AN AGENTIC AI SCIENTIFIC COMMUNITY FOR AUTOMATED NEURAL OPERATOR DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

We present an agentic approach to autonomous neural operator discovery based on an *AI scientific community*, which consists of a swarm of virtual laboratories that interact under a citation-based economy of influence. Highly-cited labs found new labs that follow their research direction and replace non-performing labs. Each virtual lab contains three agents: an LLM planner that writes a complete PyTorch program for a neural operator, a numerical worker that checks, repairs and trains it, and an LLM reviewer that participates in cross-lab peer review. The labs either compose neural operator building blocks (DeepONet/branch-trunk, Fourier, Transformer/attention, wavelet, and residual convolutional) in code, or write every layer from primitives. Simulation campaigns on the Darcy flow, Navier-Stokes and Allen-Cahn 2D problems show that the AI scientific community discovers operators that are more accurate than strong baselines on some problems, while no operator is the best on every problem. We also conduct comprehensive ablation studies to isolate the effects of using a community of labs versus a single lab, of allowing coding versus using a fixed vocabulary of blocks, and of using LLM agents versus rule-based alternatives, the latter on piecewise regression, the linear advection and Burgers 1D PDEs, and the Navier-Stokes and Darcy flow 2D PDEs. The code and full results are available at https://anonymous.4open.science/r/neural-operator-community.

1 INTRODUCTION

Operator learning has become a central tool for building fast surrogates of partial differential equation (PDE) solvers (Li et al., 2021; Lu et al., 2021; Kovachki et al., 2023). However, each operator family encodes a fixed inductive bias: global spectral convolutions for the Fourier Neural Operator (FNO) (Li et al., 2021), a branch-trunk decomposition for DeepONet (Lu et al., 2021), multiresolution bases for wavelet operators (Tripura & Chakraborty, 2023), and attention for transformer-style operators (Cao, 2021; Hao et al., 2023). In addition, selecting the architecture hyperparameters, such as depth, width, and block composition, for a new PDE is still done by hand. This raises a natural question:

> *Can a simulation of a neural operator research community consisting of automated agentic virtual labs discover the right neural operator architecture for a given problem with no human prior or intervention?*

To answer this question, we employ an *AI scientific community* framework, which combines agentic AI with swarm intelligence to create decentralized networks of virtual laboratories (Braga-Neto, 2026). In this paradigm, each particle in the swarm represents a complete virtual laboratory instance, enabling collective scientific exploration that mirrors real-world research communities. This framework leverages the inherent properties of swarm intelligence, namely, decentralized coordination, balanced exploration-exploitation trade-offs, and emergent collective behavior. The swarm of virtual laboratories explores the design space under a citation-based economy, where well-cited labs gain influence and breeding priority while uncited labs are replaced.

In our neural operator AI scientific community (Figure 1), each virtual lab has an LLM planner, a numerical worker and an LLM reviewer, and the citations of the reviewer decide which research directions spread and which labs are refounded. The planner writes the code of its operator, a common capability of LLM-based systems (Romera-Paredes et al., 2024; Novikov et al., 2025; Chen et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

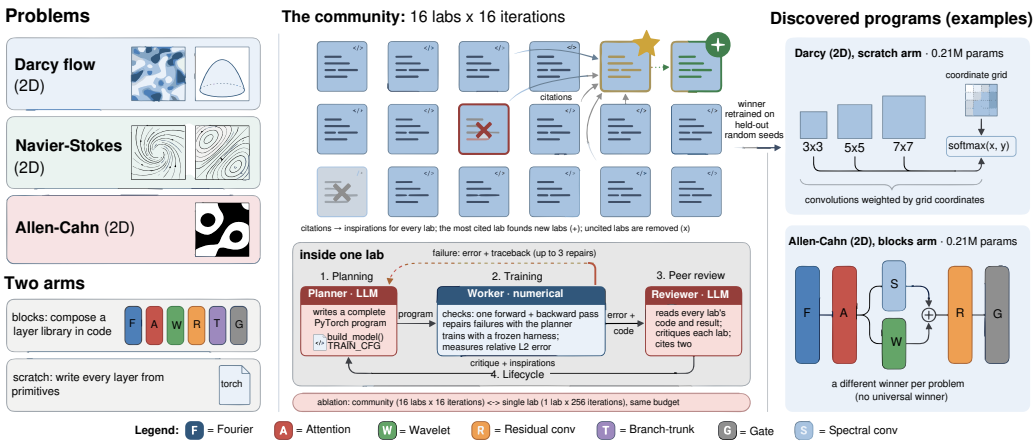


Figure 1: The neural operator AI scientific community: the three problems and the two arms (left), a community of 16 labs and the four phases of each iteration inside a lab (center), and two discovered programs (right). Every winning program is retrained on held-out random seeds before it is compared with other models.

2023) that frees the labs from a fixed vocabulary of blocks: in the *blocks* arm the labs compose neural operator building blocks in code, and in the *scratch* arm they write every layer from primitives, while the worker checks, repairs and trains every program with a frozen harness. We measure what the community adds by comparing it with a single lab, what coding adds with an ablation study in which the labs compose a genome from the fixed vocabulary instead, and what the language model adds by replacing the LLM agents by rule-based alternatives. Since the error that a search records for its winner is biased low, we retrain every winning program on random seeds that played no part in selecting it and compare it with tuned baselines.

Main Contributions.

1. **An agentic framework for neural operator discovery.** We present an AI scientific community for neural operator architecture search, with LLM planner and reviewer agents and a numerical worker, whose labs interact under a citation-based economy and write the code of their operators. Measured on held-out random seeds, the discovered operators are 2.6 times more accurate on the Darcy problem than the hybrid architecture discovered without coding, and beat every baseline on the Allen-Cahn problem.
2. **Ablation studies of the community, of coding and of LLM agency.** When the labs write every layer from primitives, a single lab loses to a community of 16 labs on all five random seeds, with 2.9 times its error, and each role of the citations recovers most of the difference on its own. In the ablation without coding, on five problems, the community with LLM agents proposes parameter-lean multi-family hybrids, whereas the community in which rule-based agents replace the LLM planners and reviewers tends to collapse around canonical families under an identical training budget.
3. **A no-free-lunch result for neural operators.** The results show that no operator is universal. Measured on held-out random seeds against tuned baselines, a tuned FNO is the best model on the Darcy problem, the programs that the labs write are the best on the Allen-Cahn problem, and their searches on the Navier-Stokes problem are unreliable. Since the error recorded during a search understates the error of its winning program on new random seeds by up to 2.0 times, we retrain the winning programs on held-out random seeds.

2 Related Work

Neural operators. FNO (Li et al., 2021) learns a convolution kernel in Fourier space, whereas DeepONet (Lu et al., 2021) separates input-function encoding (branch) from query-location encoding

(trunk); other families use wavelets with multiresolution bases (Tripura & Chakraborty, 2023), attention (Cao, 2021; Hao et al., 2023), residual convolutions (Raonic et al., 2023), U-shaped multiscale architectures (Rahman et al., 2023) and geometry-adapted designs (Li et al., 2023b). Rigorous comparisons find that each family wins in different regimes (Lu et al., 2022; Takamoto et al., 2022), suggesting that no neural operator architecture dominates uniformly over all problems, a fact that the neural operator AI scientific community rediscovers in a much broader context.

Architecture search. Neural architecture search (NAS) automates model design via reinforcement learning, evolutionary search, gradient-based relaxations and many other strategies (Zoph & Le, 2017; Real et al., 2019; Liu et al., 2019; Elsken et al., 2019), and a more radical line evolves learning algorithms from primitive operations rather than selecting among predefined blocks (Real et al., 2020). Random search is a strong baseline (Li & Talwalkar, 2019; Yu et al., 2020; Yang et al., 2020; Lindauer & Hutter, 2020), and selecting the best of many models on the data used to score them biases the score (Varma & Simon, 2006; Cawley & Talbot, 2010; Bouthillier et al., 2021), which has also been documented for AI scientist systems (Luo et al., 2025; Toledo et al., 2025; Xu et al., 2026; Ning et al., 2026).

LLMs as optimizers and code writers. Large language models have been used as black-box optimizers (Yang et al., 2024), as proposal engines for neural architecture search (Zheng et al., 2023b; Nasir et al., 2024), and as the mutation operator of evolutionary searches over programs (Romera-Paredes et al., 2024; Lehman et al., 2022; Novikov et al., 2025; Assumpção et al., 2025); closest to our planner is EvoPrompting (Chen et al., 2023), which proposes code-level architectures inside an evolutionary loop. Simple baselines are competitive with code evolution (Gideoni et al., 2026; Oved et al., 2026). In scientific machine learning, LLM agents coordinate physics-informed and operator-learning tasks (Jiang & Karniadakis, 2026), write PyTorch neural operators (Song et al., 2026) or PDE solvers (Li et al., 2026), and search domain-specific languages for neural PDE solvers (Kong et al., 2026). In our community, programs are selected and shared through the citations of a reviewer, and the contributions of the community, of coding and of the language model are isolated by ablation.

Multi-agent LLM systems. *Virtual labs*, teams of LLM agents with distinct roles, solve tasks in conversational frameworks (Wu et al., 2024; Li et al., 2023a), emulate software companies and operating procedures (Qian et al., 2024; Hong et al., 2024), and drive research agents (Lu et al., 2026; Schmidgall et al., 2025; Gottweis et al., 2026) and numerical algorithm design (Toscano et al., 2025), often with an LLM as judge (Zheng et al., 2023a). The AI scientific community (Braga-Neto, 2026) takes this paradigm one level up with a swarm of virtual labs bound by citations. Several studies find that many agents often do no better than one at matched cost (Cemri et al., 2025; Wang et al., 2024; Tran & Kiela, 2026; Jwalapuram et al., 2026); our results agree for labs that compose a fixed vocabulary of blocks, and identify a case where the community does help: labs that write every layer from primitives.

3 Problem Formulation

Operator learning. Given a PDE with solution operator $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ mapping an input function $a \in \mathcal{A}$ (e.g. an initial condition or coefficient field) to the solution $u = \mathcal{G}(a) \in \mathcal{U}$, we seek a parametric surrogate $\mathcal{G}_\theta$ minimizing the expected relative L_2 error

$$\mathcal{L}(\theta) \;=\; \mathbb{E}_{a\sim\mu}\frac{\|\mathcal{G}_\theta(a) - \mathcal{G}(a)\|_2}{\|\mathcal{G}(a)\|_2}, \tag{1}$$

estimated on sampled input-output pairs. The relative L_2 error is our primary accuracy metric throughout.

Architecture search space. An architecture is a *program* p: a Python file that defines a function `build_model()`, which returns a PyTorch module mapping input fields of shape $(B, 1, 32, 32)$ to output fields of the same shape, and a dictionary `TRAIN_CFG` with a learning rate and a weight decay. The discovery problem is the bilevel program

$$p^\star \;=\; \arg\max_{p}\; F\big(p, \theta^\star(p)\big), \qquad \theta^\star(p) \;=\; \arg\min_{\theta}\; \widehat{\mathcal{L}}_{\text{train}}\big(\theta; p\big), \tag{2}$$

where the inner problem is ordinary gradient-based training of a fixed architecture by a frozen harness, and the outer problem, which the swarm of virtual labs solves, searches for the program of highest fitness F (Section 4). In the ablation without coding, an architecture is instead a *genome* $g = (b_{1:L},\, c,\, m,\, \sigma,\, \dots)$, where $b_{1:L}$ is a sequence of $L \in [2, 8]$ blocks drawn from a vocabulary

$$\mathcal{B} = \{\text{BRANCH_TRUNK}, \text{FOURIER}, \text{ATTENTION}, \text{WAVELET}, \text{RESIDUAL_CONV}\}, \tag{3}$$

c is the hidden width, m the number of spectral modes, σ the activation, plus gating, skip connections, dropout, and learning rate. A genome compiles to a concrete neural operator, and the outer problem searches genomes in the same way.

4 METHODOLOGY

AI scientific community framework. A swarm of N virtual labs explores the space of programs for T iterations. Each lab holds its current program, its history and the citations it has received. Each iteration consists of four phases (Figure 1):

1. *Planning.* The planner of each lab writes a new program.
2. *Training.* The worker checks the program, repairs it with the planner if it fails, and trains it with a frozen harness, returning the measured relative L_2 error.
3. *Peer review.* An LLM reviewer reads the programs and results of all labs, critiques each lab and cites the two labs whose work the community should build on.
4. *Lifecycle.* The cited programs are shown to the other labs as inspirations, and labs that are not cited are replaced by new labs founded from the most cited program.

A trained program with test error e and P parameters receives the fitness

$$F \;=\; -\log_{10} \max(e, 10^{-6}) - \max\!\left(0,\, \tfrac{P}{2\cdot 10^{6}} - 1\right), \tag{4}$$

so a program above two million parameters is penalized rather than rejected. When the budget is spent, the trained program with the highest fitness is the winner.

Individual virtual labs. Each virtual lab consists of three agents. The **planner** is an LLM prompted as a research scientist. Its prompt describes the task, the fitness, what the program may use and how to keep tensor shapes consistent, and contains the lab's last three attempts with their errors or failure messages, its current program, the reviewer's last critique, and up to two programs from other labs, which we call inspirations. The planner makes one testable change, states it as a hypothesis and writes the complete new program. The **worker** is deliberately numerical and not an LLM, so that the fitness signal is consistent and comparable across labs and iterations. It builds the model and passes a batch of random inputs forward and backward before training; if this check or the training fails, the planner receives the error and the end of the traceback and is asked for the smallest fix, up to three times. If every attempt fails, the proposal is recorded as failed, and the failed program still becomes the lab's current program, the one its planner sees next. The **reviewer** reads the latest error, hypothesis and code of every lab, writes a critique of each lab, which goes into that lab's next prompt, and cites two labs.

Lifecycle. The inspirations shown to a lab are the two most cited programs of other labs in the previous review. A lab that has not been cited in its last two reviews dies, unless it holds the best program so far or is the most cited lab, and a new lab takes its place, starting from a copy of the most cited program with the instruction to diverge from it. Across the 49 simulations of 16 labs with this rule, 11.4 labs die after each review on average, so most of the community is refounded every round from the programs that the reviewer cites.

Two arms. In the *blocks* arm, the labs compose models in code from a layer library with the neural operator building blocks: a Fourier layer (Li et al., 2021), self-attention over the grid points (Vaswani et al., 2017), a branch-trunk layer after DeepONet (Lu et al., 2021), a two-scale layer in the spirit of wavelet operators (Tripura & Chakraborty, 2023), a residual convolution and a gate, starting from a three-layer FNO with 0.89M parameters. In the *scratch* arm, the labs implement every layer from `torch` primitives, starting from six residual convolutions with 0.90M parameters; the prompt says that re-implementing a textbook FNO, DeepONet, U-Net or Transformer does not count as a discovery, and the reviewer caps the score of such re-implementations.

Held-out evaluation. A search selects its winner on the test split of its own random seed, which biases the recorded error low (Cawley & Talbot, 2010). We therefore retrain every winning program from scratch on random seeds that played no part in selecting it and report the mean, train every baseline with the same harness at the best of three learning rates, and compare configurations with an exact sign-flip test on paired log ratios.

Community ablations. To measure what the scientific community adds, a single lab runs for 256 iterations against 16 labs for 16 iterations, with the same planner, reviewer and number of proposals, but no inspirations and no deaths. Two variants separate the roles of the citations: a single lab that returns to its last working program after each failed proposal (reset), and a community whose labs never die, in which the citations still choose the inspirations but no longer decide which labs survive. In an accuracy-first variant only the reviewer's instructions change: between two labs, the one with the lower measured error ranks higher.

Ablation without coding. To measure what coding adds, the labs compose a genome from the vocabulary $\mathcal{B}$ instead of writing a program (Algorithm 1 in Appendix L). The planner queries an LLM for the next genome, given its own history, the global best and a summary of peer labs and their measured fitness, and each lab's reviewer casts soft votes over a sample of peers, which accumulate as citation-like influence; influential labs persist and breed, and uncited labs are replaced. The fitness of lab i is the composite score

$$F_i \;=\; w_a\, A_i + w_g\, G_i + w_e\, E_i + w_n\, \nu_i\,, \qquad (w_a, w_g, w_e, w_n) = (0.70, 0.20, 0.05, 0.05), \tag{5}$$

where A_i is an accuracy score derived from the relative L_2 error, G_i rewards robustness to input noise, E_i rewards parameter parsimony, and ν_i is an architectural-novelty score; the fitness is zeroed below an accuracy floor, and the weights are fixed a priori and were not tuned. Each candidate is trained at a reduced "screening" fidelity (small data, few epochs), so that a single operator trains in seconds.

LLM ablation study and agent auditing. Without coding, the planner and reviewer roles can also be filled by rule-based agents, which lets us run a clean ablation study of LLM agency: the rule-based planner uses explore/exploit particle-swarm updates (Kennedy & Eberhart, 1995), and the rule-based reviewer uses accuracy-proportional voting, with the same swarm size, iteration count, training budget, fitness, and random seeds. Since a language model can silently degrade to a rule-based fallback (e.g. on a transient error or malformed output), we log every LLM call, namely, system prompt, user prompt, raw response, role, and parse status, and audit each run's log for planner/reviewer fallbacks.

Language models. The planner, the repair calls and the reviewer use Qwen2.5-Coder 7B (Hui et al., 2024) and, in one comparison, Qwen3-Coder-30B-A3B (Yang et al., 2025; Qwen Team, 2025); the ablation without coding uses `gemma3:12b`. All models are served locally with Ollama (Ollama, 2026), with no paid API.

5 Results

Experimental setup. We evaluate the community on three 2D problems on the unit square, discretized on 32×32 grids, with 256 training and 64 test samples per random seed (Appendix A). In the Darcy flow problem, $-\nabla \cdot (a \nabla u) = 1$ with $u = 0$ on the boundary, and the model maps the log-permeability $\log a$ to u. In the Navier-Stokes problem, the model maps the vorticity of a periodic flow with viscosity 10^{-2} and a fixed forcing from $t = 0$ to $t = 4$. In the Allen-Cahn problem, $\partial_t u = \varepsilon^2 \Delta u + u - u^3$ with $\varepsilon = 0.1$ and periodic boundaries (Allen & Cahn, 1979), and the model maps u from $t = 0$ to $t = 4$. Every program is trained by one frozen harness: AdamW, a cosine schedule over 80 epochs, batches of 16 and mean squared error. A community has 16 labs for 16 iterations (256 proposals) unless stated otherwise. Each simulation runs with one search seed, 42, 137 or 2024, and the main comparisons also with 7 and 99; the winning program is retrained on the seeds among 42, 137 and 2024 that differ from its search seed. The baselines are the FNO (Li et al., 2021) at width 32 (1.19M parameters) or 64 (4.74M, above the cap), a U-Net (Ronneberger et al., 2015), DeepONet (Lu et al., 2021), and the hybrid architecture discovered on Darcy by the community without coding (1.26M parameters), also widened to 1.97M parameters (Appendix C); random search samples 256 programs per search seed from the block vocabulary (Li & Talwalkar, 2019). The

Table 1: Test relative L^2 error (mean $\pm$ standard deviation). Baselines are trained once per seed at their best learning rate in $\{5 \cdot 10^{-4}, 10^{-3}, 2 \cdot 10^{-3}\}$. Searched rows report the winning program retrained on seeds not used to select it, searched on the problem of each column; the transferred rows retrain programs discovered on Darcy unchanged. Searched rows average over five search seeds, except the 1024-proposal community and the accuracy-first community on Darcy (three). Budgets count proposals. Bold: lowest error per column within the 2M parameter cap; †exceeds the cap.

Model	Params	Darcy	Navier-Stokes	Allen-Cahn
Baselines, no search				
FNO, width 32	1.19M	**0.0201 $\pm$ 0.0020**	0.0330 $\pm$ 0.0032	0.1462 $\pm$ 0.0199
FNO, width 64†	4.74M	0.0154 $\pm$ 0.0006	0.0293 $\pm$ 0.0022	0.1388 $\pm$ 0.0109
U-Net	1.93M	0.0741 $\pm$ 0.0039	0.0402 $\pm$ 0.0005	0.0926 $\pm$ 0.0084
DeepONet	0.60M	0.0612 $\pm$ 0.0044	0.1668 $\pm$ 0.0190	0.2924 $\pm$ 0.0056
Hybrid (fixed vocabulary)	1.26M	0.0719 $\pm$ 0.0061	0.0287 $\pm$ 0.0016	0.0661 $\pm$ 0.0132
Hybrid, widened	1.97M	0.0630 $\pm$ 0.0055	0.0294 $\pm$ 0.0018	0.0629 $\pm$ 0.0125
Search seed program	0.89M	0.3133 $\pm$ 0.0279	0.6557 $\pm$ 0.0210	0.0999 $\pm$ 0.0104
Searched: arm or method, proposals, reviewer				
Random search, 256		0.0640 $\pm$ 0.0041	n/a	n/a
Blocks, 256		0.0525 $\pm$ 0.0286	n/a	n/a
Scratch, 256		0.0554 $\pm$ 0.0405	n/a	n/a
Blocks, 1024, acc.-first		0.0275 $\pm$ 0.0046	n/a	n/a
Blocks, 256, acc.-first		0.0370 $\pm$ 0.0101	0.0578 $\pm$ 0.0469	**0.0513 $\pm$ 0.0069**
From the FNO, 256, acc.-first		0.0215 $\pm$ 0.0040	n/a	n/a
Transferred, blocks	1.68M	source	0.0326 $\pm$ 0.0012	0.0731 $\pm$ 0.0112
Transferred, acc.-first	1.91M	source	**0.0253 $\pm$ 0.0044**	0.1183 $\pm$ 0.0220
Transferred, scratch	1.66M	source	0.0508 $\pm$ 0.0047	0.1533 $\pm$ 0.0089

ablation without coding considers five problems, namely piecewise regression, the linear advection and Burgers 1D PDEs, and the Navier-Stokes (in vorticity form with Kolmogorov forcing) and Darcy flow 2D PDEs, with the random seeds 42, 137 and 2024 in both the LLM and the rule-based configuration ($5 \times 2 \times 3 = 30$ runs) and $N = 16$ labs for $T = 20$ iterations (Appendix L).

5.1 What the virtual labs discover

The Darcy problem. Table 1 compares the winning programs, retrained on held-out random seeds, with baselines trained once per seed. The best configuration of the community, 16 labs over 1024 proposals with the accuracy-first reviewer, reaches 0.0275 relative L_2 error, 2.6 times more accurate than the hybrid architecture discovered without coding (0.0719), and the community with 256 proposals reaches 0.0525. The best winning program of our 93 Darcy simulations that start from the seed program reaches 0.0218 (Appendix G). The most accurate model within the parameter cap, however, is the tuned FNO (0.0201). Started from the tuned FNO instead of the seed program, the community improves on it in one of five simulations, with a program that adds a fifth Fourier layer, a change of depth rather than a new mechanism; on average, this configuration (0.0215) does not improve on the FNO.

The Allen-Cahn and Navier-Stokes problems. On the Allen-Cahn problem, the community that searches on that problem reaches 0.0513, and all five simulations end below every baseline, including the widened hybrid discovered without coding (0.0629); the FNO is one of the weakest models on this problem (0.1462). The gain is modest, a factor of 1.2, and it does not require large models: one winning program has 207,585 parameters and an error of 0.0510. On the Navier-Stokes problem the simulations are unreliable: two of five end below the hybrid discovered without coding (0.0287), while two stall far above it (0.0669 and 0.1360), so that the mean (0.0578) is 2.0 times the error of the hybrid.

No universal winner. Programs discovered on the Darcy problem do not carry over reliably (Appendix D): retrained unchanged on Navier-Stokes, only one of three beats the hybrid discovered without coding on average, and on Allen-Cahn two of the three end worse than the seed program. There is no universal winner: the best models are the tuned FNO on the Darcy problem, the programs

Table 2: Community (16 labs $\times$ 16 iterations), single lab (256 iterations) and random search over the block vocabulary (256 sampled programs) on Darcy at 256 proposals: held-out relative L^2 error of each winning program by search seed, and mean number of trained proposals. With reset, the single lab returns to its last working program after each failure; without deaths, no lab is ever refounded.

Method	42	137	2024	7	99	Mean	Trained
Blocks arm							
Community	0.0229	0.0724	0.0414	0.0342	0.0916	0.0525	242
Single lab	0.0447	0.0521	0.1148	0.0583	0.0459	0.0631	223
Random search	0.0617	0.0628	0.0693	0.0669	0.0593	0.0640	256
Scratch arm							
Community	0.1264	0.0252	0.0449	0.0362	0.0445	0.0554	198
Community, no deaths	0.0801	0.0434	0.0730	0.0530	0.1004	0.0700	179
Single lab	0.2548	0.1077	0.1408	0.0489	0.2496	0.1604	19
Single lab, reset	0.0503	0.0486	0.0893	0.0433	0.1112	0.0685	125

that the community discovered on the Allen-Cahn problem, and a program transferred from Darcy together with the hybrid discovered without coding on the Navier-Stokes problem. The discovered programs do not win by being larger: widening the hybrid to 1.97M parameters lowers its Darcy error only slightly (0.0630), and retrained for 300 epochs instead of 80, the discovered programs stay 2.0 times more accurate than the hybrid (Appendix F).

Selection bias. The error that a simulation records for its winning program understates its error on new random seeds by a factor of 1.1 in the median simulation and up to 2.0. It also makes a larger budget look more useful than it is: from 256 to 1024 proposals the recorded error falls by a factor of 1.57, the held-out error by only 1.25 (Appendix H).

5.2 Ablation study: when does the scientific community help?

Table 2 compares, at the same budget of 256 proposals on the Darcy problem, a community of 16 labs run for 16 iterations with a single lab run for 256 iterations, which has the same planner, harness and selection rule but no other labs to learn from.

The scratch arm. When the labs write every layer from primitives, the community has the lower error on all five random seeds, 0.0554 against 0.1604 on average, a factor of 2.9 ($p = 0.0625$, the smallest value attainable with five pairs). A proposal that still fails after three repair attempts is never trained, yet it becomes the program its lab builds on next. The communities train 198 programs per simulation and never go through an iteration in which no program trains, while the single lab trains 19 (Figure 2 in Appendix E): after its last trained program, it spends between 84 and 215 consecutive iterations proposing programs that fail, mostly with type errors and tensor shape mismatches. The single lab also makes more LLM calls (1232 per simulation against 463), so counted in calls the comparison favours it.

What the citations do. The citations act in two ways: they choose the programs that every lab sees as inspirations, and through the lifecycle they decide which labs survive. A lab whose program no longer trains is rarely cited (11 of the 150 citations in the scratch communities went to a lab whose latest proposal had failed), so it dies within one or two reviews and is replaced by a lab founded from a cited program that works, while the single lab never dies and keeps building on its broken program. A single lab that returns to its last working program after each failure reaches 0.0685, closing 80% of the gap to the community on a log scale, and a community whose labs never die reaches 0.0700, closing 78%. Each role thus recovers most of the difference on its own, and the full community, which has both, beats the single lab with reset on four of five seeds and the community without deaths on four (by factors of 1.24 and 1.26, $p = 0.5$ and 0.1875).

The blocks arm. When the labs compose the building blocks in code, almost every proposal trains (242 and 223 per simulation), and the difference in error is small and not consistent: the community wins on three of five seeds, with a ratio of means of 1.2 ($p = 0.5625$). Random search over the same

Table 3: Ablation study of the LLM agents, in the community without coding. Discovered relative L_2 error on the clean data at screening fidelity and parameter count, LLM vs. rule-based (PSO) agents, mean±std over the three simulations.

Problem	Discovered rel. L_2 (clean)		Params	
	LLM (Gemma 3)	PSO	LLM	PSO
Piecewise Reg. (1D)	0.0417±0.0011	0.0405±0.0007	487K±258K	198K±95K
Advection (1D)	0.0159±0.0038	0.0183±0.0007	273K±157K	456K±159K
Burgers (1D)	0.0799±0.0067	0.0709±0.0047	472K±415K	197K±103K
Navier-Stokes (2D)	0.0009±0.0005	0.0007±0.0000	2.2M±613K	5.6M±4.1M
Darcy (2D)	0.0095±0.0003	0.0096±0.0015	1.4M±403K	966K±593K

vocabulary reaches 0.0640 on average, already below the hybrid discovered without coding (0.0719); the community is better on average, by a factor of 1.22, but only on three of five seeds. With the building blocks, neither role of the citations has much to add.

5.3 Ablation study: what happens without coding?

Without coding, every LLM winner is a multi-family hybrid, and the mixture is regime-appropriate; e.g., the planners tend to lead the stacks with Fourier blocks in the spectral-friendly advection, Burgers and Navier-Stokes problems (Appendix M). Retrained in the same harness as the programs (Table 1), the hybrid discovered on Darcy reaches 0.0719, 2.6 times the error of the best configuration with coding, every simulation with coding on Allen-Cahn beats it, and on Navier-Stokes it remains among the best models (0.0287). The best programs of the blocks arm use the same building blocks, but wire them into graphs with parallel paths, gates and residual connections (Appendix G), where a genome stacks them in sequence. The two searches also differ in the language model, the fitness, the review rules and the screening budget; in the same harness and at the same budget, random search over genomes of the same vocabulary reaches 0.0640 on Darcy (Table 2).

5.4 Ablation study: what do LLM agents contribute?

Table 3 reports the discovered error and parameter count per problem under LLM agents and rule-based alternatives (Figures 7 and 8 in Appendix P compare errors and convergence). Because both conditions train identical operators under an identical budget, any difference is attributable to the planner/reviewer policy. The screening accuracies are statistically indistinguishable on all five problems (overlapping within 1 s.d.), so we do not claim an accuracy win for LLM agency. The measurable difference is architectural: the LLM discovers parameter-lean multi-family hybrids where the rule-based condition tends to collapse to canonical, nearly single-family designs, e.g., pure-Fourier stacks on Navier-Stokes and wavelet-dominated stacks on Darcy and piecewise regression. On Navier-Stokes the LLM hybrids match the pure-spectral rule-based accuracy with $2.5\times$ fewer parameters on average (2.2M vs. 5.6M).

Across 4,817 planner calls the LLM chose HYBRIDIZE in 99.8% of decisions (the remainder EXPLORE; never EXPLOIT), and its rationales condition on the summarized swarm state, e.g. "*several labs are converging on FNO and transformer architectures… I'll introduce a hybrid approach by incorporating attention blocks*" (Navier-Stokes problem, random seed 42). This action bias, partly prompt-induced (Appendix O), is the mechanism behind the hybrids. Across 4,804 scoreable review calls, the Spearman correlation between the reviewer's votes and the candidates' measured accuracy is $\bar{\rho} = 0.64$ (positive in 88% of calls), so peer review is grounded in the fitness signal, with a diversity tilt: the rationales cite accuracy in 97% of calls, parameter efficiency in 83%, robustness in 71%, and novelty in 26%. The audit of all 9,623 LLM calls finds zero fallbacks to rule-based decisions (Table 15 in Appendix P).

6 Computational Cost

A simulation of the community with 256 proposals takes 2.6 hours in the median on one GPU, including the language model, and the 117 simulations reported in this paper trained 21,332 programs

in 310 hours of wall-clock time (Appendix K). In the ablation without coding, the LLM and rule-based communities train identical operators under an identical budget, so the difference in time per lab evaluation is the overhead of LLM-driven planning and review: roughly 14 s in the 1D problems and 25-33 s in the 2D problems, a 2-4$\times$ increase in total wall-clock time over rule-based coordination (Appendix P).

7 Limitations

Tool use and literature priors. The planner writes programs that the worker executes, but the LLM does not call external tools; fully-agentic virtual labs as in Lu et al. (2026) are left to future work. The LLM has also read the operator-learning literature and may "know", for example, that FNO is suitable for periodic, spectral-friendly problems, which is both a feature (informed priors) and a confounding factor; the rule-based agents provide the prior-free reference point.

Scale and statistics. We use open models of 7B to 30B parameters (Appendix I), and the problems are small (32×32 grids, 256 training samples, 80 epochs), which made 117 simulations affordable on one workstation but may favour different models than larger problems would. With five paired random seeds the smallest attainable p-value is 0.0625, and LLM planning is stochastic, so we report independent replicates rather than claiming bit-for-bit reproducibility.

Ablation design. The community without coding also differs in the language model, the fitness, the review rules and the screening budget. The ablation of LLM agency switches planner and reviewer together, and we separated the two roles of the citations but did not replace the reviewer's choices by random ones; disentangling these factors requires factorial designs.

8 Conclusion

We presented an agentic LLM-driven neural operator AI scientific community. Each virtual lab in the community consists of a planner, a worker, and a reviewer: the planner writes the code of an operator, the worker checks, repairs and trains it, and the citations of the reviewer decide which programs spread and which labs are refounded. Measured on held-out random seeds against tuned baselines, the virtual labs discover operators that are 2.6 times more accurate on the Darcy problem than the hybrid architecture discovered without coding and that beat every baseline on the Allen-Cahn problem, while their searches on Navier-Stokes are unreliable and a tuned FNO remains the best model for Darcy: there is no universal winner. When the labs write every layer from primitives, the citation economy is what keeps the search going. In the ablation without coding, whose 9,623 LLM calls are logged and audited, the LLM agents maintain diversity, while rule-based agents collapse to pure stacks. These conclusions rest on scoring every winning program on held-out random seeds and on tuning the baselines, two cheap practices that we recommend for any comparison of automated model search with fixed models. Future work will include frontier LLM backends, larger problems, fully autonomous agents, more roles per virtual lab, a factorial role ablation study, and a reviewer that cites at random.

Ethics statement

This paper studies the automated design of neural operators by language-model agents; it involves no human subjects and no personal data. The main risk we see in this line of work is misplaced trust in automated results: the error that a search records for its winning architecture can understate its error on new random seeds, and an untuned baseline can produce a false positive. We report held-out errors and tuned baselines for this reason. Programs written by a language model run as ordinary processes with a time limit but without a sandbox; a system of this kind should isolate the code it executes.

Reproducibility statement

The code of the community and of the ablation without coding, their configurations and logged results, and the winning program of every simulation with coding are available at `https://anonymous.4open.science/r/neural-operator-community`; the complete per-run

LLM transcripts of the ablation without coding, from which the agent audit (Table 15) is reproducible, will be released with the final version. The language models (Qwen2.5-Coder 7B, Qwen3-Coder-30B and `gemma3:12b`) run locally through Ollama, with no paid API. The prompts are summarized in Appendices B and O, and the data generators, the harness and the seeds are given in Appendix A. Every number reported for the simulations with coding is produced by a script from the logged results.

AI USE STATEMENT

Language models are the object of this study: the planners, repair calls and reviewers of the community are the open models described in Section 4, run locally. We have used AI tools to help create and edit code and scripts, to help run and analyze experiments under our direction, to search for related work, to understand code, and to draft and polish sections of the text. We have not used generative AI tools to generate data, which come from the numerical solvers of Appendix A, and the tasks that concern mathematical claims and proofs are not applicable to this work. We have reviewed all AI-assisted work: every result in the paper is produced by a script from the logged results, and every reference was checked against its source. We take responsibility for all the work produced with the aid of generative AI.

REFERENCES

Samuel M. Allen and John W. Cahn. A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening. *Acta Metallurgica*, 27(6):1085–1095, 1979. doi: 10.1016/0001-6160(79)90196-2.

Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. CodeEvolve: an open source evolutionary coding agent for algorithmic discovery and optimization. *arXiv preprint arXiv:2510.14150*, 2025. URL https://arxiv.org/abs/2510.14150.

Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Nazanin Mohammadi Sepahvand, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Tal Arbel, Chris Pal, Gaël Varoquaux, and Pascal Vincent. Accounting for variance in machine learning benchmarks. In *Proceedings of Machine Learning and Systems*, volume 3, pp. 747–769, 2021.

Ulisses Braga-Neto. The AI scientific community: Agentic virtual lab swarms. *arXiv preprint arXiv:2603.21344*, 2026.

Shuhao Cao. Choose a transformer: Fourier or galerkin. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(70):2079–2107, 2010. URL https://www.jmlr.org/papers/v11/cawley10a.html.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei A. Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-4082.

Angelica Chen, David Dohan, and David So. EvoPrompting: Language models for code-level neural architecture search. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023. arXiv:2302.14838.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Yonatan Gideoni, Sebastian Risi, and Yarin Gal. Simple baselines are competitive with code evolution. *arXiv preprint arXiv:2602.16805*, 2026.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomašev, Dina Zverinski, Ivor Rendulic, Elahe Vedadi, Florian Hasler, Luka Rimanic, Marina Boia, Ivan Budiselic, Ben Feinstein, Mathias Bellaiche, Tom Sheffer, Jan Freyberg, Jeremy Ratcliff, Ottavia Bertolli, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R. D. Costa, José R. Penadés, Gary Peltz, Yossi Matias, James Manyika, Demis Hassabis, Yunhan Xu, Pushmeet Kohli, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Accelerating scientific discovery with Co-Scientist. *Nature*, 655(8122):487–496, 2026. doi: 10.1038/s41586-026-10644-y.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. GNOT: A general neural operator transformer for operator learning. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202, 2023.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2308.00352.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. URL https://arxiv.org/abs/2409.12186.

Qile Jiang and George Karniadakis. AgenticSciML: collaborative multi-agent systems for emergent discovery in scientific machine learning. *npj Artificial Intelligence*, 2(1):57, 2026. doi: 10.1038/s44387-026-00102-5.

Prathyusha Jwalapuram, Hehai Lin, Chuyuan Li, Fangkai Jiao, Sudong Wang, Yifei Ming, Zixuan Ke, Chengwei Qin, Giuseppe Carenini, and Shafiq Joty. The illusion of multi-agent advantage. *arXiv preprint arXiv:2606.13003*, 2026. URL https://arxiv.org/abs/2606.13003.

James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN'95)*, volume 4, pp. 1942–1948, 1995. doi: 10.1109/ICNN.1995.488968.

Shengxin Kong, Liwen Xu, and Jingwen Fu. Improving auto-design of neural PDE solvers with a domain-specific language. *arXiv preprint arXiv:2608.04384*, 2026.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. *arXiv preprint arXiv:2206.08896*, 2022. URL https://arxiv.org/abs/2206.08896.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023a. arXiv:2303.17760.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pp. 367–377. PMLR, 2019. URL https://proceedings.mlr.press/v115/li20c.html.

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. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=eG3Qy5Oux6. J2C Certification.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for PDEs on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023b. arXiv:2207.05209.

Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, 21(243):1–18, 2020. URL https://www.jmlr.org/papers/v21/20-056.html.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. arXiv:1806.09055.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. Towards end-to-end automation of AI research. *Nature*, 651(8107):914–919, 2026. doi: 10.1038/s41586-026-10265-5.

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. doi: 10.1038/s42256-021-00302-5.

Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022. doi: 10.1016/j.cma.2022.114778.

Ziming Luo, Atoosa Kasirzadeh, and Nihar B. Shah. The more you automate, the less you see: Hidden pitfalls of AI scientist systems. *arXiv preprint arXiv:2509.08713*, 2025. NeurIPS 2025 Workshop on AI for Science.

Muhammad U. Nasir, Sam Earle, Christopher W. Cleghorn, Steven James, and Julian Togelius. LLMatic: Neural architecture search via large language models and quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '24, pp. 1110–1118, New York, NY, USA, 2024. Association for Computing Machinery. doi: 10.1145/3638529.3654017.

Jingjie Ning, Shanshan Zhong, Xiaochuan Li, Ji Zeng, and Chenyan Xiong. One run is not an idea: The implementation lottery in automated research. *arXiv preprint arXiv:2607.26587*, 2026.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

Ollama. Ollama. https://ollama.com, 2026. Open-source software (MIT license), source code at https://github.com/ollama/ollama. Accessed 2026-09-23.

Tal Oved, Roi Pony, Oshri Naparstek, and Udi Barzelay. Evolution or illusion? rethinking evaluation in LLM evolutionary search. *arXiv preprint arXiv:2609.19799*, 2026.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2307.07924.

Qwen Team. Qwen3-Coder-30B-A3B-Instruct. Hugging Face model card, https://huggingface.co/Qwen/Qwen3-Coder-30B-A3B-Instruct, 2025. Accessed 2026-09-23.

Md Ashiqur Rahman, Zachary E. Ross, and Kamyar Azizzadenesheli. U-NO: U-shaped neural operators. *Transactions on Machine Learning Research (TMLR)*, 2023. arXiv:2204.11127.

Bogdan Raonic, Roberto Molinaro, Tobias Rohner, Siddhartha Mishra, and Emmanuel de Bezenac. Convolutional neural operators. In *ICLR 2023 workshop on physics for machine learning*, 2023.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4780–4789, 2019. doi: 10.1609/aaai.v33i01.33014780.

Esteban Real, Chen Liang, David R. So, and Quoc V. Le. AutoML-Zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8007–8019, 2020. arXiv:2003.03384.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi (eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, volume 9351 of *Lecture Notes in Computer Science*, pp. 234–241, Cham, 2015. Springer. doi: 10.1007/978-3-319-24574-4_28.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent Laboratory: Using LLM agents as research assistants. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 5977–6043, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.320. URL https://aclanthology.org/2025.findings-emnlp.320/.

Zeyuan Song, Xiaocong Zhen, and Zheyu Jiang. Can large language models design effective neural operators for solving partial differential equations? In *3rd AI for Math Workshop: Toward Self-Evolving Scientific Agents (ICML 2026 Workshop)*, 2026. URL https://openreview.net/forum?id=6GCIuS0oO0.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: An extensive benchmark for scientific machine learning. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 35, 2022.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, Andrei Lupu, Roberta Raileanu, Kelvin Niu, Tatiana Shavrina, Jean-Christophe Gagnon-Audet, Michael Shvartsman, Shagun Sodhani, Alexander H. Miller, Abhishek Charnalia, Derek Dunfield, Carole-Jean Wu, Pontus Stenetorp, Nicola Cancedda, Jakob Nicolaus Foerster, and Yoram Bachrach. AI research agents for machine learning: Search, exploration, and generalization in MLE-bench. *arXiv preprint arXiv:2507.02554*, 2025.

Juan Diego Toscano, Daniel T. Chen, and George Em Karniadakis. ATHENA: Agentic team for hierarchical evolutionary numerical algorithms. *arXiv preprint arXiv:2512.03476*, 2025.

Dat Tran and Douwe Kiela. Single-agent LLMs outperform multi-agent systems on multi-hop reasoning under equal thinking token budgets. *arXiv preprint arXiv:2604.02460*, 2026. URL https://arxiv.org/abs/2604.02460.

Tapas Tripura and Souvik Chakraborty. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023. doi: 10.1016/j.cma.2022.115783.

Sudhir Varma and Richard Simon. Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics*, 7:91, 2006. doi: 10.1186/1471-2105-7-91.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. Rethinking the bounds of LLM reasoning: Are multi-agent discussions the key? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6106–6131, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.331. URL https://aclanthology.org/2024.acl-long.331/.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *First Conference on Language Modeling (COLM)*, 2024. arXiv:2308.08155.

Yang Xu, Jiefu Zhang, Haixiang Sun, Zihan Zhou, Tianyu Cao, and Vaneet Aggarwal. Towards reliable LLM evaluation: Correcting the winner's curse in adaptive benchmarking. *arXiv preprint arXiv:2605.05973*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. NAS evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HygrdpVKvr.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://iclr.cc/virtual_2020/poster_H1loF2NFwr.html.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems 36 (NeurIPS), Datasets and Benchmarks Track*, 2023a. arXiv:2306.05685.

Mingkai Zheng, Xiu Su, Shan You, Fei Wang, Chen Qian, Chang Xu, and Samuel Albanie. Can GPT-4 perform neural architecture search? *arXiv preprint arXiv:2304.10970*, 2023b.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

A HARNESS AND DATA

Darcy. The domain is $[0,1]^2$ with a 32×32 grid of nodes that includes the boundary, so the spacing is $1/31$. The coefficient is $a = \exp(g)$ with

$$g(x,y) = \sum_{m,n=1}^{6} \frac{c_{mn}\sin(\pi m x)\sin(\pi n y) + d_{mn}\cos(\pi m x)\cos(\pi n y)}{m^2+n^2},$$

where c_{mn} and d_{mn} are standard normal. The equation $-\nabla\cdot(a\nabla u) = 1$ is solved on the 30×30 interior nodes with a five-point finite-volume stencil and harmonic means of a on the cell faces, with $u = 0$ on the boundary. The input is $\log a$ and the target is u, both without normalization. The 64×64 data use the same generator with spacing $1/63$.

Navier-Stokes. The vorticity ω on the periodic unit square follows $\partial_t\omega + \mathbf{u}\cdot\nabla\omega = \nu\Delta\omega + f$ with $\nu = 10^{-2}$ and $f = 0.1\,(\sin 2\pi(x+y) + \cos 2\pi(x+y))$, where the velocity comes from the stream function. It is integrated to $t = 4$ with a pseudo-spectral method, two-thirds dealiasing and an implicit-explicit step of $2\cdot 10^{-3}$. The input is $\omega(0)$ and the target $\omega(4)$.

Allen-Cahn. The field follows $\partial_t u = \varepsilon^2\Delta u + u - u^3$ with $\varepsilon = 0.1$ on the periodic unit square, integrated to $t = 4$ with explicit Euler steps of $5\cdot 10^{-3}$ and a five-point Laplacian. The input is $u(0)$ and the target $u(4)$.

Initial fields and scaling. The initial fields of Navier-Stokes and Allen-Cahn are sums of periodic sines and cosines with wave numbers up to 5 and amplitudes proportional to $(m^2+n^2)^{-1}$, each normalized to unit standard deviation. For these two families the inputs are standardized with the training mean and standard deviation and the targets are divided by the training standard deviation, which leaves the relative error unchanged.

Seeds. The seed of a run initializes the random number generators before the training and test samples are drawn, and again before the model is built, so it fixes the data, the initialization and the batch order. Training runs on the GPU without deterministic kernels; retraining the same program on the same seed reproduces its error to about three significant digits.

Harness. AdamW over all parameters, cosine annealing over the epochs without warm-up, batch size 16 with shuffling, mean squared error on the raw targets, gradient norm clipped at 1. The learning rate and weight decay of a program are clipped to $[10^{-5}, 10^{-2}]$ and $[0, 10^{-2}]$. A non-finite loss ends the run as a failure. The harness also reports the error under Gaussian input noise with standard deviation 0.1, which we do not use.

B PROMPTS

The prompts are in the released code; we summarize what each contains. The planner samples at temperature 0.7, repairs at 0.3 and the reviewer at 0.2, with a context of 12,288 tokens (16,384 for the reviewer).

Planner. A system message casts the model as a research scientist in a virtual lab whose mission is to discover new neural operators, reasoning about the solution operator of an elliptic PDE with Dirichlet boundaries. The user message states the Darcy problem, the grid, the data split and the fitness with examples; quotes the errors reported by the ablation without coding for DeepONet, the hybrid and the FNO; describes the protocol, the parameter cap and the time limit; lists the layer library (blocks arm) or the rules for writing every layer by hand (scratch arm); adds the shape guidance; and gives the lab's last three attempts, its current program, the last critique and up to two inspirations with their code and errors, with the instruction not to copy them verbatim. The requested output is a line `HYPOTHESIS:` (and, in the scratch arm, a short `MECHANISM:`) followed by one Python block.

Shape guidance. A paragraph shared by both arms reminds the model that the output must have shape $(B, 1, 32, 32)$, names two frequent failures (channel counts that do not add up after a

concatenation, and in-place operations on tensors that autograd still needs) and asks it to trace the channel count through the network.

Repair. The task and library text, the error, the last 1,800 characters of the traceback and the failing program, with the instruction to fix the bug with the smallest change and keep the idea.

Reviewer. The task text and, for every lab, its latest error, hypothesis and full code. The rubric asks for evidence, soundness for the problem, novelty with respect to the other labs and efficiency; in the scratch arm it adds novelty with respect to canonical operators, caps textbook re-implementations at 5, and asks the reviewer to check that the code does what the text claims. The output is a JSON object with a score and a critique per lab and the two cited labs.

C BASELINES

Table 4 lists every baseline run. The FNO rows follow the reference implementation of Li et al. (2021), which appends the two coordinates of each grid point to the input and pads the domain before the Fourier layers. DeepONet follows the configuration of the ablation without coding, including its 600 training epochs. The harness does not normalize inputs or outputs, while the ablation without coding normalized them for DeepONet, so the DeepONet numbers of Table 14 are not directly comparable with ours. The U-Net has three downsampling stages, and the hybrid discovered without coding is also trained widened to 1.78M and 1.97M parameters.

Table 4: All baseline runs (test relative L^2 error, three seeds).

Model	lr	Params	Darcy	Navier-Stokes	Allen-Cahn
FNO, width 32	$5{\cdot}10^{-4}$	1.19M	0.0349 ± 0.0019	0.0450 ± 0.0053	0.1616 ± 0.0156
FNO, width 32	$1{\cdot}10^{-3}$	1.19M	0.0256 ± 0.0025	0.0355 ± 0.0036	0.1534 ± 0.0175
FNO, width 32	$2{\cdot}10^{-3}$	1.19M	0.0201 ± 0.0020	0.0330 ± 0.0032	0.1462 ± 0.0199
FNO, width 64	$5{\cdot}10^{-4}$	4.74M	0.0267 ± 0.0016	0.0308 ± 0.0022	0.1490 ± 0.0135
FNO, width 64	$1{\cdot}10^{-3}$	4.74M	0.0205 ± 0.0010	0.0294 ± 0.0012	0.1439 ± 0.0125
FNO, width 64	$2{\cdot}10^{-3}$	4.74M	0.0154 ± 0.0006	0.0293 ± 0.0022	0.1388 ± 0.0109
U-Net	$5{\cdot}10^{-4}$	1.93M	0.0800 ± 0.0052	0.0426 ± 0.0010	0.0965 ± 0.0133
U-Net	$1{\cdot}10^{-3}$	1.93M	0.0741 ± 0.0039	0.0402 ± 0.0005	0.0926 ± 0.0084
U-Net	$2{\cdot}10^{-3}$	1.93M	0.0774 ± 0.0082	0.0496 ± 0.0065	0.0959 ± 0.0057
DeepONet (600 epochs)	$5 \cdot 10^{-4}$	0.60M	0.0612 ± 0.0044	0.1668 ± 0.0190	0.2924 ± 0.0056
Hybrid, width 76	prog.	1.78M	0.0689 ± 0.0042	0.0299 ± 0.0019	0.0643 ± 0.0110
Hybrid, width 80	prog.	1.97M	0.0630 ± 0.0055	0.0294 ± 0.0018	0.0629 ± 0.0125

D TRANSFER BETWEEN PROBLEMS

Table 5 lists every fixed program that was retrained unchanged on the three problems. Retrained on Navier-Stokes, the program that won on Darcy with the accuracy-first reviewer has the lowest error of Table 1 (0.0253), below the FNO (0.0330) and an FNO with four times as many parameters (0.0293), but it beats the hybrid discovered without coding on only two of three seeds. The table also includes the seed programs of both arms and a scratch program rescaled to the parameter count of the hybrid.

Table 5: Fixed programs retrained unchanged on each family (three seeds, test relative L^2 error). Darcy values of selected programs include their selection seed and are therefore optimistic.

Program	Params	Darcy	Navier-Stokes	Allen-Cahn
Selected, blocks, 16×16 (seed 42)	1.68M	0.0235 ± 0.0012	0.0326 ± 0.0012	0.0731 ± 0.0112
Selected, blocks, acc.-first (seed 2024)	1.91M	0.0234 ± 0.0059	0.0253 ± 0.0044	0.1183 ± 0.0220
Selected, scratch, 16×16 (seed 137)	1.66M	0.0240 ± 0.0015	0.0508 ± 0.0047	0.1533 ± 0.0089
Scratch program, width matched	1.25M	0.0592 ± 0.0023	0.1863 ± 0.0077	0.2641 ± 0.0103
Hybrid (fixed vocabulary)	1.26M	0.0719 ± 0.0061	0.0287 ± 0.0016	0.0661 ± 0.0132
Seed program, blocks	0.89M	0.3133 ± 0.0279	0.6557 ± 0.0210	0.0999 ± 0.0104
Seed program, scratch	0.90M	0.2510 ± 0.0186	0.4493 ± 0.0284	0.2433 ± 0.0157

E COMMUNITY WIDTH

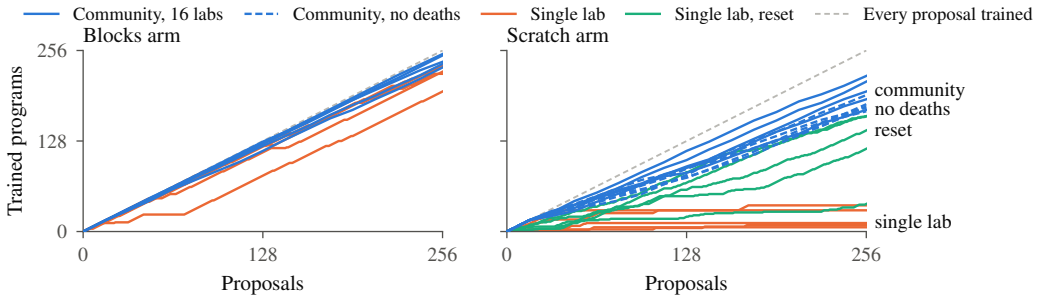


Figure 2: Cumulative number of trained programs against proposals on the Darcy problem, for five simulations of a community of 16 labs (16 iterations) and five of a single lab (256 iterations) in each arm, and, in the scratch arm, five of a single lab that returns to its last working program after each failure (reset) and five of a community whose labs never die (dashed). A proposal is trained only if it passes the check before training, possibly after repairs.

Figure 2 shows how many proposals each configuration of Table 2 trains. Table 6 varies the number of labs at a fixed budget of 96 proposals on Darcy. In the scratch arm, one and two labs train 8 and 27 programs per simulation and four and eight labs 59 and 55, with about half the error (0.0748 and 0.0709 against 0.1377 and 0.1480). In the blocks arm every width trains almost every proposal, and the error decreases slowly with the number of labs, from 0.1154 to 0.0839, with a large spread across seeds.

Table 6: Community width at a fixed budget of 96 proposals on Darcy: N labs for T iterations, three search seeds. Error of the selected program retrained on held-out seeds (test relative L^2 error) and mean number of proposals that passed the pre-training check and were trained.

	Blocks		Scratch	
$N \times T$	Error	Trained	Error	Trained
1×96	0.1154 ± 0.0753	88	0.1377 ± 0.1042	8
2×48	0.0917 ± 0.0701	86	0.1480 ± 0.0975	27
4×24	0.0902 ± 0.0204	90	0.0748 ± 0.0316	59
8×12	0.0839 ± 0.0565	91	0.0709 ± 0.0473	55

F CONTROLS

Table 7 repeats the Darcy comparison with longer training and at a higher resolution. Retrained for 300 epochs instead of 80, every model improves, the discovered programs stay 2.0 times more accurate than the hybrid discovered without coding (0.0187 against 0.0372), and the FNO stays the most accurate model (0.0118). At 64×64, the FNO with a tuned learning rate reaches 0.0198, while the best of the 11 discovered programs that could be trained at this resolution reaches 0.0225 and the median 0.0464; two more exceed the 32 GB of GPU memory or diverge. At its default learning rate the FNO reaches 0.0304, and three discovered programs beat it; with the learning rate tuned as at 32×32, none does. The 30B planner does not lower the error (Appendix I), but its winning programs have about half the parameters; the smallest, in Figure 3, has 6 times fewer parameters than the hybrid and a lower error (0.0307).

Table 8 lists the discovered programs used in these controls. Each won a simulation at 32×32 and 80 epochs and is scored only on the seeds not used to select it. A program that fails at 64×64 either exceeds the 32 GB of GPU memory with a batch of 16 or diverges in the first epoch.

Table 7: Controls on Darcy. Top: training for 300 instead of 80 epochs. Bottom: training and testing at 64×64. Discovered programs were selected at 32×32 and 80 epochs and are scored only on seeds not used to select them; "median" and "best" are over the programs listed in Appendix F. Test relative L^2 error.

Model	80 epochs	300 epochs
Hybrid (fixed vocabulary)	0.0719 ± 0.0061	0.0372 ± 0.0008
FNO, width 32, lr 10^{-3}	0.0256 ± 0.0025	0.0118 ± 0.0010
Discovered (6 programs)	0.0261 ± 0.0035	0.0187 ± 0.0050
	32×32	64×64
FNO, width 32	0.0201 ± 0.0020	0.0198 ± 0.0006
FNO, width 64†	0.0154 ± 0.0006	0.0146 ± 0.0005
U-Net	0.0741 ± 0.0039	0.0662 ± 0.0049
Discovered, best of 11	0.0218	0.0225
Discovered, median of 11	0.0299	0.0464

Table 8: Every discovered program in the Darcy controls, scored on the seeds not used to select it (test relative L^2 error). "Search" is the configuration (N labs $\times$ T iterations) and the search seed of the run that selected the program.

Search	Seed	80 ep., 32^2	300 ep., 32^2	80 ep., 64^2
blocks, 16×64, acc.-first	137	0.0304	0.0274 ± 0.0095	fails
blocks, 16×64, acc.-first	2024	0.0299	0.0165 ± 0.0017	0.0464 ± 0.0180
blocks, 16×64, acc.-first	42	0.0222	0.0149 ± 0.0007	0.0259 ± 0.0019
blocks, 16×16, acc.-first	137	0.0396	n/a	0.0430 ± 0.0126
blocks, 16×16, acc.-first	42	0.0455	n/a	0.0505 ± 0.0026
blocks, 16×16	42	0.0229	0.0164 ± 0.0014	n/a
blocks, 16×16, acc.-first	2024	0.0259	0.0221 ± 0.0038	0.0284 ± 0.0013
blocks, 16×64	137	0.0428	n/a	0.0588 ± 0.0138
blocks, 16×64	2024	0.0218	n/a	0.0225 ± 0.0043
blocks, 16×64	42	0.0404	n/a	0.0411 ± 0.0083
blocks, 16×16	137	0.0724	n/a	0.0758 ± 0.0139
blocks, 16×16	2024	0.0414	n/a	0.0481 ± 0.0065
scratch, 16×16	2024	0.0449	n/a	0.1017 ± 0.0049
scratch, 16×16	42	0.1264	n/a	fails
scratch, 16×16	137	0.0252	0.0149 ± 0.0014	n/a

G EXAMPLES OF WINNING PROGRAMS

The best winning program of the Darcy simulations that start from the seed program (`sw_gpu_blocks_L16_I64_s2024`, held-out error 0.0218, 1,064,482 parameters) is a genuine architectural composition rather than a tuned version of the seed. After a pointwise lift and a 5×5 convolution, it applies three stacked units that add the outputs of a Fourier layer and a wavelet layer, gate the sum and project it, with a residual connection around the stack. Global self-attention, a gate and a second residual follow. The model then multiplies the outputs of two convolutional paths, adds a residual path and projects to the output. Two details are typical of the programs we read. The same convolutional module is applied twice with shared weights, and two modules are defined but never called, so part of the parameter count is unused. Names are also unreliable: a module called `GlobalLowPassFilter` is an unconstrained 5×5 convolution, and one called `spectral_filter` is a 1×1 convolution. We therefore describe programs by reading their code, never by their names or by the hypothesis the planner states.

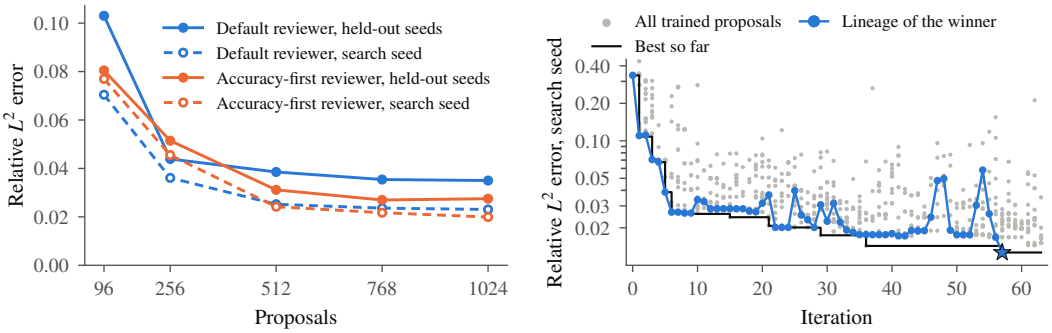
On Allen-Cahn, the smallest winning program (207,585 parameters, held-out error 0.0510) is a compact hybrid at width 32: a Fourier layer with six modes, self-attention over the grid, a spectral convolution and a wavelet layer applied in parallel and summed, a residual convolution and a gate. Like the hybrids found without coding, it mixes four families of layers, with 6 times fewer parameters than the hybrid of Table 1.

```
HYPOTHESIS: I will replace the fixed Green's function kernel with a learned integral
    ↪ operator that uses coordinate-based attention to model long-range interactions ...

def forward(self, x, coeff):
    B, _, H, W = x.shape
    y_coords = torch.linspace(-1, 1, H, device=x.device)
    x_coords = torch.linspace(-1, 1, W, device=x.device)
    grid_y, grid_x = torch.meshgrid(y_coords, x_coords, indexing='ij')
    coord_grid = torch.stack([grid_x, grid_y], dim=0).unsqueeze(0).expand(B, -1, -1, -1)
    attn_weights = self.coord_attn(coord_grid)
    outputs = []
    for i, kernel in enumerate(self.kernels):
        out = kernel(x)
        outputs.append(out * attn_weights[:, i:i+1])
    weighted_output = sum(outputs)
    ...
```

Figure 3: The hypothesis and part of the forward pass of the smallest winning program of the 30B planner, written from scratch for the Darcy problem (207,757 parameters, held-out error 0.0307; verbatim, comments removed). Each block weights 3×3, 5×5 and 7×7 depthwise convolutions by a softmax of the grid coordinates. In its full text, the hypothesis also says that the weights depend on the coefficient field, which the code never uses.

H SELECTION BIAS AND BUDGET



(a) Search seed against held-out seeds.

(b) Lineage of the winner of the best simulation.

Figure 4: Simulations of 1024 proposals on the Darcy problem (16 labs for 64 iterations). (a) Best program so far, on the search seed (dashed) and retrained on the two held-out seeds (solid), mean over three simulations (numbers in Table 9). (b) Search-seed errors in the best simulation: every trained proposal (grey), the best so far (black) and, at each iteration, the ancestor of the winning lab through the lifecycle (blue). The winner (star) is proposed at iteration 57.

Figure 4a follows the best program found so far inside the three simulations of 1024 proposals. Measured on the search seed, going from 256 to 1024 proposals lowers the error by a factor of 1.57. Measured on held-out seeds, the same step lowers it by a factor of only 1.25: the error falls from 0.0439 to 0.0385 at 512 proposals and then barely moves (0.0350 at 1024). In one of the six simulations, the program selected at 1024 proposals is worse on held-out seeds than the one selected at 256. The gap between the two measurements is the selection bias described by Cawley & Talbot (2010): the winner of many candidates compared on the same test set partly owes its score to luck on that set. The held-out error is 1.22 times the search-seed error after 256 proposals and 1.52 times after 1024 (Table 10), so search-seed errors, the ones a search log records, would have made a larger budget look far more useful than it is. With the accuracy-first reviewer, the held-out error at 1024 proposals is 0.0275, a factor of 1.27 below the default reviewer and lower on two of three seeds, and all three simulations end below the hybrid discovered without coding, by a factor of 2.6 on average (Table 9).

How the community reaches its winner. Figure 4b follows the best of these simulations. The winning lab descends through 39 labs, each founded from the most cited program of its round, and

Table 9: Best program found so far inside the three 16-lab runs of 1024 proposals, measured on the search seed ("search") and retrained on the two held-out seeds ("held-out"). Test relative L^2 error, mean over the three runs.

	Default objective		Accuracy-first	
Proposals	Search	Held-out	Search	Held-out
96	0.0704	0.1030	0.0770	0.0805
256	0.0361	0.0439	0.0455	0.0515
512	0.0252	0.0385	0.0242	0.0312
768	0.0236	0.0354	0.0217	0.0270
1024	0.0231	0.0350	0.0199	0.0275

its ancestor was cited in 78% of the reviews. The lineage is not monotone, since labs keep their latest program even when it is worse: the citations select a direction of research rather than the best program at each moment, as the citation economy was designed to do.

Table 10: Selection bias. For each searched configuration: number of search seeds, error of the selected program on its own search seed, the same program retrained on held-out seeds, and their ratio. Test relative L^2 error.

Configuration	Seeds	Search	Held-out	Ratio
blocks, 16×16	5	0.0425	0.0525	1.24
scratch, 16×16	5	0.0416	0.0554	1.33
blocks, 1×256	5	0.0581	0.0631	1.09
scratch, 1×256	5	0.1619	0.1604	0.99
scratch, 1×256, reset	5	0.0626	0.0685	1.10
scratch, 16×16, no deaths	5	0.0619	0.0700	1.13
blocks, 16×16, acc.-first	3	0.0323	0.0370	1.14
blocks, 16×64	3	0.0231	0.0350	1.52
blocks, 16×64, acc.-first	3	0.0199	0.0275	1.38
blocks, 16×16, 30B	3	0.0412	0.0498	1.21
scratch, 16×16, 30B	3	0.0501	0.0602	1.20
random search, 256	5	0.0605	0.0640	1.06

I PLANNER MODEL AND PRE-TRAINING CHECKS

Table 11 compares the two planner models and the effect of the checks that run before training (Section 4). The winning programs of the 30B planner have about half the parameters of those of the 7B planner (0.87M against 1.60M in the blocks arm, 0.82M against 1.31M in the scratch arm, means over three seeds). The checks help the blocks arm at both budgets. In the scratch arm they help slightly at 24 proposals and hurt at 96.

Table 11: Planner model and scaffold. Held-out test relative L^2 error of the selected programs (three search seeds, 42, 137 and 2024) and mean parameter count of the selected programs.

Configuration	Model	Held-out	Params
blocks, 16×16	7B	0.0456 ± 0.0250	1.60M
blocks, 16×16	30B	0.0498 ± 0.0077	0.87M
scratch, 16×16	7B	0.0655 ± 0.0536	1.31M
scratch, 16×16	30B	0.0602 ± 0.0260	0.82M
Configuration	**Scaffold**	**Held-out**	
blocks, 4×6	without checks	0.2226 ± 0.0783	
blocks, 4×6	with checks	0.0824 ± 0.0412	
blocks, 4×24	without checks	0.1053 ± 0.0123	
blocks, 4×24	with checks	0.0482 ± 0.0146	
scratch, 4×6	without checks	0.1851 ± 0.0488	
scratch, 4×6	with checks	0.1623 ± 0.0657	
scratch, 4×24	without checks	0.0685 ± 0.0341	
scratch, 4×24	with checks	0.0982 ± 0.0190	

J SIMULATIONS EXCLUDED BECAUSE OF A DATA-ROUTING ERROR

We ran nine further simulations of 16 labs for 16 iterations that were meant to search on Navier-Stokes (six) and Allen-Cahn (three). A variable-scoping error in the search driver discarded the data directory, so all nine trained and verified their programs on Darcy, with prompts that describe Darcy. We found the error by comparing the error of the seed program at iteration 0, which is identical to its Darcy value, with its error on the intended data. We exclude these runs from all tables. As additional Darcy replicates they are consistent with the main configurations: the six blocks runs reach 0.0341 and the three scratch runs 0.0460, against 0.0525 and 0.0554 for the corresponding configurations in Table 2. We then fixed the error and repeated these simulations on those problems, with prompts that describe each PDE; Table 1 reports the repeated simulations.

K COMPUTE

All simulations with coding, their retrainings and the baselines ran on one workstation with an NVIDIA RTX 5090 (32 GB) and the planner models served locally. The 117 simulations reported in this paper trained 21,332 programs and took 310 hours of wall-clock time in total, often three at a time on the same GPU. A simulation of 256 proposals took 2.6 hours in the median, and a simulation of 1024 proposals about 16 hours. We neither report nor count six pilot simulations of 4 labs for 6 iterations, which were run while the pipeline was being set up and repeat a configuration of Table 11.

L ABLATION WITHOUT CODING: EXPERIMENTAL DETAILS

Algorithm 1 summarizes an iteration of the community without coding. In each iteration, every active lab proposes its next genome; compiles and trains it at screening fidelity, returning the measured relative L_2 error on clean and noisy data; and reviews a sample of peers, casting soft votes that accumulate as citation-like influence and update trust scores. The composite fitness and the global best are then updated, influential labs persist and breed, and uncited labs are replaced.

Table 12 lists the screening-phase configuration per problem (from the file `config.json` of each run in the repository). Both LLM and rule-based agents share these budgets exactly; only the planner/reviewer policy differs. Every candidate is trained from scratch for the stated epochs each iteration; robustness to noise is measured by adding Gaussian noise $\varepsilon \sim \mathcal{N}(0,\, 0.1^2)$ to the data. The LLM backend is `gemma3:12b` served by Ollama with an 8192-token context window; all timings are on a single RTX 4080. The separately trained baselines and winning architectures run for 50 epochs for FNO, 80 epochs for the winning hybrids, POD-DeepONet and transformer, and 600 epochs for the DeepONet with input/output normalization, as encoded in the file `validate_baselines_v3.py` in the repository. At first, we adopted `gemma4:e4b`, a smaller efficiency-tier LLM, which ran into

the silent non-LLM fallback issue, even with prompt hardening and an enlarged context window. This led us to switch to `gemma3:12b` and audit the LLM calls.

Why a swarm of hundreds of trainings is feasible. The swarm screens architectures at reduced fidelity, so a single operator trains in seconds, not hours. Table 17 in Appendix P lists single-operator training times to convergence at screening fidelity in the Navier-Stokes problem; cost is dominated by epochs and block type, not parameter count (the 105K-parameter transformer is slower to train than the 1.2M-parameter FNO because attention is quadratic in sequence length). A full simulation run evaluates $16 \times 20 = 320$ architectures within the budgets above, reserving expensive full-fidelity training for the final discovered architecture and the baseline comparison. This follows the standard multi-fidelity procedure to screen cheaply and commit expensively.

Algorithm 1 Neural operator AI scientific community without coding.

initialize N labs across paradigms (FNO, DeepONet, attention, wavelet, convolutional, hybrid).
for iteration $t = 1, \ldots, T$ **do**
  for each active lab i **do**
    $g_i \leftarrow \textsc{Plan}(g_i,\ g^{\text{best}},\ \{g_j\}_{j \neq i},\ \text{history}_i)$
  end for
  for each active lab i **do**
    $\theta_i \leftarrow \textsc{Train}(g_i)$; measure relative L_2
  end for
  $\{v_i\} \leftarrow \textsc{PeerReview}(\{(g_i, \text{fitness}_i)\})$
  update composite fitness, trust, global best; deactivate uncited labs, breed influential labs
end for

Table 12: Screening-phase configuration per problem in the ablation without coding (identical across conditions and simulations).

Problem	Dim	Resolution	Train / test	Epochs per candidate
Piecewise Reg.	1D	128	256 / 64	40
Advection	1D	128	256 / 64	40
Burgers	1D	128	256 / 64	40
Navier-Stokes	2D	32×32	512 / 128	50
Darcy	2D	32×32	512 / 128	50

M Ablation without coding: winning architectures

Table 13 lists the details for the winning architectures in each simulation (5 problems $\times$ 2 conditions $\times$ 3 random seeds), including the rule-based agents in the ablation study, and Figure 5 draws the LLM winners of one simulation. We observe that the LLM agents return diverse architectures, but are problem-specific; e.g., they almost always lead the stacks with Fourier blocks in the spectral-friendly advection, Burgers, and Navier-Stokes problems. The rule-based agents tend to be less diverse and prefer Fourier blocks in the spectral-friendly problems and wavelet blocks in the piecewise regression and Darcy problems. A machine-readable version of these results is in the file `results/aggregate_v2.json` in the repository.

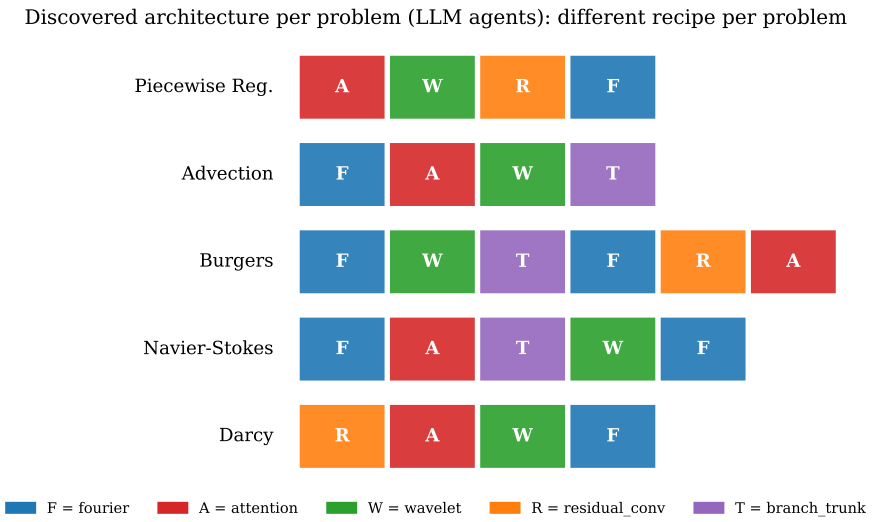


Figure 5: Winning architectures without coding in one of the simulations (random seed 42). Every winner is a multi-family hybrid. As expected, Fourier blocks lead the stacks in the spectral-friendly advection, Burgers and Navier-Stokes problems.

Table 13: Winning architectures without coding. Blocks abbreviated as T=DeepONet, F=Fourier, A=Attention, W=Wavelet, R=Residual Convolution.

Problem	Cond.	Seed	Discovered blocks	rel. L_2	Params
Piecewise Reg. (1D)	LLM	42	A–W–R–F	0.0429	692K
Piecewise Reg. (1D)	LLM	137	R–R–W–T–A	0.0419	122K
Piecewise Reg. (1D)	LLM	2024	A–R–W–T	0.0404	647K
Piecewise Reg. (1D)	PSO	42	W–A–W–W	0.0410	333K
Piecewise Reg. (1D)	PSO	137	W–R–W–W–W	0.0396	128K
Piecewise Reg. (1D)	PSO	2024	R–W–W–R–W	0.0410	134K
Advection (1D)	LLM	42	F–A–W–T	0.0191	143K
Advection (1D)	LLM	137	A–R–F–F–T	0.0105	494K
Advection (1D)	LLM	2024	F–A–W–T	0.0180	182K
Advection (1D)	PSO	42	F–A–T–T–T–T	0.0186	559K
Advection (1D)	PSO	137	A–F–W–F	0.0173	576K
Advection (1D)	PSO	2024	R–F–F	0.0189	231K
Burgers (1D)	LLM	42	F–W–T–F–R–A	0.0759	1.1M
Burgers (1D)	LLM	137	F–W–A–R	0.0745	142K
Burgers (1D)	LLM	2024	F–A–T–W	0.0894	217K
Burgers (1D)	PSO	42	F–R–F–T–A–F	0.0767	335K
Burgers (1D)	PSO	137	F–F–F–T–A	0.0707	87K
Burgers (1D)	PSO	2024	F–R–F	0.0653	169K
Navier-Stokes (2D)	LLM	42	F–A–T–W–F	0.0011	1.4M
Navier-Stokes (2D)	LLM	137	F–A–W–T–R–F	0.0003	2.6M
Navier-Stokes (2D)	LLM	2024	A–W–T–A–F	0.0015	2.8M
Navier-Stokes (2D)	PSO	42	F–F–F–F	0.0007	10.9M
Navier-Stokes (2D)	PSO	137	F–W–T–F–F	0.0006	921K
Navier-Stokes (2D)	PSO	2024	F–F–F–F	0.0007	5.1M
Darcy (2D)	LLM	42	R–A–W–F	0.0099	1.9M
Darcy (2D)	LLM	137	A–W–R–F–W	0.0091	1.3M
Darcy (2D)	LLM	2024	A–W–R–T–F	0.0095	930K
Darcy (2D)	PSO	42	W–W–R–R–F	0.0089	1.7M
Darcy (2D)	PSO	137	W–W–F–W	0.0082	933K
Darcy (2D)	PSO	2024	W–W–W–W	0.0116	257K

N ABLATION WITHOUT CODING: ACCURACY-PARAMETER TRADE-OFF

Figure 6 displays the trade-off between accuracy and parameter count obtained in the simulations for the Navier-Stokes and Darcy 2D problems. All 48 labs (16 labs $\times$ 3 simulations) produced multi-family hybrids, consistent with the planner hybridizing in 99.8% of its decisions. We observe that the discovered hybrids populate a competitive region of the accuracy-parameter plane.

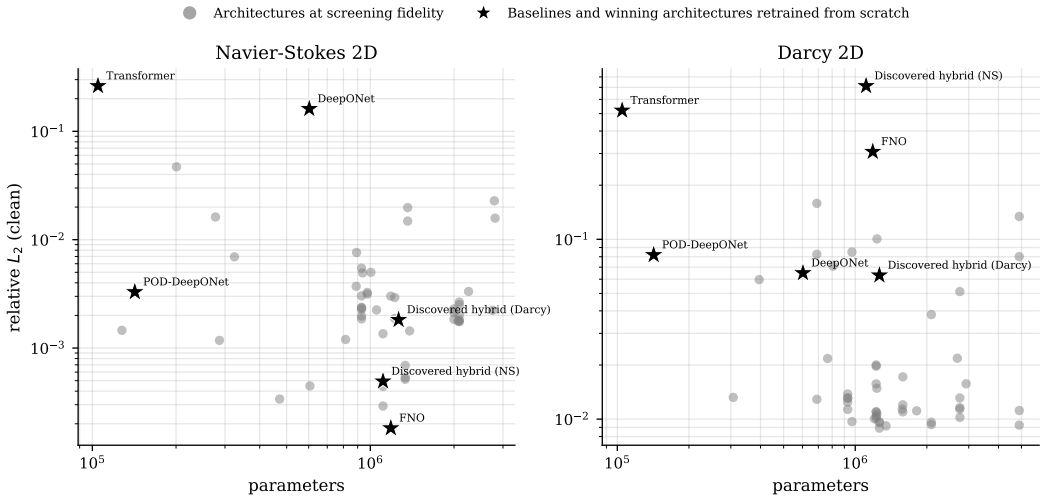


Figure 6: Accuracy-parameter trade-off without coding in the Navier-Stokes and Darcy problems. The gray dots represent the individual labs, while stars depict the baselines and the winning discovered architectures retrained from scratch to convergence. Errors are therefore not directly comparable across the two marker types.

O ABLATION WITHOUT CODING: AGENT PROMPTS

The planner and reviewer prompts are reproduced verbatim below (the specific problem name is instantiated per run). Note that the planner prompt specifies the genome record to be used by each virtual lab. The outputs are logged in the file `llm_transcript.jsonl` of each run.

Planner prompt.

```
You are the planning agent of a virtual research lab in a
swarm of labs collectively discovering neural operator architectures for 2D
Navier-Stokes. Each lab is one PSO particle; your job is to propose the next
architecture (genome) for your lab to train and evaluate next iteration.

You must balance EXPLORATION (novel hybrids, unexplored block combinations)
with EXPLOITATION (move toward the global best architecture observed in the
swarm). Use the lab's current exploration_rate as a guide: high -> favor
exploration, low -> favor exploitation. Consider what other labs are doing -
if the swarm is converging, propose something different to maintain diversity
and avoid groupthink.

Output STRICT JSON ONLY (no prose, no fences) matching this schema:
{
  "action": "explore" | "exploit" | "hybridize",
  "rationale": "<1-2 sentences explaining your choice>",
  "new_genome": { "block_sequence": [...], "hidden_channels": <int>,
    "fourier_modes": <int>, "activation": "<str>", "use_gating": <bool>,
    "use_skip_connections": <bool>, "dropout_rate": <float>,
    "learning_rate": <float>, "weight_decay": <float> }
}
```

```
- ALLOWED_BLOCKS = ["fourier","attention","branch_trunk","wavelet",
                    "residual_conv"]
```

Reviewer prompt.

```
You are an anonymous peer reviewer evaluating neural
operator architectures proposed by other virtual labs in a research swarm.

You must rank the candidates on overall scientific merit for solving 2D
Navier-Stokes operator learning. Reward: high accuracy, good generalization
under noise, parameter efficiency, architectural novelty (unusual block
combinations), and theoretical soundness (e.g. fourier blocks for periodic
domains, attention for nonlocal coupling). Penalize: groupthink (identical to
global best), pathological setups (too few or too many blocks, mismatched
modes), or poor measured fitness.

Output STRICT JSON ONLY:
{
  "scores":   { "<lab_id>": <float in [0,1]>, ... },
  "rationale": "<1-2 sentences total, not per lab>"
}
Scores must sum to roughly 1.0 across the candidates (soft-vote distribution).
```

P ABLATION WITHOUT CODING: ADDITIONAL RESULTS

Retrained winners. Table 14 reports the winning architectures of the two most challenging problems, the Navier-Stokes and Darcy 2D PDEs, each retrained from scratch in the setting of this ablation and evaluated on both problems, together with baselines trained in the same setting. The Darcy hybrid attains 0.0631 ± 0.0083 relative L_2 error, statistically tied with DeepONet (0.0649 ± 0.0037), and the Navier-Stokes hybrid reaches 0.0005. Neither hybrid carries over to the other problem: the Navier-Stokes hybrid does badly on Darcy, while the Darcy hybrid is about $3.7\times$ worse than the Navier-Stokes hybrid on Navier-Stokes.

Table 14: Relative L_2 error for the winning architectures without coding, each retrained from scratch in the setting of the ablation (Table 12), and baselines trained in the same setting, for the Navier-Stokes and Darcy 2D PDEs. Mean and standard deviations over the three simulations are reported.

Model	Params	NS-2D rel. L_2	Darcy-2D rel. L_2
DeepONet	603K	0.1613±0.0122	0.0649±0.0037
Discovered hybrid (Darcy)	1.3M	0.0018±0.0002	0.0631±0.0083
Discovered hybrid (NS)	1.1M	0.0005±0.0000	0.7127±0.0736
FNO	1.2M	0.0002±0.0000	0.3070±0.0404
POD-DeepONet	142K	0.0033±0.0001	0.0819±0.0028
Pure Transformer	105K	0.2624±0.0038	0.5207±0.0102

Computational cost. Table 16 and Figure 9 report wall-clock times for a full simulation run and the time per lab evaluation, for LLM and rule-based labs. Because the LLM and rule-based AI scientific communities train identical operators under an identical budget, the difference in per-lab time is precisely the overhead of LLM-driven planning and review (two language-model calls per lab per iteration): roughly 14 sec per lab evaluation in the 1D problems and 25-33s in the 2D problems, i.e. a 2-4$\times$ increase in total wall-clock time over rule-based coordination. The LLM labs also carry higher run-to-run variance in the 2D problems (e.g. 271 ± 104 min on Navier-Stokes): a consequence of letting the planner commit to parameter-heavy hybrids whose training dominates the budget.

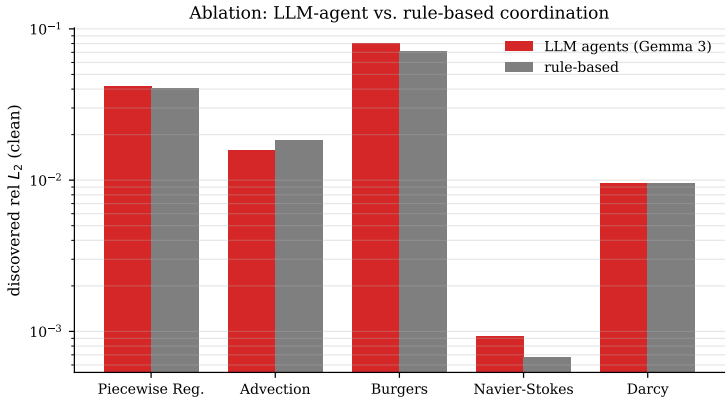


Figure 7: Ablation of LLM agency without coding. Discovered relative L_2 error per problem: LLM vs. rule-based, under an identical training budget.

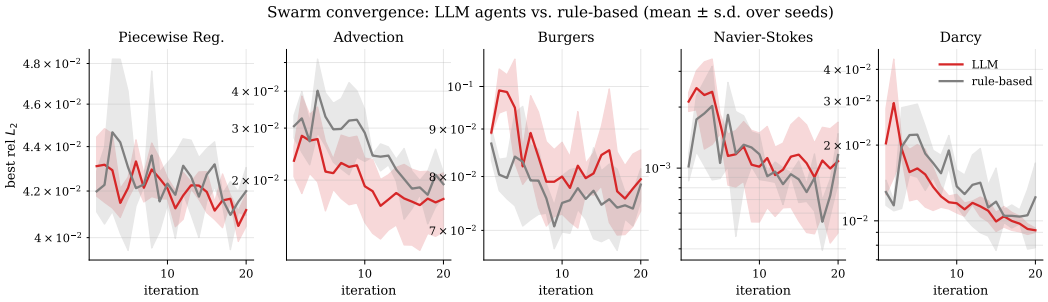


Figure 8: Ablation of LLM agency without coding. Best relative L_2 error per iteration, LLM vs. rule-based, mean $\pm$ s.d. over simulations.

Table 15: Agent audit. Planner/reviewer LLM call counts per run, parse-success, and fallbacks. Zero fallbacks certifies LLM-driven planning and review. Counts above the nominal $16 \times 20 = 320$ reflect re-planned iterations after mid-run interruptions (the runs are checkpointed and resume-safe); every call, including repeats, is in the transcripts.

LLM run	planner	reviewer	parse-ok	fallbacks
`advec seed137`	320	320	640/640	0
`advec seed2024`	320	320	640/640	0
`advec seed42`	320	320	640/640	0
`burgers seed137`	320	320	640/640	0
`burgers seed2024`	320	320	640/640	0
`burgers seed42`	320	320	640/640	0
`darcy seed137`	335	326	661/661	0
`darcy seed2024`	320	320	640/640	0
`darcy seed42`	320	320	640/640	0
`ns seed137`	320	320	640/640	0
`ns seed2024`	320	320	640/640	0
`ns seed42`	320	320	640/640	0
`pwreg seed137`	320	320	640/640	0
`pwreg seed2024`	322	320	642/642	0
`pwreg seed42`	320	320	640/640	0
total	9623 LLM calls			**0**

Table 16: Computational cost without coding on a single RTX 4080: wall-clock simulation and per lab evaluation time, LLM vs. rule-based, mean±std over simulations. The LLM-rule-based gap reveals the cost of using LLMs.

Problem	Total run (min)		Per lab-eval (s)	
	LLM	PSO	LLM	PSO
Piecewise Reg. (1D)	89.7±18.3	26.0±11.7	19.4±1.8	4.9±2.4
Advection (1D)	96.7±10.6	29.2±12.2	18.2±2.2	5.5±2.4
Burgers (1D)	108.3±2.3	41.1±0.9	19.8±1.8	7.0±0.5
Navier-Stokes (2D)	270.9±104.3	103.1±6.7	51.3±27.1	18.7±3.3
Darcy (2D)	173.6±71.9	99.9±7.9	42.1±15.9	18.4±2.6

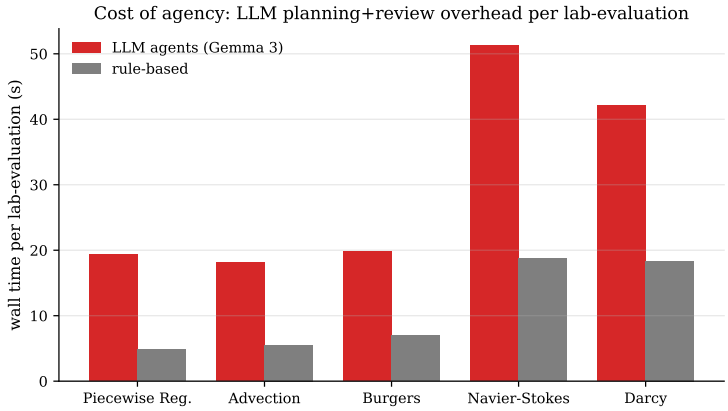


Figure 9: Per lab-evaluation wall time without coding, LLM vs. rule-based. The gap is the cost of LLM planning and peer review at identical training budget.

Table 17: Single-operator training time at screening fidelity in the Navier-Stokes problem.

Operator (NS-2D, screening fidelity)	Train time (s)
POD-DeepONet	4.1±0.0
FNO	9.7±0.4
Discovered hybrid (NS)	20.3±0.5
Discovered hybrid (Darcy)	22.5±0.2
Pure Transformer	37.4±0.1
DeepONet	48.4±0.7