• **Rewriting**. By expanding the task background through adaptive integration of domain knowledge and providing algorithmic strategies, LLMs' understanding of specific scientific computing problems is deepened.

• **Resolution**. In the initial response, Python code are generated leveraging the augmented task text and run in the terminal, while capturing the runtime output text for subsequent refinement.

• **Review**. As an independent third party, an independent LLM evaluates the solution quality of the designed algorithm, with suggestions for the code's implementation details, thereby endowing the agent with the traits of self-debugging and self-refinement.

• **Revision**. In the revision loop, the executable code are comprehensively enhanced in execution/solving success rate, readability, modularity, and solution accuracy through feedback-driven refinement.

We quantitatively assess the performance of both the agent with and without the Reviewer module in autonomously solving PDEs, challenging Hilbert systems and conducting data-driven physical analysis. This assessment encompasses various dimensions, including bug-free rate, occurrence of non-physical solutions, high-precision solution generation rate (with errors below a certain threshold), and solution accuracy. Through multi-model cross-validation and the implementation of self-debugging, the review mechanism improved the average execution success (bug-free code and non-NaN solutions) rate of the modern reasoning models: DeepSeek R1 improved from 59% to 82%, ChatGPT 4.1-mini improved from 66% to 87% and Gemini-2.5 improved from 60% to 84%. Additionally, this framework significantly increased the probability of selecting high-precision methods, resulting in a noticeable improvement in average solution accuracy compared to single models. This agent framework demonstrates generality and versatility; we have extended it to data-driven analyses of governing physical relationships, and results indicate its capacity to consistently produce correct analytical outcomes.

	Reasoning	Expansion	Debugging	Refinement	Review	Single/Multiple LLMs
Non-reasoning LLMs	×	×	×	×	×	Single
Reasoning LLMs	✓	×	×	×	×	Single
CodePDE Agent	✓	×	✓	based on posterior error	×	Single
PINNsAgent	✓	×	✓	✓	×	Single
Madaan's Agent	×	×	×	✓	×	Single
Our Agent	✓	✓	✓	✓	✓	Multiple

Table 1: Comparison among existing LLM-based agent frameworks for scientific computing and the one we proposed.

A detailed comparison of current LLM-based agents' capabilities is presented in Table 1. Our contributions are as follows:

(1) We introduce a novel agent framework for scientific computing that incorporates a "rewriting-resolution-review-revision" logical chain, significantly improving the bug-free code generation rate and reducing the occurrence of non-physical solutions, thereby establishing a highly reliable framework for autonomous code generation based on natural language descriptions.

(2) We present a robust multi-LLMs collaborative framework for scientific computing that outperforms single models across all performance metrics.

(3) We apply the agent framework to the analysis of governing physical mechanisms, validating its generality and versatility.

The rest of the paper is organized as follows: the construction of the agent framework is introduced in Section 2; the comprehensive validation on different scientific computing problems is presented in Section 3; and the conclusions are drawn in Section 4.

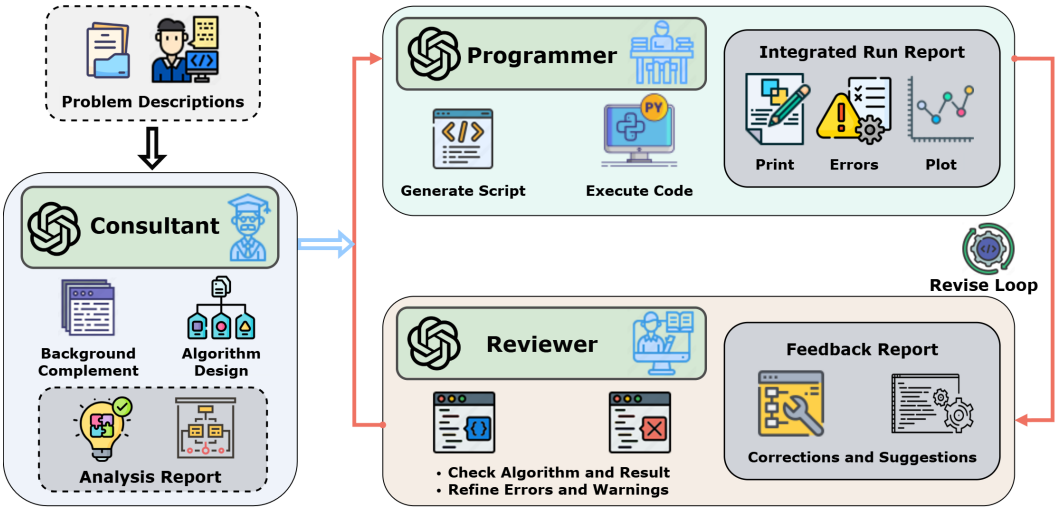


Figure 1: Schematic of the agent framework.

2 METHODOLOGY

This section details the overall framework of the agent and the responsibilities of each module. As depicted in figure 1, the agent is orchestrated via LangGraph (AI, 2025), where each module is defined as a functional node, and the "rewriting-resolution-review-revision" workflow is orchestrated through conditional edges. To ensure operational reliability, each module employs structured output protocols (leveraging Pydantic-validated JSON schemas) to strictly separate internal reasoning from functional data. The collaborative framework encompasses three main modules:

(i) **Consultant module.** Functioning as a mathematical consultant and numerical analyst, this node expands the context of the original problem statement. Its primary goal is to dissect vague natural language descriptions to identify underlying mathematical and numerical challenges. Furthermore, it generates a structured report that provides an expanded contextualization and a variety of alternative solution strategies (e.g., pseudocode or structured plans), thereby deepening the task understanding through domain-specific text augmentation.

(ii) **Programmer module.** Acting as an expert Python programmer, this node translates the consultant's analysis into a well-structured, modular, and executable Python script. The Programmer receives different context based on the operational phase: during the resolution phase, it ingests the consultant's augmented context, whereas during the revision loop, it focuses on error traces and Reviewer recommendations. Adhering to structured output protocols, the module separately generates a concise architectural description of the core algorithm and data flow, and a standardized Python script block.

(iii) **Reviewer module.** Serving as a code reviewer and scientific computing expert, this node assesses the reliability of numerical results and the quality of code implementation. Governed by an LLM independent of the Programmer module, it evaluates inputs comprising the consultant's augmented context, the programmer's code, and the integrated run report (containing stdout, warnings, and errors). Its objectives include: (a) determining whether the selected algorithm is appropriate and perfectly solves the problem; (b) assisting in debugging runtime errors and compiler warnings; (c) providing suggestions for algorithmic and code optimization. The Programmer and Reviewer form a feedback loop, enabling the agent's self-debugging and self-refinement.

Beyond a single-LLM loop, our framework supports a collaborative multi-model architecture, allowing distinct LLMs to operate within different functional nodes to leverage their respective strengths.

Given the great potential of reasoning LLMs(Wang et al., 2025; Jiang et al., 2025), we primarily evaluate the agent's performance of **GPT-4.1-mini** (OpenAI, 2025a), **Gemini-2.5-flash-preview** (DeepMind, 2025), and **Deepseek-R1** (DeepSeek-AI, 2025) **as Programmers**. We designate **GPT-4.1-mini**

as both the Consultant and Reviewer due to its high response efficiency and 1-million-token context window, which is essential for managing large datasets or extensive execution logs.

To manage context window constraints, we implement a selective context preservation strategy for terminal outputs. The system retains only the beginning and end segments of the runtime logs (e.g., 800 characters each), substituting the intermediate redundant content with a descriptive prompt marker (e.g., "*[Truncated: Content too long...]*"). By filtering out repetitive intermediate data, this strategy ensures that the agent preserves the most critical information (e.g., the error origin and final convergence state), thereby preventing reasoning failures caused by excessively long prompts.

The key innovation of the proposed agent lies in its collaborative framework of multiple LLMs, structured based on the logical chain of "rewriting-resolution-review-revision":

(i) Leveraging the Consultant module to transform vague natural language into a structured algorithmic formulation, linking general problems to specific domain insights through text augmentation;

(ii) Introducing the Reviewer module, which generates detailed feedback through interaction with code execution outputs, endowing the agent with self-debugging and self-refinement capabilities.

3 EXPERIMENTS

Tasks. In this section, we conduct a comprehensive performance evaluation of the proposed agent architecture across diverse specialized scenarios in scientific computing, with explicit focus on PDE benchmarks (including six specific equations), ill-conditioned Hilbert linear systems, and data-driven physical analysis based on dimensional analysis. In the first two tasks, the agent is required to solve given PDEs or linear algebraic systems. In the last task, the agent autonomously analyzes the dominant dimensionless quantities from an infinite set of possibilities based on available experimental data.

Models. We evaluate the newest and most popular LLMs as the Programmer, including GPT-4.1-mini, Gemini-2.5-flash-preview, and Deepseek-R1.

Metrics. The capabilities of generating high-quality code and solutions, compared to the original LLM without feedback from the Reviewer module, are measured by:

• **Code execution success rate.** The percentage of generating bug-free code and providing non-NaN solutions. This metric indicates the reliability of the code generation.

• **Solving success rate.** The percentage of errors below a specified threshold, particularly in the task of solving ill-conditioned Hilbert linear systems. For the data-driven physical analysis task, this refers to the percentage of successfully identifying dominant dimensionless quantities.

• **Accuracy.** The L^2-norm or L^∞-norm relative error.

3.1 PARTIAL DIFFERENTIAL EQUATION BENCHMARK

In this subsection, we assess the performance of the proposed agent framework in designing algorithms to solve a set of PDE problems that represent a wide range of physical challenges. The benchmark encompasses three categories: **discontinuous problems** (Burgers equation, Sod shock tube), **elliptic equations** (Poisson equation, Helmholtz equation), and **Navier-Stokes (NS) equations** (Lid-driven cavity, Unsteady NS).

Detailed mathematical definitions, inherent challenges (e.g., nonlinearity, complex geometry, unsteady evolution), and reference solutions for each equation are provided in **Appendix B.1**. To evaluate the deviation between the agent's answers and reference solutions, we adopt the relative L^2 error metric to rigorously quantify the global fidelity of the solutions, defined as follows:

$$error = \sqrt{\frac{\sum_{i=1}^{n}(y_i' - y_i)^2}{\sum_{i=1}^{n}(y_i)^2}} \tag{1}$$

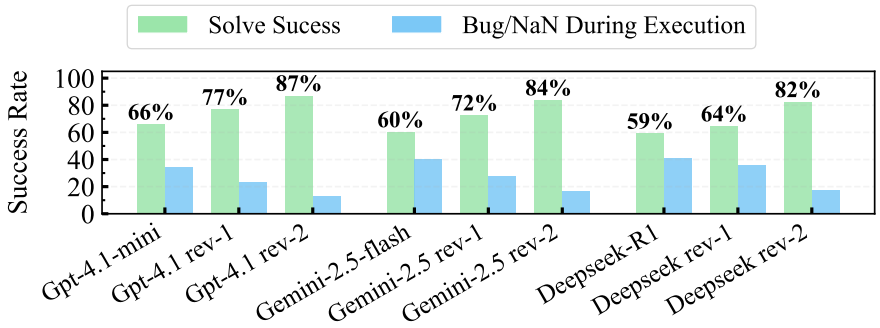


Figure 2: The overall average execution success rate of numerical algorithms employed by Programmers across all equations in the PDEbench.

Programmer	GPT-4.1-mini			Gemini-2.5-flash			Deepseek-R1		
	ans-0	rev-1	rev-2	ans-0	rev-1	rev-2	ans-0	rev-1	rev-2
Burgers	5.8e-02	2.5e-02	**2.3e-02**	4.1e-02	3.5e-02	3.1e-02	7.0e-02	4.1e-02	3.0e-02
Sod Shock	1.3e-01	6.0e-02	6.1e-02	6.4e-02	6.4e-02	7.0e-02	1.7e-01	**4.4e-02**	4.6e-02
Poisson	4.4e-02	2.6e-02	1.9e-02	3.5e-02	2.4e-02	**1.5e-02**	4.1e-02	2.6e-02	1.5e-02
Helmholtz	4.9e-02	4.5e-02	3.7e-02	3.4e-02	3.2e-02	**2.6e-02**	4.5e-02	3.5e-02	2.8e-02
Lid-Driven	4.2e-01	1.4e-01	**5.7e-02**	2.7e-01	2.3e-01	9.9e-02	3.0e-01	2.6e-01	1.8e-01
Unsteady NS	2.9e-01	2.3e-01	1.9e-01	2.1e-02	2.1e-02	2.1e-02	2.4e-01	9.0e-02	**2.0e-02**

Table 2: The summary of average L^2 relative error across all cases in PDEbench, where underlined entries denote each Programmer's minimum errors at different response stages, and bold entries indicate the overall minimum errors among all Programmers.

where $\boldsymbol{y} = (y_i)_{i=1}^n$ is the ground truth and $\boldsymbol{y'} = (y'_i)_{i=1}^n$ is the calculated results, and n is the reference solution's grid point count.

The overall average execution success rate of the code generated by the agent framework across all PDE problems is presented in figure 2. Comparing the initial response (ans-0) against the final output (rev-2) serves as an ablation study for the proposed Reviewer module. It is evident that with the involvement of the Reviewer, the Programmers effectively perform self-debugging and self-refinement, leading to up to 24% improvement in code execution success rate.

Furthermore, the framework demonstrates a marked and consistent refinement in numerical accuracy. As evidenced by the optimal runtime outputs (Best-of-n perspective) summarized in Appendix B.2 (Table 4), the L^2-norm relative errors for all three Programmers exhibit a robust and monotonic downward trend through the "rewriting-resolution-review-revision" workflow. Such advancement underscores the agent's capacity to not only correct syntax errors but also progressively steer the implementation toward more sophisticated and reliable numerical algorithms.

The statistical significance of this improvement is further corroborated by the average L^2-norm relative errors metrics in Table 2 and the error distribution visualized in Appendix B.2 (figure 7). The boxplots reveal that the Reviewer not only lowers the median error but also tightens the interquartile range. This contraction of the error distribution indicates that the Reviewer effectively mitigates outliers, such as unreasonable or non-physical solution attempts, and promotes consistent convergence toward high-precision outcomes across diverse initial attempts.

To illustrate the agent's capability in handling complex physical problems, we conduct an in-depth analysis of the **unsteady 2D Navier-Stokes equations** (Appendix B.3), representing the most challenging case in the benchmark. This coupled system (u, v, p) requires sophisticated treatment to maintain long-term stability. Notably, the Reviewer steers the Programmer toward critical algorithmic upgrades, including **enhanced boundary condition treatments, higher-order finite-difference discretization, ILU-preconditioned solver strategies, and dynamic CFL conditions**. These refinements ensure numerical stability and enable the accurate capture of transient phenomena (Figures 10-11). Detailed formulations and prompts for other PDE cases are provided in Appendix B.4.

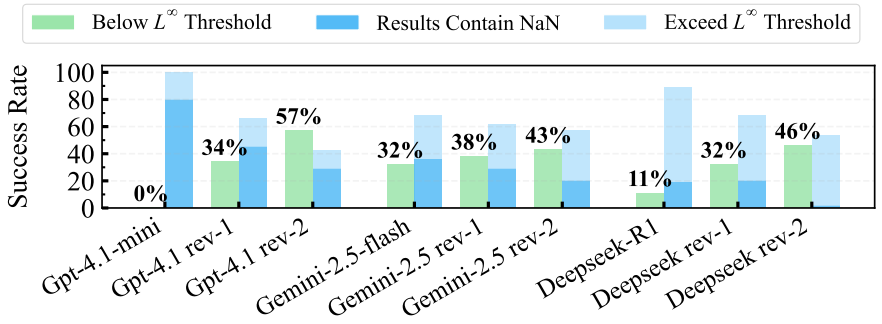


Figure 3: Proportional distribution of executable code provided by Programmers across three different completion statuses.

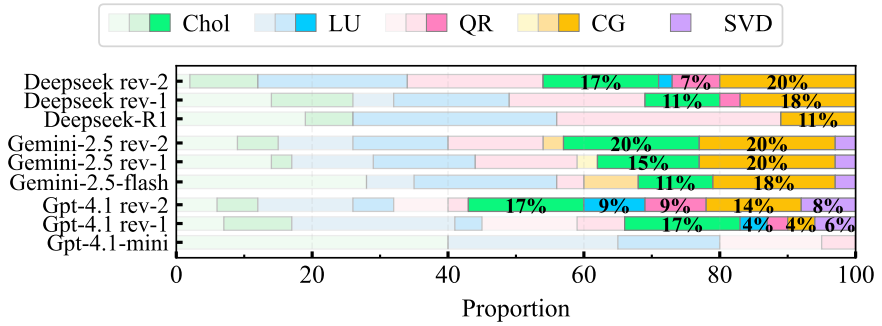


Figure 4: Proportional distribution of completion status for various methods at different response stages. A single dark-colored patch represents singular value decomposition (SVD) method results consistently below the threshold. The color patches corresponding to other methods are divided into three segments, with colors darkening gradually to indicate the following respectively: results contain NaN, exceed L^∞ threshold and below L^∞ threshold.

3.2 HILBERT LINEAR ALGEBRAIC SYSTEMS

In this subsection, we assess the performance of the proposed agent framework in solving the Hilbert linear algebraic system (Kress, 1998). Hilbert matrices serve as prototypical ill-conditioned problems where the condition number grows exponentially with the dimension n, rendering naive numerical methods ineffective. The detailed mathematical formulation, condition number analysis, and problem description prompt are provided in Appendix C.1.

Specifically, the evaluation spans dimensions $n \in \{5, 10, \ldots, 25\}$, utilizing the L^∞ error against the exact solution $\mathbf{x}^* = (1, \ldots, 1)^\top$ as the metric. To bypass standard high-level APIs (e.g., `numpy.linalg.solve`) and rigorously assess the generation of robust algorithms, the problem description explicitly mandates implementing methods from scratch. Based on statistical analysis of eight independent runs, we categorize the completion status of executable code into three types: **results contain NaN, exceed L^∞ threshold, and below L^∞ threshold**, with the threshold set to 10^{-2}.

As shown in figure 3, GPT-4.1-mini failed to provide any solution with errors below the threshold in its initial response, i.e., the original solving success rate is zero. However, with the guidance of the Reviewer, all three Programmers provided more numerically precise and stable algorithms. Specifically, in terms of the enhancement in solving success rate driven by review mechanism, GPT-4.1-mini improved from 0% to 57%, while Gemini-2.5-flash improved from 32% to 43% and Deepseek-R1 improved from 11% to 46%. Figure 4 presents the percentage contribution of various methods to the solution success rate, with Cholesky and Conjugate Gradient accounting for the major share. The marked increase in the darkest color patch (below L^∞ threshold) fully demonstrates the refined stability and accuracy of solution schemes with the Reviewer's suggestions.

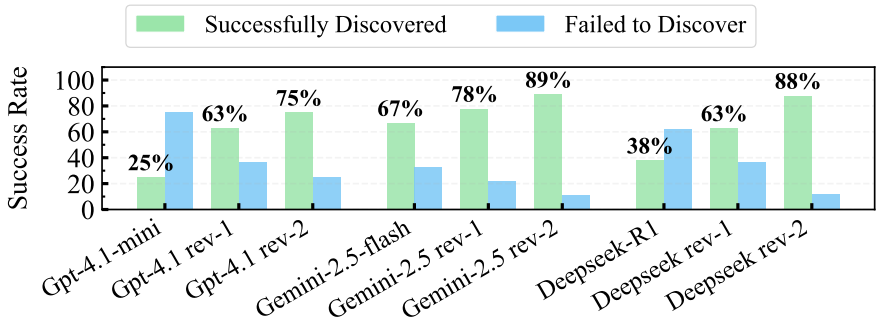


Figure 5: The success rate of Programmers' search algorithms in identifying dominant dimensionless quantities Ke.

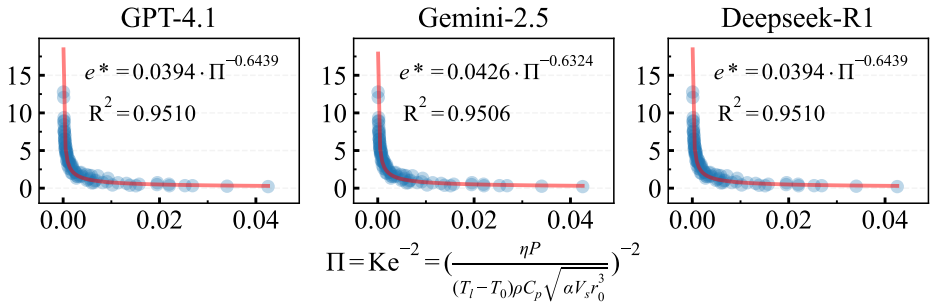


Figure 6: Fitting curve of the dimensionless number Π as a function of e* derived by all programmers in final response (review-2).

To more intuitively illustrate the adaptability of various approaches underlying the success rate in solving Hilbert ill-conditioned matrices, figure 31 in Appendix C.2 presents the proportional distribution of specific methods across three completion statuses. The improvement observed from the above statistical perspective stems from the Reviewer's intervention in steering the agent toward regularization techniques (e.g., Tikhonov regularization (Golub et al., 1999)) or iterative methods (e.g., conjugate gradient (Hestenes et al., 1952)), which represents the key insight for stably and accurately solving ill-conditioned Hilbert matrices. Fundamentally, the reason is that the agent framework recognizes that ill-conditioning restricts the effectiveness of naive algorithms, thus necessitating sophisticated algorithmic enhancements rather than trivial adjustments.

Table 5 in Appendix C.2 summarizes the results of various approaches provided by each Programmer after 2 rounds of Reviewer intervention. From the best-of-n sample perspective, the superior performance of the agent framework's self-refinement mechanism in solving ill-conditioned matrix problems is amply demonstrated.

3.3 Data-driven physical analysis based on dimensional analysis

In this subsection, we assess the performance of the proposed agent framework in the data-driven physical analysis based on the dimensional analysis. The representative problem of modeling keyhole dynamics during laser-metal interaction is selected for this analysis. The detailed physical background, variable definitions, and the experimental dataset are provided in Appendix D, along with the specific problem description prompt.

The specific task requires the agent to identify the dominant dimensionless number governing the keyhole aspect ratio e^* while adhering to physical constraints such as dimensional homogeneity and rational exponents. The ground truth solution is the well-established Keyhole number (Ke) (Gan et al., 2021) (see Eq. 4 for the mathematical formulation). Consequently, we evaluate the agent based on its success rate in correctly reconstructing the Ke form from the raw experimental data.

In contrast to the crude generation of dimensionless quantities directly via Buckingham's Π theorem, the Programmers consistently adopt grid search and linear regression methods. The specific differences in the implementation of executable code stem from the compatibility between dimensional homogeneity and exponent constraints, as well as the robustness of data extraction.

The solving success rate, obtained by statistically analyzing the agent's results from multiple attempts to solve this problem, is shown in figure 5. Statistically, it can be intuitively observed that as the response stage advances, the success rate of discovering the dimensionless number Ke increases by up to 50%. This improvement arises because, with the involvement of Reviewer, the Programmers revised the implementation details of exponent constraints and ensured strict compliance with dimensional homogeneity.

Specifically, deficiencies in the implementation logic of the triple exponent constraints could undermine dimensional homogeneity, leading to search results with high coefficients of determination (R^2) but physically meaningless; on a more essential level, the agent framework realized that R^2 can only measure the goodness of fit for curves but is insufficient to guarantee the dimensional homogeneity of dimensionless numbers.

The fitting curve of the dimensionless number Ke as a function of the keyhole aspect ratio e* is shown in figure 6. The high R^2 value calculated on a log-log scale further validates the excellent performance of the agent framework in fully solving dimensional analysis problems.

4 Conclusions

In this work, we introduced a reliable agent framework for autonomous code generation in scientific computing without domain-specific training. We quantitatively assess its performance in solving PDEs, challenging ill-conditioned linear algebra systems, and conducting data-driven physical analysis. By solving various problems with distinctly different backgrounds, the agent's capability to analyze problems and link them to specific domains through knowledge transfer is fully demonstrated; from both statistical and sampling results perspectives, the agent's ability to progressively perform self-debugging and self-refinement through interaction with runtime outputs is reflected. Therefore, with guidance from the Consultant and feedback from the Reviewer, the algorithms designed by the Programmers achieved significant improvements in performance and robustness, as measured by code execution success rates, solving success rates, and overall accuracy.

This agent framework provides a collaborative platform for multiple LLMs, overcoming the reasoning limitations of single models. The core modules, Consultant, Programmer, and Reviewer, support heterogeneous model combinations (such as ChatGPT, DeepSeek, and Gemini), and the Reviewer module allows for parallel evaluation by multiple models. Multi-model collaboration can mitigate hallucination issues arising from design flaws in a single model. For instance, in solving the SOD shock tube problem, Gemini 2.5-flash consistently "stubbornly" selected the forward Euler scheme; however, after review by ChatGPT 4.1-mini, it shifted toward selecting high-precision time schemes.

Looking ahead, several directions exist to enhance the current agent. First, while the multi-agent collaboration significantly boosts reliability, it inevitably incurs higher token usage and time overhead compared to single-step generation. Future optimizations will explore adaptive inference strategies to balance computational cost with solution quality. Second, the current workflow relies heavily on the Consultant's initial strategy. If the Consultant proposes a flawed strategy, the Reviewer may struggle to steer the Programmer back on track. To address this, we plan to introduce a backtracking mechanism that allows the agent to revisit the Consultant phase upon persistent failure.

Beyond these workflow improvements, the Reviewer module's assessment of code design is currently based on general and abstract principles for scientific computing, highlighting the need for a more comprehensive, detailed, and quantifiable evaluation system. Additionally, the LLMs in the Reviewer module may encounter token limitations when handling long contexts (such as review comments combined with runtime logs), necessitating the development of information distillation algorithms or adaptations for long-context models. Furthermore, we observed that LLMs generate code with version lag issues (e.g., outdated Python syntax), indicating a need to update and optimize the LLMs themselves or integrate them with domain knowledge bases.

As LLMs' reasoning capabilities continue to evolve, the performance of this agent framework is expected to improve further. Finally, to further broaden the framework's applicability to complex engineering problems or tasks involving industrial software like OpenFOAM, future work will focus on augmenting each module with dynamic knowledge retrieval, enabling the agent to incorporate both domain-specific insights from technical literature and operational guidelines from software documentation.

In summary, the agent framework we proposed establishes automatic code generation and review as a promising paradigm in the field of scientific computing, thereby offering a new perspective for physically interpretable and clearly reasoning-grounded algorithm design.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (NSFC) Excellence Research Group Program for "Multiscale Problems in Nonlinear Mechanics" (No. 12588201). L. Zhang was also supported by the National Natural Science Foundation of China (NSFC) under Grant No. 12202451. The authors would like to thank Zhaobin Li for his suggestions on early topic selection for this paper.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

LangChain AI. Langgraph: Build llm applications with stateful graphs, 2025. URL `https://github.com/langchain-ai/langgraph`. Accessed: 2025-07-15.

Grigory Isaakovich Barenblatt. *Scaling*. Cambridge University Press, 2003. doi:10.1017/CBO9780511814921.

Edward R Benton and George W Platzman. A table of solutions of the one-dimensional burgers equation. *Quarterly of Applied Mathematics*, 30(2):195–212, 1972.

Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual review of fluid mechanics*, 52(1):477–508, 2020.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Alexandre Joel Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of computational physics*, 2(1):12–26, 1967.

Google DeepMind. Gemini-2.5-flash-preview-05-20, 2025. URL `https://ai.google.dev/gemini-api/docs/models#gemini-2.5-flash`. Accessed: 2025-07-10.

DeepSeek-AI. Deepseek-r1-0528, 2025. URL `https://api-docs.deepseek.com`. Accessed: 2025-07-10.

Remy Fabbro, Morgan Dal, Patrice Peyre, Frédéric Coste, Matthieu Schneider, and Valerie Gunenthiram. Analysis and possible estimation of keyhole depths evolution, using laser operating parameters and material properties. *Journal of Laser Applications*, 30(3), 2018.

Zhengtao Gan, Orion L Kafka, Niranjan Parab, Cang Zhao, Lichao Fang, Olle Heinonen, Tao Sun, and Wing Kam Liu. Universal scaling laws of keyhole stability and porosity in 3d printing of metals. *Nature communications*, 12(1):2379, 2021.

UKNG Ghia, Kirti N Ghia, and CT Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982.

Gene H Golub, Per Christian Hansen, and Dianne P O'Leary. Tikhonov regularization and total least squares. *SIAM journal on matrix analysis and applications*, 21(1):185–194, 1999.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2023.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

Qile Jiang, Zhiwei Gao, and George Em Karniadakis. Deepseek vs. chatgpt vs. claude: a comparative study for scientific computing and scientific machine learning tasks. *Theoretical and Applied Mechanics Letters*, 15(3):100583, 2025. doi:10.1016/j.taml.2025.100583.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Rainer Kress. Ill-conditioned linear systems. In *Numerical Analysis*, pp. 77–92. Springer, 1998.

Shanda Li, Tanya Marwah, Junhong Shen, Weiwei Sun, Andrej Risteski, Yiming Yang, and Ameet Talwalkar. Codepde: An inference framework for llm-driven pde solver generation. *arXiv preprint arXiv:2505.08783*, 2025. doi:10.48550/arXiv.2505.08783.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Zhaoyuan Meng and Yue Yang. Quantum computing of fluid dynamics using the hydrodynamic schrödinger equation. *Physical Review Research*, 5(3):033182, 2023.

OpenAI. gpt-4.1-mini-2025-04-14, 2025a. URL `https://platform.openai.com/docs/models/gpt-4.1-mini`. Accessed: 2025-07-10.

OpenAI. o3-mini-2025-01-31, 2025b. URL `https://platform.openai.com/docs/models/o3-mini`. Accessed: 2025-03-10.

Ardavan F Oskooi, David Roundy, Mihai Ibanescu, Peter Bermel, John D Joannopoulos, and Steven G Johnson. Meep: A flexible free-software package for electromagnetic simulations by the fdtd method. *Computer Physics Communications*, 181(3):687–702, 2010.

Rampi Ramprasad, Rohit Batra, Ghanshyam Pilania, Arun Mannodi-Kanakkithodi, and Chiho Kim. Machine learning in materials informatics: recent applications and prospects. *npj Computational Materials*, 3(1):54, 2017.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

Gary A Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of computational physics*, 27(1):1–31, 1978.

Walter A Strauss. *Partial differential equations: An introduction*. John Wiley & Sons, 2007.

Long Wang, Lei Zhang, and Guowei He. Evaluations of large language models in computational fluid dynamics: Leveraging, learning and creating knowledge. *Theoretical and Applied Mechanics Letters*, pp. 100597, 2025. doi:10.1016/j.taml.2025.100597.

Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Qingpo Wuwu, Chonghan Gao, Tianyu Chen, Yihang Huang, Yuekai Zhang, Jianing Wang, Jianxin Li, Haoyi Zhou, and Shanghang Zhang. Pinnsagent: Automated pde surrogation with large language models. *arXiv preprint arXiv:2501.12053*, 2025.

Xiaoyu Xie, Arash Samaei, Jiachen Guo, Wing Kam Liu, and Zhengtao Gan. Data-driven discovery of dimensionless numbers and governing laws from scarce measurements. *Nature communications*, 13(1):7562, 2022.

Zhaoyue Xu, Long Wang, Chunyu Wang, Yixin Chen, Qingyong Luo, Hua-Dong Yao, Shizhao Wang, and Guowei He. Cfdagent: A language-guided, zero-shot multi-agent system for complex flow simulation. *arXiv preprint arXiv:2507.23693*, 2025.

Zhongxin Yang, Yuanwei Bin, Yipeng Shi, and Xiang IA Yang. Large language model driven development of turbulence models. *arXiv preprint arXiv:2505.01681*, 2025. doi:10.48550/arXiv.2505.01681.

Jianchao Ye, Saad A Khairallah, Alexander M Rubenchik, Michael F Crumb, Gabe Guss, Jim Belak, and Manyalibo J Matthews. Energy coupling mechanisms and scaling behavior associated with laser powder bed fusion additive manufacturing. *Advanced Engineering Materials*, 21(7):1900185, 2019.

Lei Zhang, Ye Lu, Shaoqiang Tang, and Wing Kam Liu. Hidenn-td: reduced-order hierarchical deep learning neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114414, 2022.

Cang Zhao, Qilin Guo, Xuxiao Li, Niranjan Parab, Kamel Fezzaa, Wenda Tan, Lianyi Chen, and Tao Sun. Bulk-explosion-induced metal spattering during laser processing. *Physical Review X*, 9(2): 021052, 2019.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Hao Zhongkai, Jiachen Yao, Chang Su, Hang Su, Ziao Wang, Fanzhi Lu, Zeyu Xia, Yichi Zhang, Songming Liu, Lu Lu, et al. Pinnacle: A comprehensive benchmark of physics-informed neural networks for solving pdes. *arXiv preprint arXiv:2306.08827*, 2023. doi:10.48550/arXiv.2306.08827.

A DETAILED CONSTRUCTION OF AGENT PROMPT TEMPLATE

Consultant module prompt

[Role] You are a distinguished Mathematical Consultant and Numerical Analyst.

(i) Provide deep technical insights into algorithm implementation, code optimization, and numerical solution strategies.

(ii) Use programming-specific terminology to analyze code structures, identify bugs, and offer advanced coding solutions.

[Task]
Expand the context of a given scientific problem and generate multiple alternative solution options.
[Goal]
Accurately dissect the problem by preserving its original meaning, explicitly identifying the underlying mathematical and numerical challenges, and formulating rigorous and well-structured solution strategies.

[Problem Statement]
$< Problem\ Description >$

[Output Requirement]
Provide an expanded contextualization of the original problem and multiple solution plans.

(i) Elaborate on the original problem outline while preserving its core meaning, and explicitly identify the primary mathematical and numerical challenges.

(ii) Provide a variety of alternative solution strategies, each presented in a structured formulation (e.g., concise pseudocode or numbered steps), rather than long continuous paragraphs.

Programmer module prompt (initial answer)

[Role] You are an expert Python Programmer specialized in scientific computing.

(i) Provide deep technical insights into algorithm implementation, code optimization, and numerical solution strategies.

(ii) Use programming-specific terminology to analyze code structures, identify bugs, and offer advanced coding solutions.

[Task]
For a specific problem description and the associated candidate solving algorithms, select an appropriate algorithm and implement it in Python to resolve the problem.
[Goal]
Produce a complete, well-structured, and executable Python script that correctly realizes the selected algorithm, with clear modular organization and quantitative outputs suitable for verification.

[Problem Statement]
$< Problem\ Description >$

[Consultant expansion]
$< Complete\ description\ of\ the\ problem >$
$< Descriptions\ of\ Multiple\ solution\ plans >$

[Output Requirement]
Based on the specific description of the original problem and multiple solving algorithms, select an appropriate method to resolve the problem.

(i) Provide a concise architectural description of the core algorithm and data flow.

(ii) Provide a complete, bug-free Python script.

The output code should be enclosed in a single “‘python…“ markdown block as follows:

```
# Technical explanation content

```python

# All required imports included at the top.
# Modular structure with clear function definitions.
# Use brief, clear code comments, avoid verbose or tutorial-style explanations.
# Must include quantitative outputs: printed metrics or labeled plots.

'''
```
```

Reviewer module prompt

[Role] You are a Code Reviewer and Scientific Computing Expert.

(i) Provide deep technical insights into algorithm implementation, code optimization, and numerical solution strategies.

(ii) Use programming-specific terminology to analyze code structures, identify bugs, and offer advanced coding solutions.

[Task]
Review the reliability of numerical results and the quality of code implementation, then provide comments.
[Goal]
Conduct a structured review assessment based on the problem description, submitted code, and runtime output.
Be pragmatic in your decision: if results are reasonable, **ACCEPT** it; but if you request a **REVISE**, you must be thorough and list all technical blockers.

[Problem Statement]
$< Problem\ Description >$

[Consultant expansion]
$< Complete\ description\ of\ the\ problem >$

[Programmer solution]
$< Code\ and\ Executing\ Results >$

[Output Requirement]
Analyze the results output by the code and provide detailed feedback, guide the programmer in further deepening his understanding of the problem and solving it with greater perfection. **Ensure your feedback includes:**

(i) Determine whether the programmer has perfectly solved the problem and whether the most appropriate algorithm has been used.

(ii) Assist the programmer in checking and refining runtime errors and warnings in the code.

(iii) Provide suggestions for optimizing the code, including but not limited to algorithm optimization, code structure optimization, and handling of potential errors in the code.

(iv) Your feedback includes posteriori issue identification based on programmer's results, and may also include a priori analysis based on your understanding of the specific problem.

Programmer module prompt (in revise loop)

[Role] You are an expert Python Programmer specialized in scientific computing.

(i) Provide deep technical insights into algorithm implementation, code optimization, and numerical solution strategies.

(ii) Use programming-specific terminology to analyze code structures, identify bugs, and offer advanced coding solutions.

[Task]
Revise the previously generated Python implementation based on the provided execution results and structured review comments.
[Goal]
Analyze the error trace to identify the underlying root cause of the previous failure, and address the recommendations from review comments exactly.

[Problem Statement]
< Problem Description >

[Previous solution by Programmer]
< Code and Executing Results >

[Reviewer comments]
< Advice to Algorithm and Debug >

[Output Requirement]
Generate revised Python code by incorporating reviewer's recommendations to improve solution plan's quality in terms of runtime, structure, and accuracy.

(i) Provide a concise architectural description of the core algorithm and data flow.

(ii) Provide a complete, bug-free Python script.

The output code should be enclosed in a single "'python…" markdown block as follows:

```
# Technical explanation content

```python

# All required imports included at the top.
# Modular structure with clear function definitions.
# Use brief, clear code comments, avoid verbose or tutorial-style explanations.
# Must include quantitative outputs: printed metrics or labeled plots.

'''
```

B PARTIAL DIFFERENTIAL EQUATION BENCHMARK DETAILS

B.1 BENCHMARK OVERVIEW

The PDEs selected for this study encompass a diverse range of mathematical properties, ensuring that the benchmark does not favor a specific type of equation. The curated set of problems introduces several fundamental challenges, outlined as follows:

- **Nonlinear Behavior:** When PDEs exhibit nonlinearity, minor deviations in initial conditions can lead to significant divergence in results.
- **Complex Geometry:** Irregular geometries pose direct challenges for the agent in accurately discretizing the domain and representing boundary behaviors.
- **Unsteady Evolution:** The solutions of certain PDEs change dynamically over time, requiring algorithms to accurately handle time-varying processes and robustly capture transient phenomena.

Table 3 provides a detailed overview of the selected PDEs, their inherent challenges, and the corresponding reference solutions. The reference datasets are collected from the Pinnacle benchmark (Zhongkai et al., 2023). The specific mathematical formulations and prompt templates for each equation are detailed in the subsequent subsections.
```

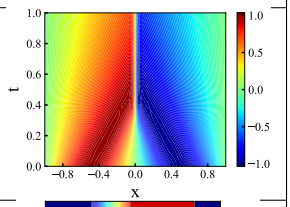
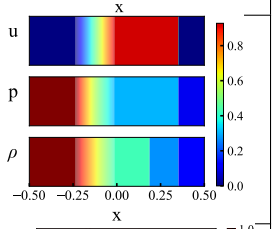
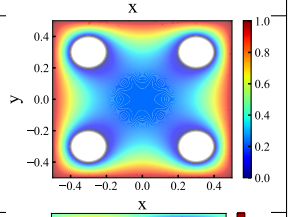
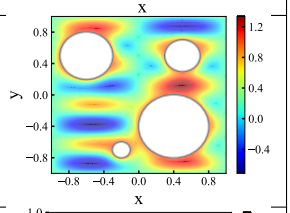
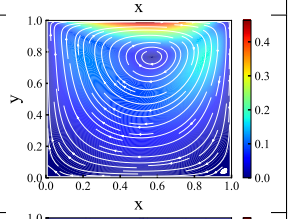
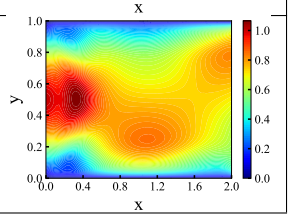
PDE systems	Challenges	Reference solutions
Burgers equation $\frac{\partial u}{\partial t} + uu_x - \nu u_{xx} = 0$	**Nonlinear,Unsteady** ✓ Accurately resolve oscillations and capture shock waves	
Sod Shock Tube $\begin{cases} \frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} = 0 \\ \frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2+p)}{\partial x} = 0 \\ \frac{\partial(\rho E)}{\partial t} + \frac{\partial(\rho E u+pu)}{\partial x} = 0 \end{cases}$	**Unsteady** ✓ Precisely capture time-dependent positions and strengths of multiple wave systems	
Poisson equation $\Delta u = 0$	**Complex Geometry** ✓ Precise discretization of curvilinear circular holes and rectangular edges	
Helmholtz equation $-\Delta u + k^2 u = f$	**Complex Geometry** ✓ Proper discretization of complex boundaries and handling of high-frequency oscillation	
Lid-Driven Cavity $\begin{cases} \nabla \cdot \mathbf{u} = 0 \\ \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p = \frac{1}{\mathrm{Re}} \Delta \mathbf{u} \end{cases}$	**Nonlinear** ✓ Complex flow details and velocity-pressure coupling	
Unsteady NS equation $\begin{cases} \nabla \cdot \mathbf{u} = 0 \\ \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p - \frac{1}{\mathrm{Re}} \Delta \mathbf{u} = \mathbf{f} \end{cases}$	**Nonlinear,Unsteady** ✓ Temporal evolution and velocity-pressure coupling	

Table 3: Summary of various equations in PDEbench.

B.2 SUPPLEMENTARY RESULTS

To supplement the main text, Table 4 summarizes the relative errors of each Programmer's optimal runtime output in the final response (review-2) against the reference solutions (Best-of-N perspective), with the reference physical quantities for comparison in the last column. Additionally, figure 7 provides the comprehensive boxplot of average L^2 relative errors for all equations. The boxes represent the interquartile range between the first and third quartiles, while the horizontal line in each box indicates the median. Additional error bars (or whiskers) depict the upper and lower limit values.

Programmer	GPT-4.1-mini	Gemini-2.5-flash	Deepseek-R1	Quantity
Burgers	1.98×10^{-2}	$\mathbf{1.82 \times 10^{-2}}$	1.85×10^{-2}	$u(x,t)$
Sod Shock	3.60×10^{-2}	5.81×10^{-2}	$\mathbf{2.31 \times 10^{-2}}$	$u(x,t)$
Poisson	9.90×10^{-3}	1.18×10^{-2}	$\mathbf{7.15 \times 10^{-3}}$	$u(x,y)$
Helmholtz	2.30×10^{-2}	2.09×10^{-2}	$\mathbf{1.47 \times 10^{-2}}$	$u(x,y)$
Lid-Driven	$\mathbf{2.74 \times 10^{-3}}$	6.55×10^{-3}	8.77×10^{-3}	$\|\vec{u}\|$
Unsteady NS	1.89×10^{-2}	1.83×10^{-2}	$\mathbf{1.81 \times 10^{-2}}$	$u(x,y)$

Table 4: The summary of L^2 relative error for the final refined runtime outputs after two rounds of Reviewer intervention (review-2) across all cases in PDEbench.

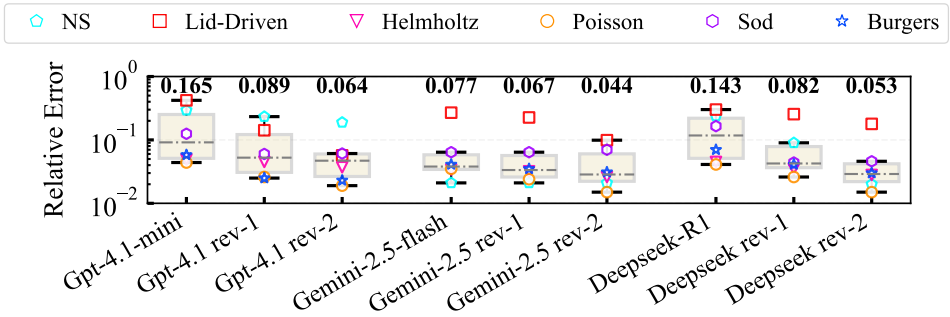


Figure 7: Boxplot of the average L^2 relative errors for all equations in the PDEbench. The **annotated numbers** in the figure denote the mean value of average relative errors.

B.3 CASE STUDY: UNSTEADY NS EQUATION

The long-time evolved unsteady NS equation is selected as a representative case study to illustrate the capabilities of our agent framework. The reason for choosing the unsteady NS equation stems from the fact that its difficulties arise in multiple aspects:

(i) **Coupled System:** The equations are a coupled system of PDEs, including the momentum equation and the continuity equation, which require simultaneous solution of the velocity (u, v) and pressure (p) (Chorin, 1967).

(ii) **Unsteady Evolution:** The equations are unsteady in time, necessitating the handling of time-varying source terms and left inlet velocity boundary conditions, both of which are time-dependent, as well as the consideration of time integration stability.

(iii) **Numerical Complexity:** The solution process typically employs the generalized minimal residual (GMRES) iterative method (Saad & Schultz, 1986) for the pressure Poisson equation, presenting numerical challenges from the ill-conditioned discrete linear system and the computational cost of achieving convergence, both of which arise from the global continuity constraint in incompressible NS simulations.

Developing numerical algorithms involves discretizing governing equations, properly handling boundary conditions, and implementing iterative solution schemes. Addressing these requirements necessitates robust expertise in numerical methods and programming, while agents must balance accuracy, stability, and computational efficiency during the solution process. These considerations fully highlight the complexities and challenges inherent in solving this unsteady NS equation.

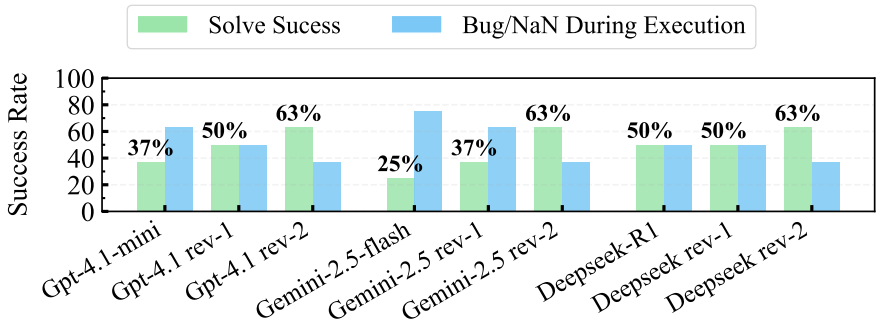


Figure 8: Execution success rate of the numerical algorithms employed by Programmers to solve the unsteady NS equation.

The specific equations and initial/boundary conditions are detailed in the following prompt template:

Problem Description Prompt: Unsteady Incompressible NS equation

The governing PDEs for 2D unsteady incompressible NS equations with explicit forcing are given by:

$$\begin{cases} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f_x, (x,y) \in \Omega, \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{\mathrm{Re}}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = f_y, (x,y) \in \Omega, \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, (x,y) \in \Omega, \end{cases}$$

The Reynolds number Re = 100. The domain is $\Omega = [0,2] \times [0,1]$, and the forcing term $f(x,y,t)$ is given by:

$$f_x = 0, \quad f_y = -\sin(\pi x)\sin(\pi y)\sin(\pi t).$$

The boundary conditions are:

$$\begin{cases} (u,v) = (0,0), \text{ on Top/Bottom walls } (y = 0,1) \\ u(0,y,t) = \sin(\pi y)\left(\sin(\pi t) + \sin(3\pi t) + \sin(5\pi t)\right), v = 0, \text{ at Inlet } (x = 0) \\ \frac{\partial u}{\partial x} = \frac{\partial v}{\partial x} = 0, p(2,y,t) = 0, \text{ at Outlet } (x = 2) \end{cases}$$

The initial condition is: $u(x,y,0) = v(x,y,0) = 0, (x,y) \in \Omega$

Implement a stable and efficient method to solve this problem.
Simulate until t=0.5. Plot contours of u, v, and p at the final step in one figure.

The statistical results of each Programmer repeatedly answering this problem are shown in Figures 8 and 9 including the code execution success rate and streamwise velocity relative error boxplot from the statistical perspective. Notably, the Reviewer's intervention has resulted in up to a 40% surge in the success rate of code execution following self-debugging. Meanwhile, the relative error shows a noticeable reduction after iterative self-refinement, illustrating how step-by-step improvements enhance code quality.

To provide a comprehensive demonstration of Programmer R1's code quality improvement, figure 10 illustrates the contours of streamwise velocity u, normal velocity v, and pressure p in the flow field generated by Deepseek-R1 after successive guidance from the Reviewer. The results generated during the initial run show larger deviations from the reference across all three physical quantities. It is worth noting that these performance improvements mainly stem from algorithmic-level refinements and optimizations in numerical implementation, rather than merely from the correction of runtime bugs. Specifically, the Reviewer's intervention prompted the Programmers to implement the following improvements: enhancing boundary condition treatment schemes, elevating the order of accuracy in finite-difference discretization, optimizing solver strategies (by incorporating incomplete LU (ILU) preconditioning), and strengthening numerical stability control (by employing a dynamic Courant-Friedrichs-Lewy (CFL) condition to determine the time-stepping).

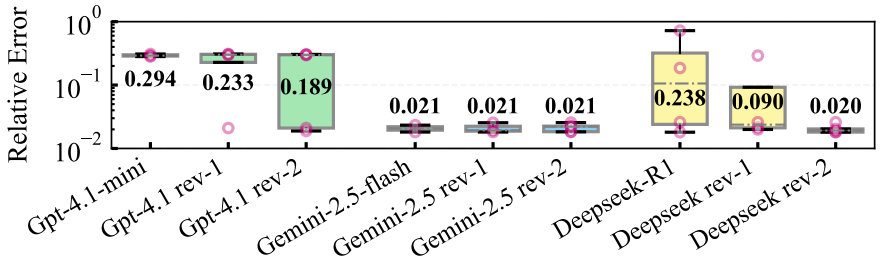


Figure 9: Boxplot of Relative L^2 error for successful solutions by Programmers.The **annotated numbers** in the figure indicate the mean relative L^2 errors.

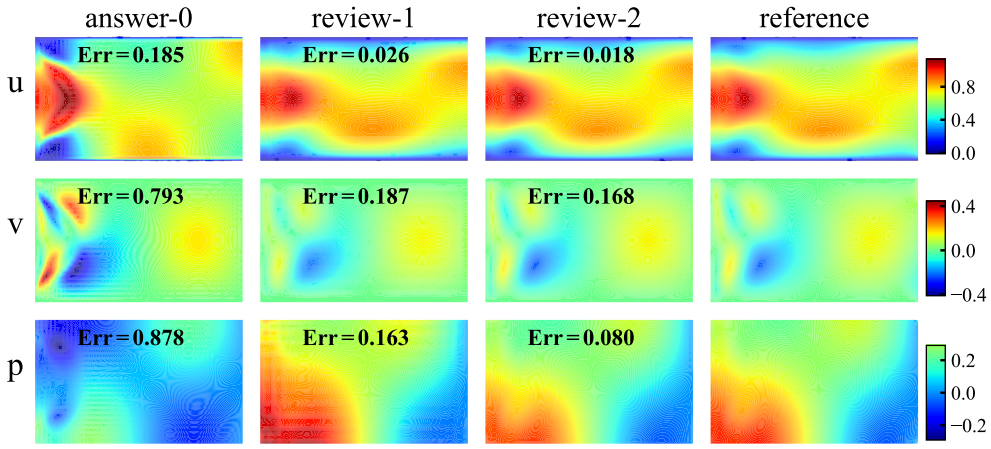


Figure 10: The runtime outputs of the executable code provided by Programmer Deepseek-R1 in the initial response (answer-0), together with those in the Reviewer's first (review-1) and second (review-2) interventions, respectively.

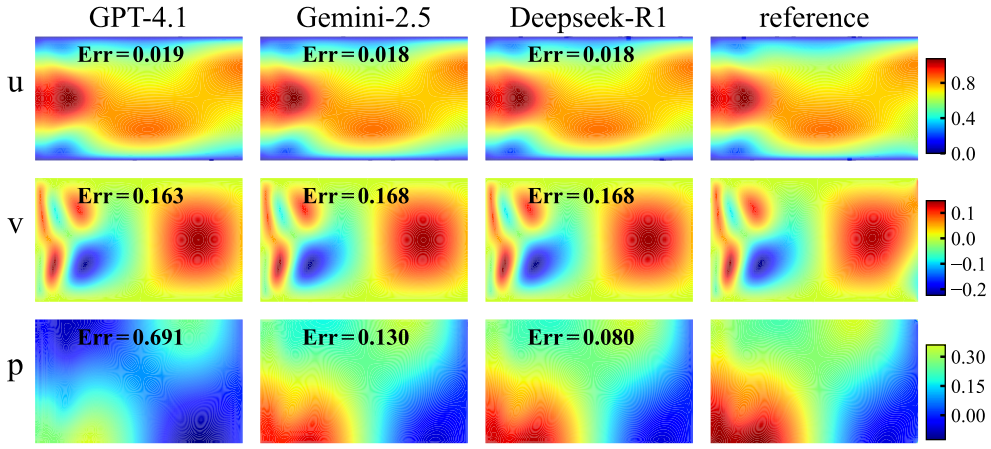


Figure 11: The best runtime outputs of the executable code provided by each Programmer in the final response (review-2).

For comparison, the contour plots and their corresponding relative errors are visualized in figure 11, which depicts the best runtime outputs from each Programmer after two rounds of Reviewer guidance. Specifically, for Gemini, the relative error in streamwise velocity from its execution results remains minimal, whether in the minority of executable code provided in the initial response or the majority of executable code after Reviewer interventions. In contrast, GPT demonstrates a weaker overall improvement in the relative error, while R1 achieves the most significant decrease, highlighting the agent's superior performance in iterative self-refinement.

B.4 Detailed description of other equations

B.4.1 Burgers equation

The Burgers equation is a prototypical PDE that couples a nonlinear convective term and a viscous diffusion term (Benton & Platzman, 1972). When the viscosity coefficient ν is small, the nonlinear convective term dominates, causing the initially smooth velocity profile to gradually form discontinuous abrupt changes (i.e., shock waves).

The specific equations and initial/boundary conditions are detailed in the following prompt template:

Problem Description Prompt: Burgers Equation

The PDE of 1-D Burgers equation is given by:

$$\frac{\partial u}{\partial t} + uu_x - \nu u_{xx} = 0, (x,t) \in \Omega$$

where the domain is defined as $(x, y) \in \Omega = [-1, 1] \times [0, 1]$, the the parameter is $\nu = \frac{0.01}{\pi}$. The initial and boundary conditions are:

$$\begin{cases} u(x,0) = -sin(\pi x) \\ u(-1,t) = u(1,t) = 0 \end{cases}$$

Implement a stable and efficient method to solve this problem.
Plot the contour of the velocity magnitude and velocity profile in only one figure.

The statistical results are shown in Figures 12 and 13, while the optimal runtime outputs from the best-of-n perspective are depicted in figure 14.

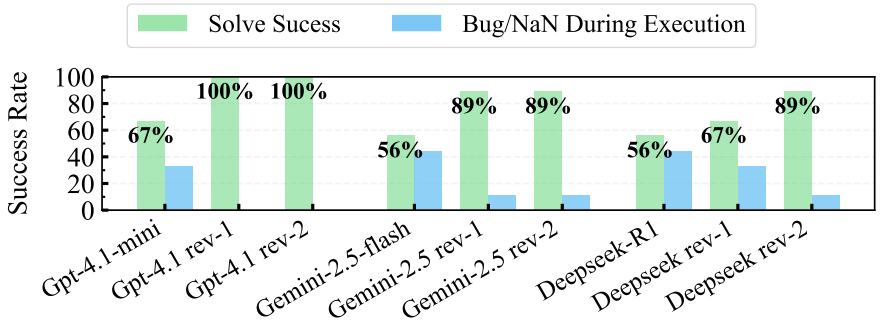


Figure 12: Execution success rate of the numerical algorithms employed by Programmers to solve the Burgers equation.

B.4.2 Sod shock tube

The Sod shock tube problem (Sod, 1978) is a classic one-dimensional PDE system as depicted in figure 15. The governing equations for this problem are the Euler equations simplified from the compressible NS equations, used to describe the wave systems in inviscid compressible flows.

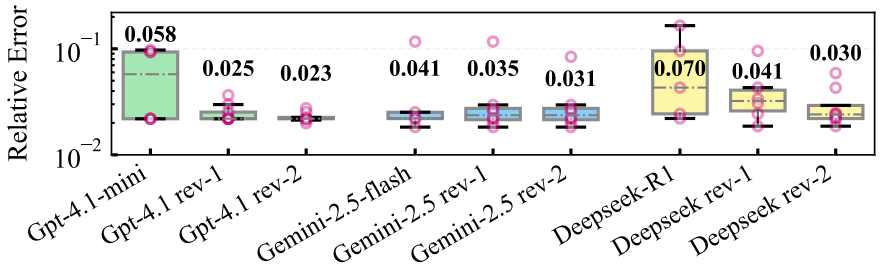


Figure 13: Boxplot of Relative L^2 error for successful solutions by Programmers.The **annotated numbers** in the figure indicate the mean relative L^2 errors.

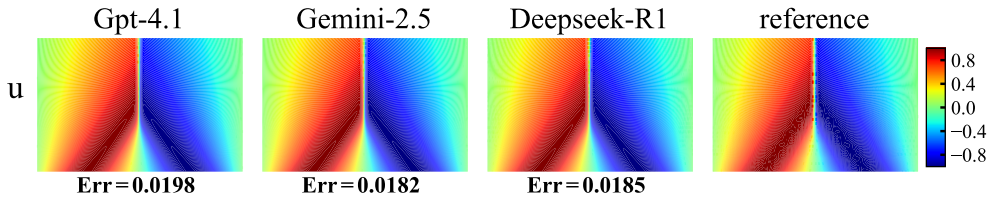


Figure 14: The best runtime outputs of the executable code provided by each Programmer in the final response (review-2).

At the initial time $t = 0$, the left domain of the diaphragm is occupied by high-pressure gas, while the right domain contains low-pressure gas. Upon removal of the diaphragm at $t > 0$, the high-pressure gas undergoes rapid expansion into the low-pressure region on the right, generating three distinct wave systems.

(i) a left-traveling rarefaction wave where pressure and density exhibit continuous variation across the wave front, with velocity gradually increasing to its peak.

(ii) a right-traveling shock wave where pressure and density sharply increase across the shock wave front while velocity decreases abruptly.

(iii) a contact between the two waves, where pressure and velocity remain continuous across the discontinuity surface, but density experiences an abrupt jump.

The specific equations and initial/boundary conditions are detailed in the following prompt template:

Problem Description Prompt: Sod Shock Tube

The PDEs of 1-D Euler equations are given by:

$$\begin{cases} \frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} = 0, \\ \frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2 + p)}{\partial x} = 0, \\ \frac{\partial(\rho E)}{\partial t} + \frac{\partial(\rho E u + p u)}{\partial x} = 0, \end{cases}$$

where the total energy $E = \frac{1}{2}u^2 + \frac{p}{(\gamma-1)\rho}$ and the adiabatic index $\gamma = 1.4$.
At $t = 0$, the initial conditions in the interval $x \in [0, 1]$ are:

$$(\rho, u, p) = \begin{cases} (1.0, 0.0, 1.0), & 0 < x \leq 0.5, \\ (0.125, 0.0, 0.1), & 0.5 \leq x < 1. \end{cases}$$

Implement a stable and efficient method to solve this problem.
Plot the density ρ, velocity u, and pressure p at $t = 0.2$ in one figure.

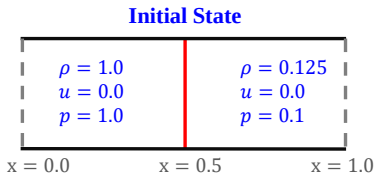


Figure 15: Schematic of the Sod shock tube with initial conditions at $t = 0$

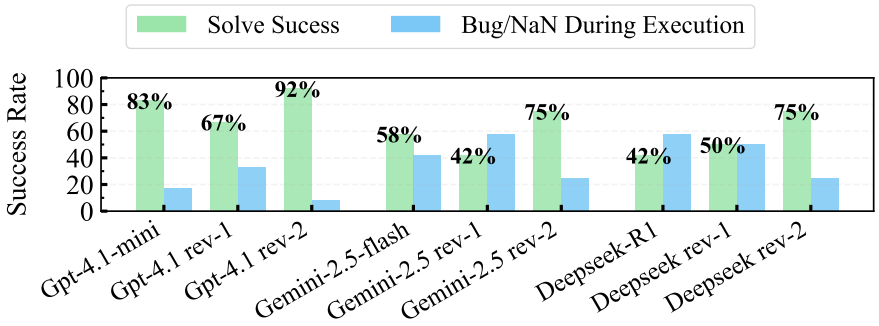


Figure 16: Execution success rate of the numerical algorithms employed by Programmers to solve the Sod shock tube problem.

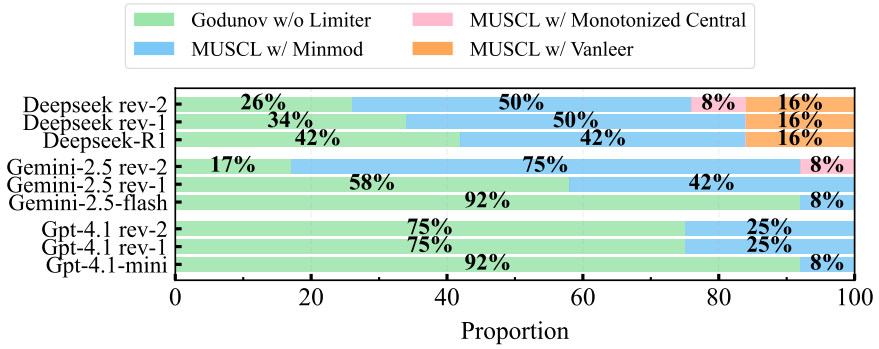


Figure 17: Proportional distribution of the spatial reconstruction schemes employed by Programmers. All Programmers selected the Harten-Lax-vanLeer-Contact (HLLC) Riemann solver during the sampling process.

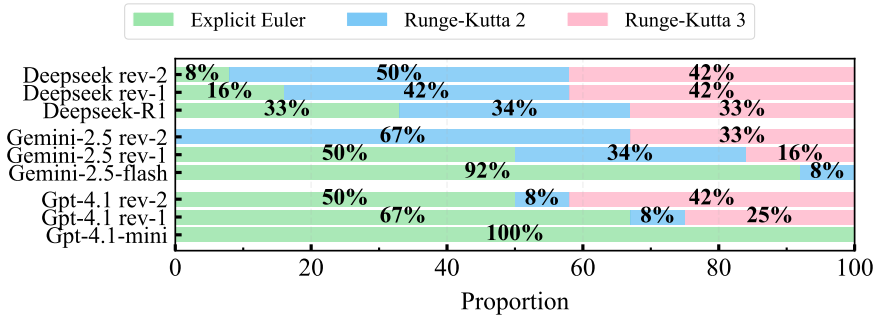


Figure 18: Proportional distribution of the time integration methods employed by Programmers.

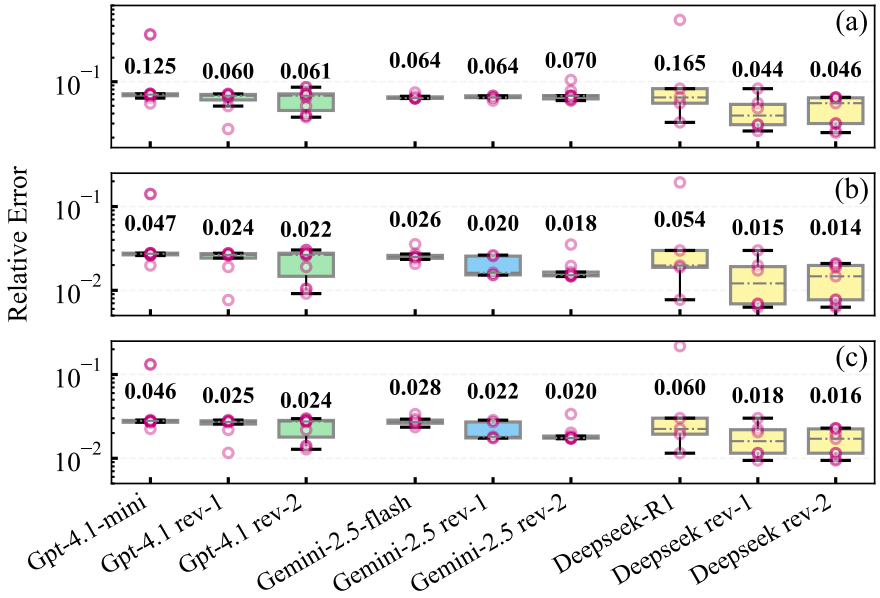


Figure 19: Boxplot of Relative L^2 error for successful solutions by Programmers. (a) velocity u, (b) pressure p, (c) density ρ.The **annotated numbers** in the figure indicate the mean relative L^2 errors.

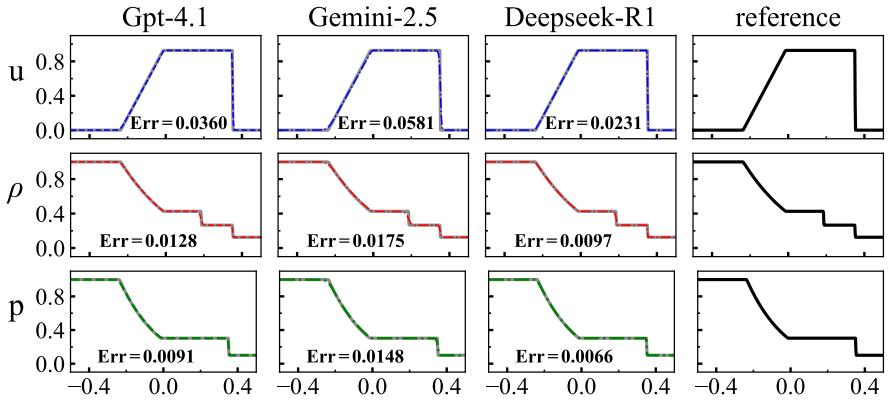


Figure 20: The best runtime outputs of the executable code provided by each Programmer in the final response (review-2).

In figure 16, the code execution success rates of **GPT-4.1-mini** and **Gemini-2.5-flash** as Programmers each exhibit a decline during the review-1 stage, followed by a subsequent recovery. To explain this trend, additional statistics on the proportions of numerical methods employed by each Programmer across different response stages were compiled, as presented in Figures 17-18. Results indicate that with Reviewer intervention, the proportional distribution of spatiotemporally high-order schemes increased significantly, which improved the accuracy but also elevated the risk of introducing bugs or NaN values in specific code implementations due to the inherent complexity of these schemes.

The boxplot of relative errors and the 1D physical quantity plots compiled from runtime outputs are presented in figure 19 and 20, respectively. From both statistical and best-of-n sample perspectives, the agent framework demonstrates excellent performance in solving this problem.

B.4.3 POISSON EQUATION

The Poisson equation (Strauss, 2007), as a typical elliptic partial differential equation, the uniqueness of its solution is completely determined by the boundary conditions. In this case, the solution domain

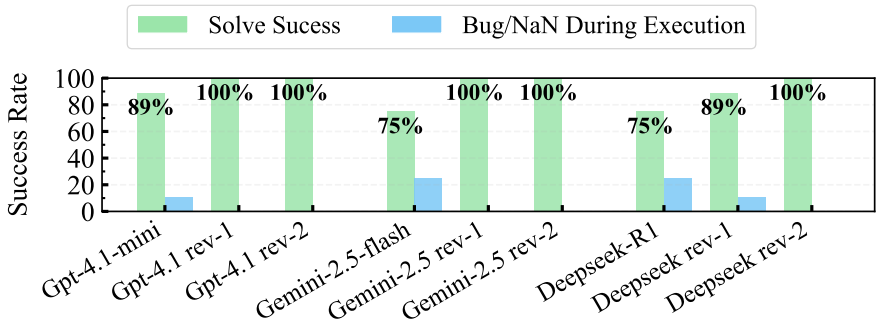


Figure 21: Execution success rate of the numerical algorithms employed by Programmers to solve the Poisson equation.

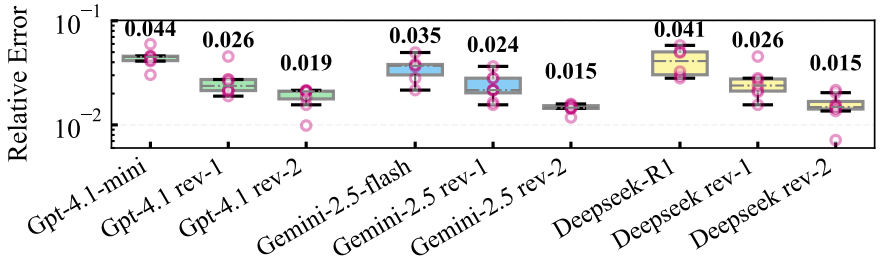


Figure 22: Boxplot of Relative L^2 error for successful solutions by Programmers.The **annotated numbers** in the figure indicate the mean relative L^2 errors.

is a multiply connected domain formed by a rectangle with four symmetric circular holes excavated. This complex configuration composed of a regular outer boundary and an irregular inner boundary poses challenges to numerical methods.

The specific equations and initial/boundary conditions are detailed in the following prompt template:

Problem Description Prompt: Poisson Equation

The PDE of 2-D Poisson equation is given by:

$$\Delta u = 0, (x, y) \in \Omega$$

The domain is a rectangle minus several circles $\Omega = \Omega_{\text{rec}} \setminus \Omega_{\text{circle}}$, where $\Omega_{\text{rec}} = [-0.5, 0.5]^2$ is the rectangle, and the circles $\Omega_{\text{circle}} = \bigcup_{i=1}^{4} R_i$ are defined as:

$$\begin{cases} R_1 = \{(x, y) : (x - 0.3)^2 + (y - 0.3)^2 \leq 0.1^2\} \\ R_2 = \{(x, y) : (x + 0.3)^2 + (y - 0.3)^2 \leq 0.1^2\} \\ R_3 = \{(x, y) : (x - 0.3)^2 + (y + 0.3)^2 \leq 0.1^2\} \\ R_4 = \{(x, y) : (x + 0.3)^2 + (y + 0.3)^2 \leq 0.1^2\} \end{cases}$$

The boundary conditions are:

$$\begin{cases} u = 0, \ x \in \partial\Omega_{\text{circle}} \\ u = 1, \ x \in \partial\Omega_{\text{rec}} \end{cases}$$

Implement a stable and efficient method to solve this problem.
Plot the contour of $u(x, y)$ in one figure, and mark the circles in the plot.

The statistical results are shown in Figures 21 and 22, while the optimal runtime outputs are depicted in figure 23.

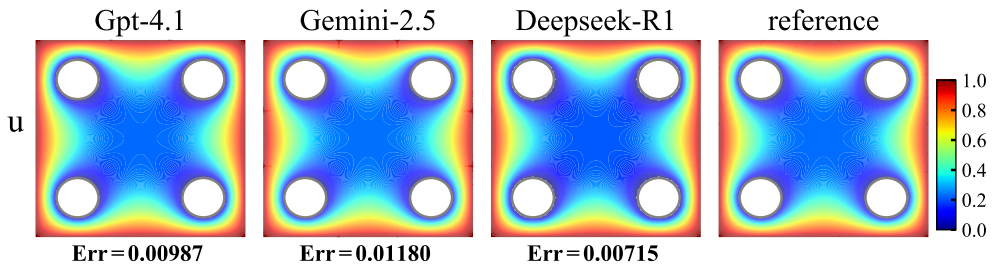


Figure 23: The best runtime outputs of the executable code provided by each Programmer in the final response (review-2).

B.4.4 HELMHOLTZ EQUATION

The Helmholtz equation (Strauss, 2007), a time-independent form of the wave equation, often emerges in spatiotemporal PDE studies. Derived via separation of variables to simplify analysis, it captures steady-state wave behaviors. As a typical elliptic PDE, the uniqueness of its solution is jointly determined by the boundary conditions and the wave number k. In this case, the solution domain is a simple rectangular region with four circular holes excavated, forming a multiply-connected domain.

The specific equations and initial/boundary conditions are detailed in the following prompt template:

Problem Description Prompt: Helmholtz Equation

The PDE of 2-D Helmholtz equation is given by:

$$-\Delta u + k^2 u = f(x, y), (x, y) \in \Omega$$

The function f(x) is defined as:

$$f(x) = A \cdot \left(\sum_i \mu_i^2 + x_i^2 \right) \sin(\mu_1 \pi x_1) \sin(\mu_2 \pi x_2)$$

The parameter values are:

$$\mu_1 = 1, \quad \mu_2 = 4, \quad k = 8, \quad A = 10$$

The domain is a rectangle minus several circles $\Omega = \Omega_{\text{rec}} \setminus \Omega_{\text{circle}}$, where $\Omega_{\text{rec}} = [-1, 1]^2$ is the rectangle, and the circles $\Omega_{\text{circle}} = \bigcup_{i=1}^{4} R_i$ are defined as:

$$\begin{cases} R_1 = \{(x, y) : (x - 0.5)^2 + (y - 0.5)^2 \leq 0.2^2\} \\ R_2 = \{(x, y) : (x - 0.4)^2 + (y + 0.4)^2 \leq 0.4^2\} \\ R_3 = \{(x, y) : (x + 0.2)^2 + (y + 0.7)^2 \leq 0.1^2\} \\ R_4 = \{(x, y) : (x + 0.6)^2 + (y - 0.5)^2 \leq 0.3^2\} \end{cases}$$

The boundary conditions are:

$$\begin{cases} u = 0.2, & x \in \partial\Omega_{\text{rec}} \\ u = 1, & x \in \partial\Omega_{\text{circle}} \end{cases}$$

Implement a stable and efficient method to solve this problem.
Plot the contour of $u(x, y)$ in one figure, and mark the circles in the plot.

The statistical results are shown in Figures 24 and 25, while optimal runtime outputs are shown in figure 26.

B.4.5 LID-DRIVEN CAVITY

Lid-driven cavity flow (Ghia et al., 1982) refers to a flow phenomenon in which a specified velocity is imposed on the top boundary of a closed cavity, whereas the other boundaries remain stationary.

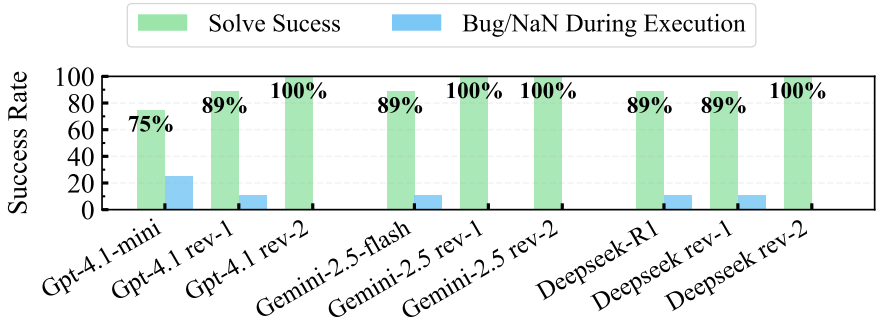


Figure 24: Execution success rate of the numerical algorithms employed by Programmers to solve the Helmholtz equation.

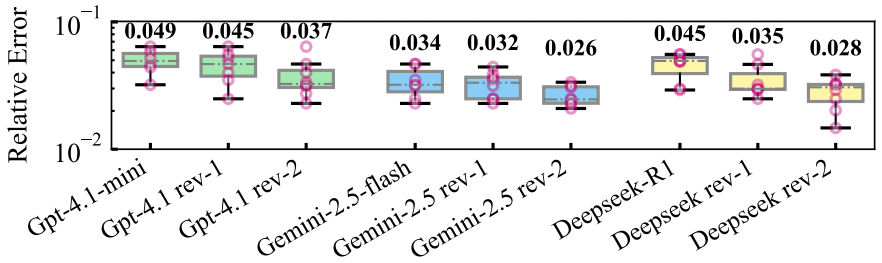


Figure 25: Boxplot of Relative L^2 error for successful solutions by Programmers.The **annotated numbers** in the figure indicate the mean relative L^2 errors.

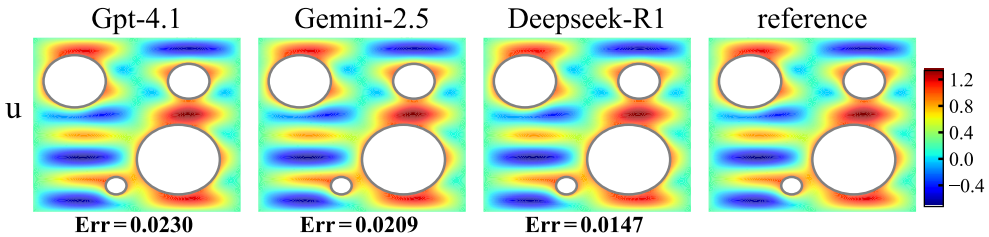


Figure 26: The best runtime outputs of the executable code provided by each Programmer in the final response (review-2).

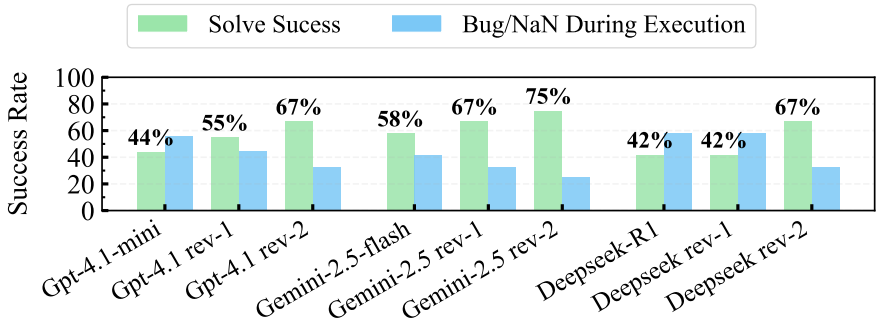


Figure 27: Execution success rate of the numerical algorithms employed by Programmers to solve the Lid-driven flow.

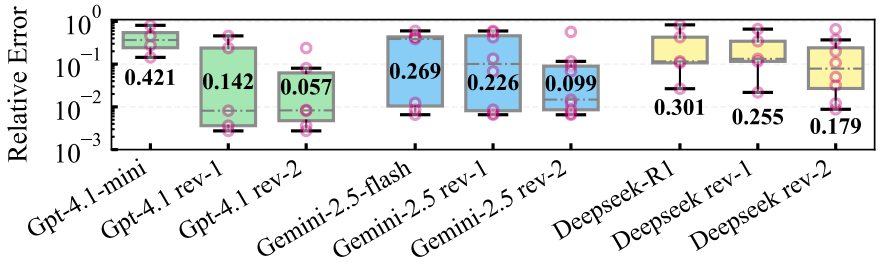


Figure 28: Boxplot of Relative L^2 error for successful solutions by Programmers.The **annotated numbers** in the figure indicate the mean relative L^2 errors.

The governing equations are the incompressible steady NS equations, which describe the motion of the fluid inside the cavity under the influence of viscosity.

The inherent mathematical properties of these governing equations pose significant challenges for robust numerical solutions, primarily arising from the following two aspects:

(i) **Pressure-Velocity Coupling:** Since the divergence-free continuity equation does not explicitly contain pressure while the momentum equations rely on pressure gradients, the pressure and velocity fields cannot be solved independently. This necessitates the use of special algorithms to effectively handle the coupling between pressure and velocity fields.

(ii) **Nonlinear Convection:** The presence of nonlinear convection terms often leads to numerical instabilities, such as iterative divergence or slow convergence, making the solver design particularly demanding.

The specific equations and initial/boundary conditions are detailed in the following prompt template:

Problem Description Prompt: Lid-driven flow

The PDEs of 2-D steady incompressible Navier-Stokes equations is given by:

$$\begin{cases} u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0, (x,y) \in \Omega, \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{\mathrm{Re}}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0, (x,y) \in \Omega, \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, (x,y) \in \Omega, \end{cases}$$

The Reynolds number Re = 100. The domain is $\Omega = ([0,2]^2$, and the top boundary is Γ_1, the left, right and bottom boundary is Γ_2. The boundary conditions are:

$$\begin{cases} (u,v) = (\alpha(x(1-x)), 0), (x,y) \in \Gamma_1 \\ (u,v) = (0,0), (x,y) \in \Gamma_2 \\ \text{Reference pressure: } p(x=0, y=0) = 0 \\ \text{Zero normal pressure gradient: } \frac{\partial p}{\partial n} = 0, (x,y) \in \partial\Omega \end{cases}$$

where α is 2. Implement a stable and efficient method to solve this problem.
Plot the contour of the velocity magnitude overlaid with streamlines, and the convergence history in one figure.

The statistical results are shown in Figures 27 and 28, while optimal runtime outputs are shown in figure 29.

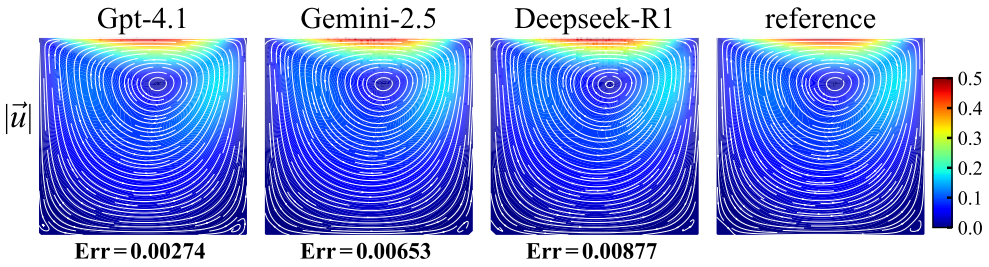


Figure 29: The best runtime outputs of the executable code provided by each Programmer in the final response (review-2).

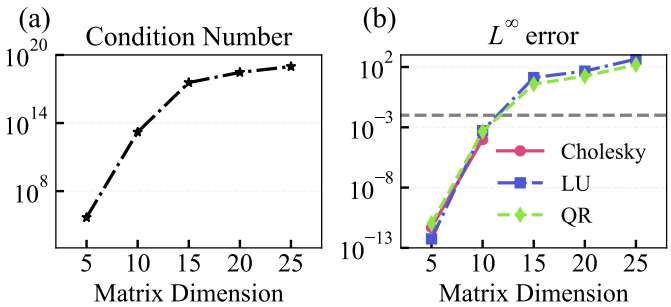


Figure 30: Schematic diagram of (a) variation of the 2-norm condition number with matrix dimension; (b) L^{∞} error obtained by direct solution using the naive methods.

C HILBERT LINEAR SYSTEMS DETAILS

C.1 PROBLEM STATEMENT

The specific linear algebraic system involving Hilbert matrix is given by:

$$\mathbf{b} = H_n \mathbf{x}^* = \begin{bmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \cdots & \frac{1}{2n-1} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{2}$$

where H_n is a $n \times n$ Hilbert matrix with elements $h_{i,j} = \frac{1}{i+j-1}$, and the vector $\mathbf{b}$ is obtained by substituting the exact solution $\mathbf{x}^* = (1, 1, \ldots, 1)^\top$ into $H_n \mathbf{x}^*$. The Hilbert matrix serves as a prototypical example of ill-conditioned matrices, whose 2-condition number $\kappa_2(H_n)$ grows exponentially with the dimension n, as shown in figure 30.

The extremely high condition number renders naive methods ineffective, that is, even minor perturbations in input data could lead to drastic errors in the computed solution, thereby posing significant challenges for solving this ill-conditioned matrix. The following template is adopted as the problem description prompt for input into the agent framework:

Problem Description Prompt: Hilbert Matrix

Implement various appropriate methods from scratch to solve the system of linear algebra equations $H_n \mathbf{x} = \mathbf{b}$ accurately and efficiently, where H_n is an $n \times n$ matrix defined by $h_{ij} = 1/(i + j - 1)$, and the vector $\mathbf{b}$ is taken to ensure that the system admits an exact solution $\mathbf{x}^* = (x_i)_{n\times 1} = (1)_{n\times 1}$. Compare the L_∞ error of the numerical results with the exact solution $\mathbf{x}^*$ for $n = 5, 10, 15, 20, 25$.

To prevent the agent from cheating by using numpy.linalg.solve or scipy.linalg.solve, while also avoiding unfair performance evaluations caused by certain methods' inherent lack of adaptability when addressing ill-conditioned matrices, we explicitly required the agent to **implement various appropriate methods from scratch**.

C.2 SUPPLEMENTARY RESULTS

Programmer	Methods	L^∞ error w.r.t. matrix size n				
		5	10	15	20	25
GPT-4.1-mini	Chol-Reg	3.08e-04	6.81e-04	9.66e-04	1.36e-03	1.36e-03
	LU-Reg	3.15e-04	8.77e-04	8.92e-04	1.38e-03	1.37e-03
	QR-Reg	3.20e-04	9.90e-04	1.18e-03	1.31e-03	1.62e-03
	Pre-CG	**9.80e-12**	8.79e-05	7.16e-05	**4.32e-05**	**9.96e-05**
	SVD	1.46e-11	7.32e-04	3.91e-03	7.81e-03	7.32e-04
Gemini-2.5-flash	Chol-Reg	8.64e-04	1.12e-03	1.45e-03	1.61e-03	1.73e-03
	CG	4.49e-04	9.34e-04	4.85e-04	6.60e-04	4.23e-04
	Pre-CG	8.44e-04	5.57e-04	5.19e-04	7.50e-04	7.41e-04
	SVD	7.13e-04	4.76e-04	5.36e-04	5.02e-04	6.30e-04
Deepseek-R1	Chol-Reg	1.11e-09	**3.84e-05**	**5.05e-05**	1.45e-04	2.68e-04
	LU-Reg	1.11e-09	8.61e-05	1.81e-04	1.51e-04	2.01e-04
	QR-Reg	3.80e-03	6.07e-04	1.01e-03	1.10e-03	1.31e-03
	CG	5.42e-11	5.99e-04	3.93e-04	2.09e-04	4.30e-04
	Pre-CG	**9.80e-12**	6.01e-04	3.95e-04	2.08e-04	4.32e-04

Table 5: The summary of L^∞ errors in solving linear systems with Hilbert matrices via various methods provided by each Programmer in the final response (review-2).

Figure 31 serving as the supplement, presents the proportions of different methods across the three completion statuses of executable code. It provides a granular view of how regularization techniques and iterative methods significantly outperform naive approaches in solving ill-conditioned matrices.

Table 5 summarizes the relative errors of the best-performing methods provided by each Programmer in the final response (review-2). The order-of-magnitude reduction in relative errors compared to the naive methods (shown in figure 30) underscores the generalizability and robustness of the agent framework.

D DIMENSIONAL ANALYSIS DETAILS

D.1 PROBLEM STATEMENT

During laser-metal interaction, a vapor-filled depression (i.e., keyhole) typically forms in the molten metal pool. Induced by vaporization-driven recoil pressure, its dynamics are inherently complex due to multi-physical dependencies. Quantifying this phenomenon is critical, as it directly influences energy absorption and defect formation in diverse industrial applications.

By leveraging X-ray pulses, images of the keyhole region within metals can be captured at micrometer-level spatial resolution. The keyhole depth e, measurable from these X-ray images, is contingent on diverse materials and process parameters, including effective laser power (η_p), laser scan speed (V_s), and laser beam radius (r_0).

A dataset of keyhole X-ray images was collated from the work of Zhao et al. (2019) and Gan et al. (2021), encompassing 90 experiments across three materials (titanium alloy Ti6Al4V, aluminium alloy Al6061, and stainless steel SS316) under varied process conditions. And materials are characterized by thermal diffusivity (α), density (ρ), heat capacity (C_p), and the temperature difference ($T_l - T_0$) between melting and ambient temperatures. Thus, the causal relationship is formalized as:

$$e^* = f(\eta_p, V_s, r_0, \alpha, \rho, C_p, T_l - T_0) \tag{3}$$

Where the output variable is normalized as the keyhole aspect ratio $e^* = \frac{e}{r_0}$, a dimensionless parameter widely used to characterize keyhole features (Fabbro et al., 2018). The aim is to search the dimensionless parameter space based on the R^2 criterion for dominant dimensionless numbers in

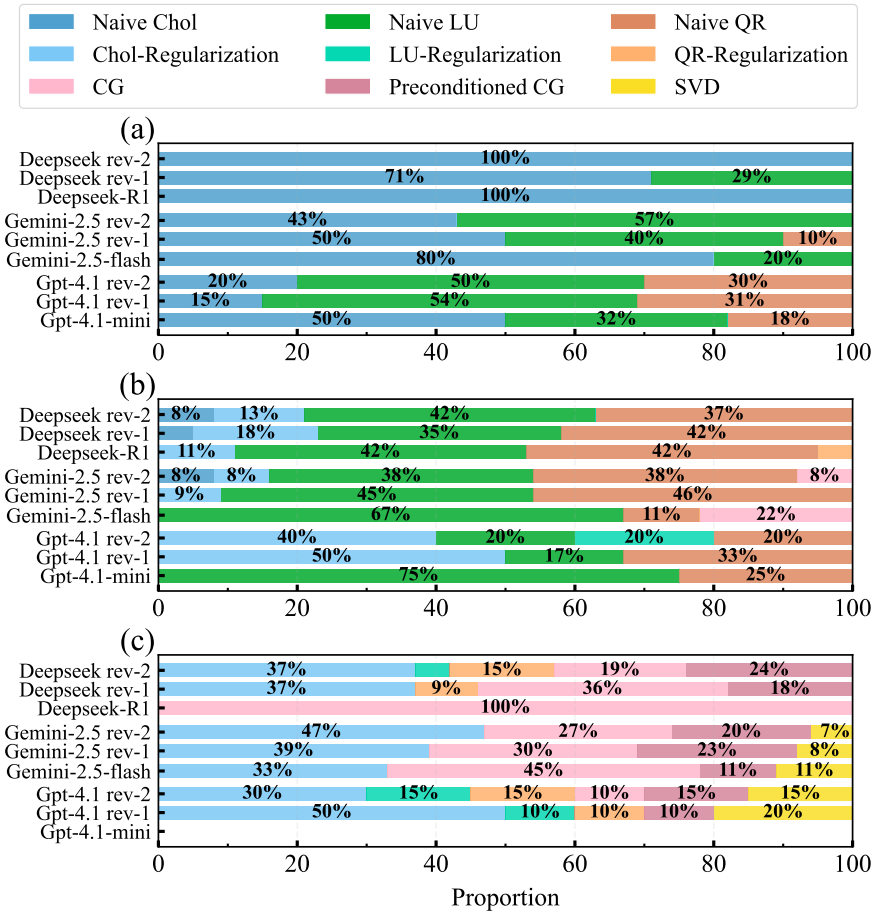


Figure 31: Proportional distribution of specific numerical methods employed by Programmers across three completion statuses, serving as a detailed elaboration to the solution success rate. (a) Results Contain NaN; (b) Over L^∞ threshold; (c) Below L^∞ threshold.

keyhole dynamics, with the following template adopted as the problem description prompt for input into the agent framework:

Problem Description Prompt: Keyhole Dimensional Analysis

Please read the CSV data file from path `./dataset_keyhole.csv` and consider the data in columns 3, 4, 5, 6, 7, 8 and 11. These columns correspond to seven physical quantities respectively:

the effective laser power (ηP), the laser scan speed (V_s), the laser beam radius (r_0), the thermal diffusivity (α), the material density (ρ), the heat capacity (C_p), and the difference between melting and ambient temperatures ($T_l - T_0$).

Based on dimensional analysis and using the data of these physical quantities in the file, please identify the optimal dimensionless quantity formed by combining these parameters, which exhibits the highest coefficient of determination (R^2) and thus dominates the variation in the keyhole aspect ratio e^* (data in the last column).

Implement a robust and reliable method from scratch for this data-driven dimensional analysis.

Ensure that the resulting dimensionless exponents are normalized by V_s and that the exponents of physical quantities be integers or rational fractions with absolute values not exceeding 3.

D.2 GROUND TRUTH AND PHYSICAL CONSTRAINTS

As established in the work of Xie et al. (2022), for the present dimensional analysis problem, the dominant dimensionless number emerging from keyhole dynamics is

$$\Pi = \frac{\eta P}{(T_l - T_0)\rho C_p \sqrt{\alpha V_s r_0^3}} \tag{4}$$

This dimensionless number exhibits a form identical to the newly discovered keyhole number Ke (Gan et al., 2021) (also referred to as normalized enthalpy (Ye et al., 2019)), which can be derived from heat transfer theory. Therefore, we adopt the dimensionless number Ke as the metric to evaluate the success rate of the executable code provided by the agent framework. At the end of the above-mentioned prompt template, we propose constraints on dimensionless exponents which are mainly derived from the following physical insights:

(i) To avoid identifying equivalent dimensionless numbers with different powers and reduce the computational cost, we select the laser scan speed V_s for normalizing the exponents.

(ii) The exponents of dimensionless numbers tend to be rational numbers to maintain dimensional invariance (Xie et al., 2022), with a preference for small rational powers such as -1, 1, or 2, etc.

(iii) The typical range of the exponents for dimensionless numbers is limited, with the absolute value of coefficients in most dimensionless numbers and scaling laws being less than 4 (Barenblatt, 2003).