AUTONUMERICS: AN AUTONOMOUS, PDE-AGNOSTIC MULTI-AGENT PIPELINE FOR SCIENTIFIC COMPUTING

Jianda Du[1] **Youran Sun**[1] **Haizhao Yang**[1,2,*]

[1]Department of Mathematics, University of Maryland, College Park, MD, USA
[2]Department of Computer Science, University of Maryland, College Park, MD, USA
jdu37576@umd.edu sun1245@umd.edu hzyang@umd.edu

ABSTRACT

PDEs are central to scientific and engineering modeling, yet designing accurate numerical solvers typically requires substantial mathematical expertise and manual tuning. Recent neural network-based approaches improve flexibility but often demand high computational cost and suffer from limited interpretability. We introduce `AutoNumerics`, a multi-agent framework that autonomously designs, implements, debugs, and verifies numerical solvers for general PDEs directly from natural language descriptions. Unlike black-box neural solvers, our framework generates transparent solvers grounded in classical numerical analysis. We introduce a coarse-to-fine execution strategy and a residual-based self-verification mechanism. Experiments on 24 canonical and real-world PDE problems demonstrate that `AutoNumerics` achieves competitive or superior accuracy compared to existing neural and LLM-based baselines, and correctly selects numerical schemes based on PDE structural properties, suggesting its viability as an accessible paradigm for automated PDE solving.

1 INTRODUCTION

Partial differential equations (PDEs) form the mathematical foundation of modern physics, engineering, and many areas of scientific computing. Accurately solving PDEs is therefore a central task in computational research. Traditionally, constructing a reliable numerical solver for a new PDE requires substantial expertise in numerical analysis, including the selection of appropriate discretization schemes (e.g., finite difference, finite element, or spectral methods) and verification of stability and convergence conditions such as the Courant–Friedrichs–Lewy (CFL) constraint (LeVeque, 2007). These classical approaches provide strong mathematical guarantees and interpretability, but their expert-driven design can limit accessibility and slow solver development for newly arising PDE models.

Neural network-based approaches such as physics-informed neural networks (PINNs) (Raissi et al., 2019) and operator-learning frameworks (Lu et al., 2019; Li et al., 2020) reduce reliance on handcrafted discretizations but introduce new concerns around computational cost and interpretability. Large language models (LLMs) have recently demonstrated strong capabilities in scientific code generation (Zhang et al., 2024), and existing LLM-assisted PDE efforts include neural solver design (He et al., 2025; Jiang & Karniadakis, 2025), tool-oriented systems that invoke libraries such as FEniCS (Liu et al., 2025; Wu et al., 2025), and code-generation paradigms (Li et al., 2025). However, these approaches either produce black-box networks, are constrained by fixed library APIs, or lack mechanisms for autonomous debugging and correctness verification. We propose that LLMs can serve as *numerical architects* that directly generate transparent solver code from first principles, preserving interpretability while automating solver construction.

Translating this vision into a reliable system poses several technical challenges. First, LLM-generated code often contains syntax errors or logical flaws, and debugging these errors on high-

*Corresponding author.

resolution grids is both time-consuming and computationally wasteful. Second, verifying solver correctness becomes difficult for PDEs lacking analytical solutions. Third, large-scale temporal simulations may lead to memory exhaustion. We address these challenges with three corresponding solutions. A coarse-to-fine execution strategy first debugs logic errors on low-resolution grids before running on high-resolution grids. A residual-based self-verification mechanism evaluates solver quality for problems without analytical solutions by computing PDE residual norms. A history decimation mechanism enables large-scale temporal simulations through sparse storage of intermediate states.

Building on these design principles, we propose `AutoNumerics`, a multi-agent autonomous framework. The system receives natural language problem descriptions, proposes multiple candidate numerical strategies through a planning agent, implements executable solvers, and systematically evaluates their correctness and performance. We evaluate the framework on 24 representative PDE problems spanning canonical benchmarks and real-world applications. Results demonstrate consistent numerical scheme selection, stable solver synthesis, and reliable accuracy across diverse PDE classes.

Position relative to prior work. Existing LLM-assisted PDE efforts include neural solver design (He et al., 2025; Jiang & Karniadakis, 2025), tool-oriented systems that invoke libraries such as FEniCS (Liu et al., 2025; Wu et al., 2025), and code-generation paradigms (Li et al., 2025). `AutoNumerics` differs from all three. It generates interpretable classical numerical schemes (not black-box networks), automatically detects and filters ill-designed or non-expert numerical plan configurations, derives discretizations from first principles (not fixed library APIs), and includes a coarse-to-fine execution strategy with residual-based self-verification for autonomous correctness assessment. A detailed review of related work is provided in Appendix A.

Contributions. The primary contributions of this work are:

- A multi-agent framework (`AutoNumerics`) that autonomously constructs transparent numerical PDE solvers from natural language descriptions.
- A reasoning module that detects ill-designed or non-expert PDE specifications and proactively filters or revises numerical plans that may lead to instability or invalid solutions.
- A coarse-to-fine execution strategy that decouples logic debugging from stability validation.
- A residual-based self-verification mechanism for solver evaluation without analytical solutions.
- A benchmark suite of 200 PDEs and systematic evaluation on 24 representative problems, with comparisons to neural network baselines and CodePDE.

The related code of our method and the benchmark dataset are available at `https://github.com/Daviddjddu/Autonumerics`.

2 METHOD

2.1 PROBLEM FORMULATION AND PLAN GENERATION

AutoNumerics consists of multiple specialized LLM agents coordinated by a central dispatcher. The system takes a natural language PDE problem description as input and produces executable numerical solver code with accuracy metrics as output. The overall architecture is illustrated in Figure 1.

The pipeline begins with the Formulator Agent, which converts the natural language description into a structured specification containing governing equations, boundary and initial conditions, and physical parameters. The Planner Agent then proposes multiple candidate schemes covering different discretization methods (e.g., finite difference, spectral, finite volume) and time-stepping strategies (explicit, implicit), while avoiding configurations that violate basic numerical stability and consistency principles. The Feature Agent extracts numerical features from both the problem and the proposed schemes, and the Selector Agent scores and ranks these candidates, further filtering out ill-designed or nonphysical plans before selecting the top-k for execution.

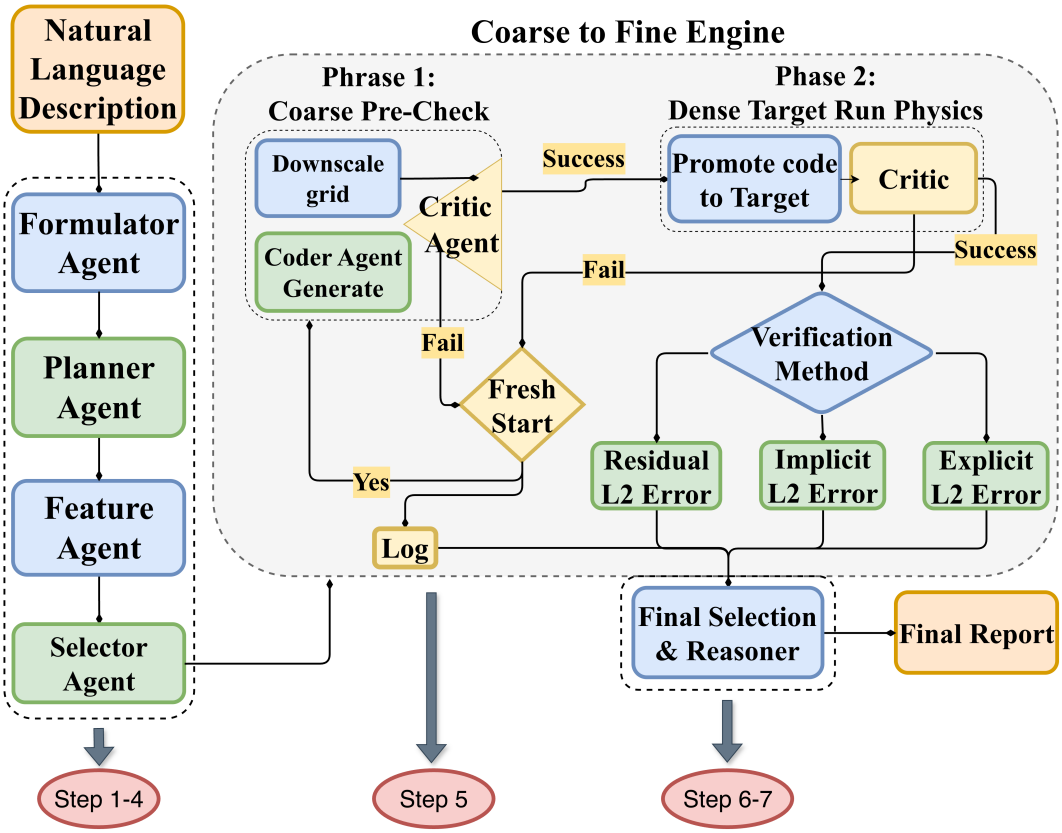

Figure 1: The AutoNumerics pipeline. Steps 1–4 handle problem formulation and plan selection. Step 5 implements the coarse-to-fine execution strategy with Fresh Restart logic. Steps 6–7 perform verification and theoretical analysis.

2.2 Coarse-to-Fine Execution

Debugging LLM-generated code directly on high-resolution grids is computationally wasteful. We decouple logic debugging from stability validation through a coarse-to-fine strategy. In the coarse-grid phase, the solver runs at reduced resolution, and the Critic Agent fixes logic issues (syntax errors, shape mismatches). Once logic validation passes, the code is promoted to the high-resolution grid, where failures are treated as numerical stability issues and addressed by adjusting the time step.

If repair attempts exceed the retry limit M at either stage, the system triggers a Fresh Restart. The current code is discarded and the Coder Agent generates a new implementation from scratch, enabling the system to escape failed code paths. For large-scale temporal simulations, the Coder Agent is instructed to store solution snapshots only at sparse intervals to avoid memory exhaustion.

2.3 Verification and Analysis

Verifying solver correctness is a core challenge in automated PDE solving. Let u denote the numerical solution, u^* the analytic solution (when available), and $\mathcal{L}$ the PDE operator. When an explicit analytic solution exists, we compute the relative L_2 error; when no analytic solution is available, we evaluate the relative PDE residual; and for implicit analytic relations (e.g., conservation laws $F(u) = 0$), we measure the relative implicit residual. These three errors are defined respectively as

$$e_{L_2} = \frac{\|u - u^*\|_{L^2(\Omega)}}{\|u^*\|_{L^2(\Omega)} + \epsilon}, \quad e_{\text{res}} = \frac{\|\mathcal{L}(u) - f\|_{L^2(\Omega)}}{\|f\|_{L^2(\Omega)} + \epsilon}, \quad e_{\text{impl}} = \frac{\|F(u)\|_{L^2(\Omega)}}{\|F_{\text{ref}}\|_{L^2(\Omega)} + \epsilon}, \text{where } \epsilon = 10^{-12} \tag{1}$$

Generated solvers are required to compute and return residuals, and the system enforces validity checks on these values. Finally, a Reasoning Agent generates theoretical analysis for the best-performing scheme.

3 Experiments & Results

3.1 Experimental Setup

Benchmark: We evaluate our framework on two benchmarks: **(1) CodePDE Benchmark.** To enable fair comparison with existing neural network solvers and LLM-based methods, we adopt the benchmark proposed by CodePDE, which comprises 5 representative PDEs: 1D Advection, 1D Burgers, 2D Reaction-Diffusion, 2D Compressible Navier-Stokes (CNS), and 2D Darcy Flow. These problems span linear and nonlinear equations, elliptic and time-dependent types, as well as diverse boundary conditions and levels of numerical stiffness. **(2) Our Benchmark.** To more comprehensively assess the generality of our framework, we construct a large-scale benchmark suite containing 200 different PDEs, covering a wide range of common PDE families (Advection, Burgers, Fokker-Planck, Heat, Maxwell, Poisson, etc.). The PDEs in our benchmark range from 1D to 5D in spatial dimension and span elliptic, parabolic, hyperbolic types as well as PDE systems. They include linear and nonlinear, stiff and non-stiff, steady-state and time-dependent problems, with Dirichlet, Neumann, and periodic boundary conditions.

Numerical Settings: The Planner Agent generates 10 candidate solver schemes and scores each one for every PDE problem. The top-5 schemes are passed to the Coder Agent for implementation. We set the maximum number of retries for code generation, coarse-grid execution, and high-resolution execution to 2, 4, and 6, respectively. The maximum wall-clock time for each coarse-grid or high-resolution run is 120 seconds.

Evaluation Metrics: We evaluate solver accuracy using the three metrics defined in Section 2.3 (e_{L_2}, e_{impl}, e_{res}) 1, depending on the available reference information. We also report execution time, defined as the wall-clock time from solver generation to the first successful evaluation.

3.2 Results and Analysis

Table 1: **nRMSE (normalized root mean square error) comparison with neural network baselines and CodePDE.** All LLM-based methods (CodePDE and Ours) use GPT-4.1. CodePDE results are obtained under the Reasoning + Debugging + Refinement setting (best of 12).

nRMSE (↓)	Advection	Burgers	React-Diff	CNS	Darcy	Geom. Mean
U-Net	5.00×10^{-2}	2.20×10^{-1}	6.00×10^{-3}	3.60×10^{-1}	–	–
FNO	7.70×10^{-3}	7.80×10^{-3}	1.40×10^{-3}	9.50×10^{-2}	9.80×10^{-3}	9.52×10^{-3}
PINN	7.80×10^{-3}	8.50×10^{-1}	8.00×10^{-2}	–	–	–
ORCA	9.80×10^{-3}	1.20×10^{-2}	3.00×10^{-3}	6.20×10^{-2}	–	–
PDEformer	4.30×10^{-3}	1.46×10^{-2}	–	–	–	–
UPS	2.20×10^{-3}	3.73×10^{-2}	5.57×10^{-2}	4.50×10^{-3}	–	–
CodePDE	1.01×10^{-3}	3.15×10^{-4}	1.44×10^{-1}	1.53×10^{-2}	4.88×10^{-3}	5.08×10^{-3}
Central Difference (Ill-designed)	7.05×10^{12}	1.64×10^{-2}	1.23×10^{-1}	3.85	2.34×10^{-1}	–
Ours	$\mathbf{4.18 \times 10^{-14}}$	$\mathbf{1.79 \times 10^{-5}}$	$\mathbf{8.98 \times 10^{-7}}$	$\mathbf{1.82 \times 10^{-4}}$	$\mathbf{4.84 \times 10^{-13}}$	$\mathbf{9.00 \times 10^{-9}}$

We select 24 representative problems from our 200-PDE benchmark suite, spanning 1D to 5D and covering elliptic, parabolic, and hyperbolic types (full results in Appendix Table 2). Among the 19 problems with explicit analytic solutions, 11 achieve relative L_2 errors of 10^{-6} or better, with Poisson (5.41×10^{-16}) and Helmholtz 2D (3.50×10^{-16}) reaching near machine precision. Biharmonic (6.14×10^{-1}) and 5D Helmholtz (9.8×10^{-1}) are notable failure cases, indicating limited capability on fourth-order and high-dimensional PDEs. End-to-end runtimes fall between 20 and 130 seconds for most problems. Across the tested PDE problems, the average token usage is approximately 33k tokens per problem. A step-by-step walkthrough of the full pipeline on one example problem is provided in Appendix C.

Table 1 compares our method with six neural network baselines, CodePDE, and an ill-designed solver on the five CodePDE benchmark problems; all baseline results are reproduced from Li et al.

(2025). Our method achieves the lowest nRMSE on all five problems, with a geometric mean of 9.00×10^{-9}, approximately six orders of magnitude below CodePDE (5.08×10^{-3}) and the Fourier Neural Operator (FNO, 9.52×10^{-3}). As a reference point, this ill-designed central finite-difference baseline, obtained from an existing online implementation and applied naively without stability safeguards, yields extremely large nRMSE across the five PDEs, reaching 7.05×10^{12} on the advection case. This counterexample highlights the importance of stability-aware plan generation and selection in our pipeline for preventing such ill-designed solvers from being executed. Analysis of the selected schemes across all 24 problems (see Appendix Table 5) reveals a consistent pattern: the Planner Agent selects Fourier spectral methods for periodic-boundary problems, finite difference or finite element methods for Dirichlet-boundary parabolic problems, and Chebyshev spectral methods for Dirichlet-boundary elliptic problems.

4 Conclusion

The Planner and Selector agents embed stability- and consistency-aware numerical reasoning into the generation process, enabling the pipeline to detect and exclude ill-designed or nonphysical solver configurations prior to execution. Through a subsequent coarse-to-fine execution strategy and residual-based self-verification, the system then performs end-to-end solver construction and quality assessment without requiring analytical solutions. Experiments on 24 benchmark PDEs indicate that the framework selects numerical schemes consistent with PDE structural properties (e.g., spectral methods for periodic domains, finite differences for Dirichlet boundaries), and achieves lower error than both neural network baselines and CodePDE on the majority of the CodePDE benchmark problems. The framework still exhibits limited accuracy on high-dimensional ($\geq$5D) and high-order PDEs, and our evaluation covers only regular domains. The system is also coupled to a single LLM (GPT-4.1), and the generated code lacks formal convergence or stability guarantees.

Acknowledgments

The authors were partially supported by the US National Science Foundation under awards IIS-2520978, GEO/RISE-5239902, the Office of Naval Research Award N00014- 23-1-2007, DOE (ASCR) Award DE-SC0026052, and the DARPA D24AP00325-00. Approved for public release; distribution is unlimited.

References

Sören Arlt, Haonan Duan, Felix Li, Sang Michael Xie, Yuhuai Wu, and Mario Krenn. Meta-designing quantum experiments with language models, 2024.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Ricardo Buitrago, Tanya Marwah, Albert Gu, and Andrej Risteski. On the benefits of memory for modeling time-dependent PDEs. In *The Thirteenth International Conference on Learning Representations*, 2025.

Claudio Canuto, M Yousuff Hussaini, Alfio Quarteroni, and Thomas A Zang. *Spectral methods: evolution to complex geometries and applications to fluid dynamics*. Springer Science & Business Media, 2007.

Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in neural information processing systems*, 34:24924–24940, 2021.

Xin He, Liangliang You, Hongduan Tian, Bo Han, Ivor Tsang, and Yew-Soon Ong. Lang-pinn: From language to physics-informed neural networks via a multi-agent framework, 2025. URL https://arxiv.org/abs/2510.05158.

Qile Jiang and George Karniadakis. Agenticsciml: Collaborative multi-agent systems for emergent discovery in scientific machine learning, 2025. URL https://arxiv.org/abs/2511.07262.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

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, 2025. URL https://arxiv.org/abs/2505.08783.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020. URL https://arxiv.org/abs/2010.08895.

Jianming Liu, Ren Zhu, Jian Xu, Kun Ding, Xu-Yao Zhang, Gaofeng Meng, and Cheng-Lin Liu. Pde-agent: A toolchain-augmented multi-agent framework for pde solving, 2025. URL https://arxiv.org/abs/2512.16214.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators, 2019. URL https://arxiv.org/abs/1910.03193.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B. Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. Llm and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. *ArXiv*, abs/2405.09783, 2024.

Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, Francois Lanusse, Mariel Pettee, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. Multiple physics pretraining for spatiotemporal surrogate models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Junhong Shen, Tanya Marwah, and Ameet Talwalkar. UPS: Efficiently building foundation models for PDE solving via cross-modal adaptation. *Transactions on Machine Learning Research*, 2024.

Mauricio Soroco, Jialin Song, Mengzhou Xia, Kye Emond, Weiran Sun, and Wuyang Chen. Pde-controller: Llms for autoformalization and reasoning of pdes. *arXiv preprint arXiv:2502.00963*, 2025.

Shashank Subramanian, Peter Harrington, Kurt Keutzer, Wahid Bhimji, Dmitriy Morozov, Michael Mahoney, and Amir Gholami. Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior. *arXiv preprint arXiv:2306.00258*, 2023.

Xiangru Tang, Bill Qian, Rick Gao, Jiakang Chen, Xinyun Chen, and Mark Gerstein. Biocoder: A benchmark for bioinformatics code generation with large language models, 2024.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning, 2023.

Haoyang Wu, Xinxin Zhang, and Lailai Zhu. Automated code development for pde solvers using large language models, 2025. URL https://arxiv.org/abs/2509.25194.

Yu Zhang, Xiusi Chen, Bowen Jin, Sheng Wang, Shuiwang Ji, Wei Wang, and Jiawei Han. A comprehensive survey of scientific large language models and their applications in scientific discovery, 2024. URL https://arxiv.org/abs/2406.10833.

Jianwei Zheng, LiweiNo, Ni Xu, Junwei Zhu, XiaoxuLin, and Xiaoqin Zhang. Alias-free mamba neural operator. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Lianhao Zhou, Hongyi Ling, Cong Fu, Yepeng Huang, Michael Sun, Wendi Yu, Xiaoxuan Wang, Xiner Li, Xingyu Su, Junkai Zhang, Xiusi Chen, Chenxing Liang, Xiaofeng Qian, Heng Ji, Wei Wang, Marinka Zitnik, and Shuiwang Ji. Autonomous agents for scientific discovery: Orchestrating scientists, language, code, and physics, 2025. URL https://arxiv.org/abs/2510.09901.

O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 2013.

A Related Work

Classical Numerical Methods. Classical numerical analysis remains the foundation for solving PDEs. The finite difference method approximates derivatives using grid-based differences (LeVeque, 2007). The finite element method represents solutions over mesh elements (Zienkiewicz & Taylor, 2013). Spectral methods expand solutions in global basis functions (Canuto et al., 2007). Despite their mathematical rigor, constructing effective solvers typically requires substantial expertise in discretization design and stability verification, motivating interest in automated solver construction.

Neural and Data-Driven PDE Solvers. Scientific machine learning has introduced neural-network-based approaches for approximating PDE solutions, including PINNs (Raissi et al., 2019) and neural operators (Lu et al., 2021; Li et al., 2020). Subsequent work explores Transformers (Cao, 2021), message-passing neural networks (Brandstetter et al., 2022), state-space models (Zheng et al., 2024; Buitrago et al., 2025), and pretrained multiphysics foundation models (Shen et al., 2024; Subramanian et al., 2023; McCabe et al., 2024).

LLMs for Scientific Computing and PDE Automation. Large language models have demonstrated strong capability in generating executable scientific code across chemistry (Bran et al., 2023), physics (Arlt et al., 2024), mathematics (Wang et al., 2023), and computational biology (Tang et al., 2024). Agentic reasoning frameworks extend these capabilities through planning and structured tool interaction (Romera-Paredes et al., 2024; Ma et al., 2024; Jiang et al., 2025; Zhou et al., 2025). FunSearch (Romera-Paredes et al., 2024) demonstrates program search for mathematical structure discovery, while PDE-Controller (Soroco et al., 2025) explores LLM-driven autoformalization for PDE control. Closer to automated PDE solving, neural solver design frameworks construct PINNs via multi-agent reasoning (He et al., 2025; Jiang & Karniadakis, 2025), tool-oriented systems orchestrate libraries such as FEniCS (Liu et al., 2025; Wu et al., 2025), and code-generation paradigms synthesize candidate solvers (Li et al., 2025).

B Full Benchmark Results

Table 2 reports per-problem accuracy and runtime for all 24 benchmark PDEs.

Table 2: Evaluation of proposed framework across 24 benchmark PDEs. The upper block reports relative L_2 error for problems with known analytic solutions; the lower block reports relative residual error.

PDE	Dim	Error	Runtime (s)
Explicit analytic solution available		*(Relative L_2 error)*	
Advection	2	1.13×10^{-13}	29.8
Allen-Cahn	1	2.23×10^{-4}	19.8
Biharmonic	2	6.14×10^{-1}	89.3
Convection Diffusion	2	8.57×10^{-3}	34.6
Euler	1	5.21×10^{-14}	26.0
Heat	1	3.21×10^{-7}	97.4
Heat	2	1.50×10^{-4}	228.1
Helmholtz	2	3.50×10^{-16}	66.3
Helmholtz	5	9.8×10^{-1}	65.8
KdV	1	2.36×10^{-7}	52.2
Laplace	2	1.24×10^{-5}	85.9
Maxwell	3	1.00×10^{-3}	126.1
Navier–Stokes	2	8.08×10^{-6}	64.5
Poisson	2	5.41×10^{-16}	68.9
Reaction Diffusion	2	9.88×10^{-6}	199.5
Schrödinger	1	5.40×10^{-14}	32.2
Shallow Water	1	1.67×10^{-10}	18.5
Vorticity	2	3.32×10^{-4}	54.1
Wave	1	8.34×10^{-10}	73.1
Implicit analytic solution available		*(Relative implicit residual error)*	
Burgers (inviscid)	1	5.65×10^{-4}	23.4
No analytic solution		*(Relative residual error)*	
Burgers (viscous)	1	8.95×10^{-14}	63.1
Cahn–Hilliard	1	9.88×10^{-4}	114.9
Fokker–Planck	2	2.24×10^{-3}	44.3
Gray–Scott	2	1.10×10^{-3}	23.7

C PIPELINE WALKTHROUGH: 2D ADVECTION

We walk through the full pipeline output for 2D Advection ($u_t + c_x u_x + c_y u_y = 0$, periodic BCs, c_x=0.3, c_y=0.2).

Step 1: Planner Agent. The Planner generates 10 candidate schemes spanning spectral, finite difference (FD), finite volume (FV), and finite element (FEM) methods with various time integrators (RK4: classical fourth-order Runge-Kutta; IMEX: implicit-explicit; ETDRK4: exponential time differencing RK4). The Selector Agent scores each based on expected accuracy, stability, and cost. Table 3 lists all candidates.

Table 3: Candidate schemes generated by the Planner Agent and scored by the Selector Agent for the 2D Advection problem.

Plan	**Score**	**Method**	**Rationale (summary)**
Spectral (RK4, high-res)	90	Spectral Fourier	Optimal for smooth periodic advection
FD (WENO3+RK3, high-res)	85	Finite Difference	High-order upwind, conservative
Spectral (ETDRK4, med-res)	80	Spectral Fourier	Good accuracy-cost balance
FV (MUSCL+RK2, med-res)	75	Finite Volume	Limiter controls oscillations
FD (semi-Lagrangian, med-res)	70	Finite Difference	Large time steps, second-order
FEM (IMEX, med-res)	60	Finite Element	Stable but diffusion treatment irrelevant
FD (upwind, low-res)	55	Finite Difference	Stable but first-order accuracy
FD (Crank-Nicolson, med-res)	50	Finite Difference	No upwinding, oscillation risk
FV (upwind, low-res)	50	Finite Volume	Stable but low accuracy
FEM (backward Euler, med-res)	45	Finite Element	Implicit cost unjustified

Step 2: Coder + Critic Agents. The top-5 plans are implemented and executed through the coarse-to-fine pipeline. Table 4 reports the execution results.

Table 4: Execution results for the top-5 candidate schemes on the 2D Advection problem.

Plan	**Residual L_2**	**Runtime (s)**	**Attempts**	**Restarts**
Spectral (RK4, high-res)	1.75×10^{-3}	23.8	2	0
FD (WENO3+RK3, high-res)	3.18×10^{4}	57.5	4	0
Spectral (ETDRK4, med-res)	8.02×10^{-15}	35.3	4	0
FV (MUSCL+RK2, med-res)	1.94×10^{-1}	33.2	2	0
FD (semi-Lagrangian, med-res)	2.27×10^{-2}	15.8	2	0

Step 3: Final Selection. The Selector Agent chooses **Spectral (ETDRK4, med-res)** based on its residual of 8.02×10^{-15} (near machine precision) at a moderate runtime of 35.3 s. The high-resolution spectral plan, despite scoring highest in planning, produces a larger residual (1.75×10^{-3}), likely due to time-stepping error at coarser Δt. The FD plan diverges entirely (residual 3.18×10^{4}). This example illustrates how the pipeline's evaluate-then-select strategy can override initial scoring when execution results differ from expectations.

D SCHEME SELECTION RESULTS

Table 5 lists the numerical scheme automatically selected by the Planner Agent for each benchmark PDE. The schemes are grouped by boundary condition type. For periodic-boundary problems, the pipeline consistently selects Fourier spectral methods. For Dirichlet-boundary parabolic problems, finite difference (FD) or finite element methods (FEM) with implicit time stepping are preferred. For Dirichlet-boundary elliptic problems, Chebyshev spectral methods are selected.

Table 5: Numerical schemes selected by the Planner Agent for each benchmark PDE.

PDE	**Dim.**	**BC**	**PDE Type**	**Selected Scheme**
Periodic boundary conditions				
Advection	2	Periodic	Hyperbolic	Spectral Fourier (RK4)
Convection Diffusion	2	Periodic	Parabolic	Spectral Fourier (IMEX)
Schrödinger	1	Periodic	Dispersive	Spectral Fourier (Split-Step)
Navier–Stokes	2	Periodic	Parabolic	FEM (IMEX)
Shallow Water	1	Periodic	Hyperbolic	FD (explicit)
Dirichlet boundary conditions, parabolic				
Allen-Cahn	1	Dirichlet	Parabolic	FD (Crank-Nicolson)
Burgers (viscous)	1	Dirichlet	Parabolic	FD (implicit)
Heat	1	Dirichlet	Parabolic	FEM (Crank-Nicolson)
Heat	2	Dirichlet	Parabolic	FD (Crank-Nicolson)
Reaction Diffusion	2	Dirichlet	Parabolic	FD (IMEX)
Dirichlet boundary conditions, elliptic				
Helmholtz	2	Dirichlet	Elliptic	Spectral Chebyshev
Laplace	2	Dirichlet	Elliptic	Spectral Chebyshev
Poisson	2	Dirichlet	Elliptic	Spectral Chebyshev
Dirichlet boundary conditions, hyperbolic				
Wave	1	Dirichlet	Hyperbolic	Spectral (explicit)

AI USAGE

This work used large language models for language polishing, formatting assistance, and limited code suggestions.