GAV: A SCIENTIFIC EXPERIMENTATION ENVIRONMENT FOR PDE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reliable feedback is essential for coding agents to develop programs for partial differential equations (PDEs). Yet evaluating these programs on new PDE problems often requires expensive reference computations, while fixed datasets restrict the range of experiments. We introduce Generator–Agent–Verifier (GAV), a scientific experimentation environment that makes new PDE problems verifiable by construction. The Generator learns a distribution of continuous reference solutions using data from numerical simulations or experimental measurements to represent a target physical regime. Using the method of manufactured solutions, it works backward from the generated reference solutions to construct matching PDE problems without a separate numerical solve. The Agent designs experiments, investigates failures, and iteratively improves its program. The Verifier evaluates candidate programs against the reference solutions and provides numerical feedback on accuracy and computational cost. This feedback can also guide the generation of new problems that probe the current program's weaknesses. We provide a theoretical analysis of evaluation reliability and guided development. Experiments on Navier–Stokes and Vlasov–Poisson–Fokker–Planck tasks across different data sources demonstrate the effectiveness of GAV. In particular, on the numerical Navier–Stokes task, GAV reduces the mean test error of three existing PDE agents by more than an order of magnitude relative to their original setups.

1 INTRODUCTION

Large language model agents increasingly contribute to scientific research by writing, executing, and revising programs. Recent progress spans mathematical discovery and algorithm design (Romera-Paredes et al., 2024; Novikov et al., 2025), as well as machine-learning experimentation and scientific software development (Yamada et al., 2025; Aygün et al., 2026). Yet reliably completing multi-step scientific tasks remains challenging (Bragg et al., 2026). An experimentation environment is essential to developing reliable scientific programs. By testing candidate programs, an agent can identify failures, compare alternatives, and assess whether a revision improves accuracy or efficiency (Aygün et al., 2026; Li et al., 2025). Repeated cycles of proposing, evaluating, and revising programs also offer a way to turn additional test-time computation into better programs (Novikov et al., 2025; Ye et al., 2026). Realizing this potential requires a scientific experimentation environment in which agents can design and run informative experiments and obtain reliable feedback to guide their next steps.

We focus on developing programs for partial differential equation (PDE) tasks, including numerical solution and reduced-order modeling of specified macroscopic quantities. In these tasks, an agent not only needs to produce a program that executes successfully, but also to ensure that the program produces accurate results. Reliable numerical feedback is therefore needed to measure progress (Dong et al., 2026). For large-scale computational problems arising from real-world applications, an effective experimentation environment for PDE agents must address the following needs:

- **Asymmetric computational cost:** Reference-based evaluation compares a candidate program's output with a reference solution, often computed by a high-resolution numerical solver (Li et al., 2021). Computing these reference solutions can be prohibitively expensive when an agent evaluates many new problems. The environment must therefore provide reliable feedback without expensive reference computations for every new problem.

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.

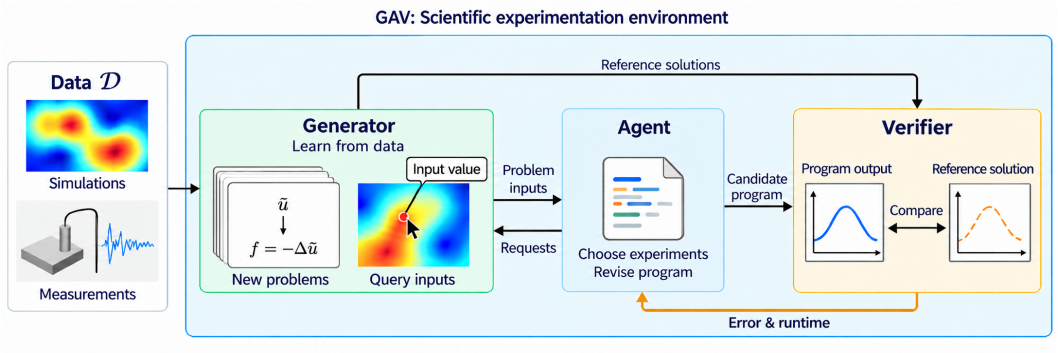

Figure 1: **GAV: a scientific experimentation environment for PDE agents.** The Generator constructs PDE problems and reference solutions from scientific data. The Agent requests problems and input information, tests programs, and uses the Verifier's feedback on error and runtime to guide further experiments and revisions. The diagram illustrates a numerical solution task.

- **Data-driven physical regimes:** The effectiveness of a numerical method or a reduced-order model depends on its operating regime. Different regimes may favor different methods. In real-world applications, the precise physical regime is often unknown a priori, and can be characterized as the distribution of plausible solutions. Therefore, the environment must provide a way to learn and represent the physical regime from simulation or measurement data, and use this knowledge to guide the agent's experimentation and program development.
- **Interactive feedback:** In experimentation environments, agents can iteratively propose experiments, receive feedback, and refine their programs. This interactive loop allows agents to explore the problem space, identify failure modes, and improve their methods. A fixed dataset alone is insufficient; the environment must provide new problems and reliable numerical feedback as experiments require.

Recent work has developed PDE agents that generate and refine solvers using numerical feedback (Li et al., 2025; Fazliani & Udell, 2025; Dong et al., 2026). However, their experimental environments do not yet meet all three needs above. We introduce the Generator–Agent–Verifier (GAV) environment (Figure 1) to address this gap. The Generator learns a distribution of reference solutions from simulation or measurement data, then constructs matching PDE problems using the method of manufactured solutions (Salari & Knupp, 2000). Each problem comes with an exact reference solution without a separate numerical solve. The Agent can request new problems and obtain problem inputs for existing problems as its experiments require. The Verifier evaluates the Agent's program outputs using the corresponding reference solutions under the task's performance objective and returns numerical feedback. This feedback can also guide the Generator toward new problems that probe the current program's weaknesses. The Agent uses this feedback to revise its program and choose its next experiment. Our work enhances existing PDE agents by providing the experimentation environment in which they develop their programs.

Specifically, our contributions are as follows:

- We construct an experimentation environment from scientific data that supports problem generation, continuous input queries, and evaluation against exact reference solutions, enabling existing PDE agents to conduct experiments throughout program development.
- We establish theoretical guarantees for evaluation reliability and guided development. Under stated assumptions, we bound the error in evaluating a fixed program and prove convergence of idealized guided development in a convex setting.
- We evaluate GAV on Navier–Stokes problems derived from simulations and measurements and on Vlasov–Poisson–Fokker–Planck problems derived from simulations. On the numerical Navier–Stokes task, GAV reduces the mean test error of CodePDE, AutoNumerics, and PDE-SHARP by more than an order of magnitude relative to their original experimental setups. Ablations and development case studies examine how the environment's components affect program quality and the Agent's experimental decisions.

2 PROBLEM SETUP

Inputs. The environment and program-development process are specified by four inputs:

1. **PDE specification.** This specifies a PDE family through its equation form, spatial domain Ω including any internal obstacles, and types of initial and boundary conditions (Li et al., 2025; Dong et al., 2026). Time-dependent problems also specify a time interval. Within this family, a *PDE problem* is represented by $c = (\lambda, f, g)$: the physical parameters λ, source term f, and initial and boundary data g. Its unknown solution u satisfies

$$\mathcal{N}_\lambda[u] = f, \qquad \mathcal{B}[u] = g. \tag{1}$$

   Here $\mathcal{N}_\lambda$ is the PDE operator with parameters λ, and $\mathcal{B}$ encodes the initial and boundary conditions. For example, Navier–Stokes equations describe fluid velocity and pressure, with viscosity as a physical parameter and external forcing as a source term. The Vlasov–Poisson–Fokker–Planck (VPFP) equations describe a particle distribution over position and velocity, coupled to an electric field and subject to collisions.

2. **Data $\mathcal{D}$.** We are given $\mathcal{D} = \{(c_i, \widehat{u}_i)\}_{i=1}^n$, where each problem c_i is paired with its discrete solution data $\widehat{u}_i$. These data contain solution values from numerical simulations or experimental measurements at a finite set of points, together with their spatial coordinates and, for time-dependent problems, times.

3. **Program-development task.** The task specifies the program's inputs, required outputs, and performance objective. We use two tasks as examples:

   Numerical solution. A numerical solver computes an approximation to u from problem inputs (Li et al., 2025). For example, the task may require minimizing solution error subject to a runtime limit per problem.

   Reduced-order modeling. In this task, the program predicts specified macroscopic quantities (e.g., density, momentum, and kinetic-energy density in VPFP) from the problem inputs. The Agent is free to choose the computational method and internal representation; no particular reduced state or macroscopic evolution equation is prescribed. The Agent aims to minimize prediction error for these quantities, subject to any runtime limit specified by the task.

4. **Agent setting.** This specifies the Agent, its development strategy, and its development budget (time and computational resources). We treat this setting as given when studying the environment.

Output. The output of program development is an executable program for the specified task. The Agent develops this program by using the environment to run experiments and obtain numerical feedback, aiming to meet the task requirements within its development budget.

Our goal is to build an environment that supports this process across different PDE specifications, types of data, tasks, and Agent settings. We evaluate the environment through the performance of the final program.

3 GAV: A SCIENTIFIC EXPERIMENTATION ENVIRONMENT

GAV supplies the Agent specified in Section 2 with problems and numerical feedback for developing its program. We first describe how the Generator constructs problems with reference solutions (Section 3.1) and how the Verifier evaluates candidate programs (Section 3.2). We then explain how the Agent uses these capabilities to choose further experiments and improve its program (Section 3.3), completing the interaction in Figure 1.

3.1 GENERATING PDE PROBLEMS WITH REFERENCE SOLUTIONS

The Generator aims to produce new PDE problems from a distribution similar to that of the problems in $\mathcal{D}$. We learn a generative model of continuous reference solutions from the discrete solution data $\widehat{u}_i$. Sampling from this model yields a reference solution $\widetilde{u}$ that can be evaluated at any point in the prescribed domain and differentiated as required by the PDE operator. Representation, training, and sampling details are given in Appendix B.1.

For each generated reference solution, we construct a corresponding PDE problem using the method of manufactured solutions (MMS) (Salari & Knupp, 2000). For prescribed physical parameters λ, we set

$$f = \mathcal{N}_\lambda[\widetilde{u}], \qquad g = \mathcal{B}[\widetilde{u}]. \tag{2}$$

This defines a problem $c = (\lambda, f, g)$ for which $\widetilde{u}$ satisfies Equation (1). Each problem therefore comes with a reference solution without a separate numerical solve. We provide this reference solution to the Verifier for evaluating program outputs. Repeating this construction supplies new problems for further experiments. The continuous representation of $\widetilde{u}$ and the construction in Equation (2) give continuous representations of the source term f and initial and boundary data g. The Agent's program can therefore obtain values of f and g at the locations needed by its numerical method, without being restricted to the resolution of the original data in $\mathcal{D}$.

3.2 Numerical evaluation and feedback

The Verifier evaluates candidate programs under the task's performance objective and returns numerical feedback to the Agent. Because the Generator constructs each problem from a continuous reference solution, the Verifier has an exact reference solution for every generated problem. It uses this reference solution to compute the solution values or derived quantities needed to evaluate the program. For *numerical solution*, it compares the program's predicted solution values with the values of the reference solution at the evaluation points. For *reduced-order modeling*, it compares the program's predicted macroscopic quantities with the same quantities computed from the reference solution at the evaluation points. The program is not required to reconstruct the full solution, but may do so internally to compute these quantities.

During development, the Agent receives the measured errors and runtime, but not the reference solutions used for evaluation. The same evaluation procedure provides feedback for program development and supports program selection and final evaluation.

Reliability of evaluation. We formalize the Verifier's evaluation procedure and provide theoretical guarantees on the accuracy of its loss estimates. We assume that the problems c_i in $\mathcal{D}$ are drawn from a target distribution P. The Generator trained on $\mathcal{D}$ produces new problems from a learned problem distribution $\widehat{P}$, each paired with a reference solution.

For this analysis, we consider a fixed test set of M problems $\widetilde{c}_1, \ldots, \widetilde{c}_M$, sampled independently from $\widehat{P}$. For a candidate program A chosen without using this test set, let $\ell_A(c) \geqslant 0$ denote its task loss on a problem c, computed by comparing the program's outputs with the corresponding quantities obtained from the reference solution. We use the empirical mean loss $\widehat{\mu}_M$ computed on this test set to estimate the expected loss μ_P under the target distribution P:

$$\widehat{\mu}_M = \frac{1}{M}\sum_{j=1}^{M} \ell_A(\widetilde{c}_j), \qquad \mu_P = \mathbb{E}_{c\sim P}[\ell_A(c)].$$

We now present the assumptions and the resulting evaluation guarantee.

Assumption 1. (i) $D_{\mathrm{KL}}(P\|\widehat{P}) \leqslant \varepsilon$ for some $\varepsilon \geqslant 0$. (ii) Let $\mu_{\widehat{P}} = \mathbb{E}_{c\sim\widehat{P}}[\ell_A(c)] \in (0, \infty)$. The normalized loss $\ell_A/\mu_{\widehat{P}}$ is sub-exponential under $\widehat{P}$ with parameters $\nu, \kappa \geqslant 0$, i.e.,

$$\log \mathbb{E}_{\widehat{P}} \exp\left(s\left[\frac{\ell_A}{\mu_{\widehat{P}}} - 1\right]\right) \leqslant \frac{\nu^2 s^2}{2}, \qquad |s| < 1/\kappa. \tag{3}$$

When $\kappa = 0$, the condition applies to every real s.

Proposition 1. *Under Assumption 1, for sufficiently small ε and any $\delta \in (0,1)$, we have with probability at least $1-\delta$,*

$$\frac{|\widehat{\mu}_M - \mu_P|}{\mu_P} \leqslant \frac{a_\varepsilon + b_M}{1 - a_\varepsilon}, \tag{4}$$

where

$$a_\varepsilon = \nu\sqrt{2\varepsilon} + \kappa\varepsilon, \qquad b_M = \nu\sqrt{\frac{2\log(2/\delta)}{M}} + \frac{\kappa\log(2/\delta)}{M}.$$

Here, "sufficiently small" means that $a_\varepsilon < 1$.

See Appendix E.1 for the proof.

Assumption 1(i) requires the learned problem distribution to be close to the target distribution in KL divergence. Under regularity assumptions, flow-matching theory bounds the forward KL divergence in terms of the vector-field approximation error (Wildberger et al., 2023, Theorem 1). Assumption 1(ii) requires that the loss does not vary too much across generated problems relative to its mean (a standard condition satisfied, for example, by bounded losses).

In Proposition 1, the bound on the relative error in estimating μ_P becomes smaller as ε decreases and M increases. Proposition 1 thus supports our Generator–Verifier design: when the learned problem distribution approximates the target distribution and enough independent problems are evaluated, the Verifier can reliably evaluate a fixed program under the target distribution.

3.3 Agent-directed experimentation

Together, the Generator and Verifier provide an experimentation environment in which the Agent can *test on new problems*, *obtain problem inputs at chosen locations and resolutions*, and *receive error feedback computed from exact reference solutions* to improve its program. The Agent can use this feedback to decide which experiments to conduct next.

During development, the Agent may need to investigate specific weaknesses or check for overfitting to the learned problem distribution. We therefore provide two optional extensions to the Generator.

Given a program, *guided generation* seeks out problems on which that program is likely to make larger errors. We fit a predictor to the Verifier's error feedback and use its gradient to guide sampling, adapting the return guidance of Janner et al. (2022). Implementation details are given in Appendix B.2.

Perturbation aims to reduce overfitting to the Generator's learned distribution. We add a small perturbation to a generated reference solution, then use MMS to construct a new problem from the perturbed solution. Implementation details are given in Appendix B.3.

Appendix D.2 examines problem difficulty under guided generation. Appendix D.3 illustrates how perturbation feedback redirects the Agent's development of an NS solver.

Guided development. Guided generation seeks problems with large program errors while staying close to the learned problem distribution. Consider a family of programs A_θ, with parameters $\theta \in \Theta \subset \mathbb{R}^d$ and task loss $\ell_\theta(c) = \ell_{A_\theta}(c)$. For a fixed finite KL radius $\rho \geqslant 0$, this gives the worst-case objective

$$F(\theta) = \sup_{R:\, D_{\mathrm{KL}}(R\|\widehat{P})\leqslant\rho} \mathbb{E}_{c\sim R}[\ell_\theta(c)]. \tag{5}$$

For the current program A_{θ_t}, replacing the KL constraint by a penalty gives the objective $\mathbb{E}_R \ell_{\theta_t} - \beta_t^{-1} D_{\mathrm{KL}}(R\|\widehat{P})$. For bounded losses and $\beta_t > 0$, its maximizer is the exponential tilt

$$\frac{dR_t}{d\widehat{P}}(c) = \frac{\exp\{\beta_t \ell_{\theta_t}(c)\}}{\mathbb{E}_{\widehat{P}} \exp\{\beta_t \ell_{\theta_t}\}}. \tag{6}$$

When β_t is chosen so that $D_{\mathrm{KL}}(R_t\|\widehat{P}) = \rho$, this law also solves the constrained problem in (5). The reformulation provides a sampling target for guided generation (Feng et al., 2025).

We analyze an idealized setting in which each guidance step returns this exact constrained maximizer, with a finite calibrated $\beta_t > 0$ when $\rho > 0$. For $\rho = 0$, take $R_t = \widehat{P}$ and $\beta_t = 0$. Let $f_t(\theta) = \mathbb{E}_{c\sim R_t}[\ell_\theta(c)]$ denote the expected loss on the guided problems; exact maximization gives $f_t(\theta_t) = F(\theta_t)$. Starting from $\theta_0 \in \Theta$, the Agent revises its program with a fixed stabilization parameter $\tau > 0$:

$$\theta_{t+1} \in \arg\min_{\theta\in\Theta} \left\{ f_t(\theta) + \frac{\tau}{2}\|\theta - \theta_t\|^2 \right\}. \tag{7}$$

Assumption 2. (i) The set Θ is nonempty, compact, and convex, and F is convex on Θ.

(ii) The losses are uniformly bounded under $\widehat{P}$ and G-Lipschitz continuous in θ, for some finite $G \geqslant 0$, on a common set of problems of full $\widehat{P}$-measure.

Theorem 1. *Under Assumption 2, consider the idealized guided updates in* (7)*, with* $0 \leqslant \beta_t \leqslant \bar{\beta} < \infty$ *for every* $t \geqslant 0$ *and* $\tau > \bar{\beta}G^2$*. Then* $F(\theta_{t+1}) \leqslant F(\theta_t)$ *for all* $t \geqslant 0$*. Let* $F^\star = \min_{\theta\in\Theta} F(\theta)$ *denote the minimum objective value over this program family. For every integer* $T \geqslant 1$*, the difference between* $F(\theta_T)$ *and* $F^\star$ *satisfies*

$$F(\theta_T) - F^\star \leqslant \frac{2\tau \operatorname{diam}(\Theta)^2}{T+1}, \tag{8}$$

where $\operatorname{diam}(\Theta) = \sup_{\theta,\theta'\in\Theta} \|\theta - \theta'\|$.

See Appendix E.2 for the proof.

The bound on tilting strength, together with Lipschitz losses, gives $F(\theta) - f_t(\theta) \leqslant \bar{\beta}G^2\|\theta - \theta_t\|^2/2$ by Lemma 1. The stabilization therefore controls the change in worst expected loss as the Agent improves its program on the current guided problems.

4 Experiments

In this section, we evaluate the effectiveness of GAV as an experimentation environment for PDE agents. We first assess the accuracy of programs developed with GAV for different PDEs and scientific tasks. We then examine whether access to GAV improves the programs developed by existing PDE agents. Finally, we analyze how individual components of the environment affect program performance.

4.1 Experimental setup

Following the setup in Section 2, we specify the PDEs, tasks, data sources, and agent settings used in our experiments, and describe how we evaluate the resulting programs.

PDEs and tasks. We consider two PDE families: the two-dimensional incompressible Navier–Stokes (NS) equations and Vlasov–Poisson–Fokker–Planck (VPFP) equations. The NS equations describe viscous fluid flow (Appendix A.1). The task is to develop a numerical solver that predicts velocity at requested space–time locations from the initial velocity, boundary data, and forcing on the specified geometry. The VPFP equations describe particles evolving under a self-consistent electric field and collisions (Appendix A.2). The task is to develop a reduced-order model (ROM) that predicts density, momentum, and kinetic-energy density from initial distributions, sources, and physical parameters. For both tasks, the objective is to minimize prediction error within the prescribed development budget.

Data sources. We use two NS datasets and one VPFP dataset, developing a separate program for each. For NS, we obtain discrete solution data by numerical simulation in a square domain containing two fixed obstacles, with varying initial conditions and forcing. We also use particle image velocimetry (PIV) measurements of a cylinder wake (Tu et al., 2014). For VPFP, we use numerical simulations with collision rate $\nu_{\mathrm{c}} \in [10, 30]$ and temperature parameter $T_{\mathrm{v}} \in [0.8, 1.2]$. The geometries, data sources, and preparation procedures are described alongside the equations in Appendix A.

Agent setting. For the experiments in Sections 4.2 and 4.4, two agents develop programs in parallel over multiple *rounds* (Figure 2). Both agents start each study from the same initial program. At the start of each round, both agents receive the current program and a cumulative memory log from previous rounds. The log preserves the agents' attempted approaches and observations, together with evaluation results and program-selection decisions. One explores alternative approaches, while the other improves the current approach. Each agent conducts experiments and submits a candidate program within its allotted time. At the end of the round, the candidates and the current program are evaluated on the same problems; the selected program becomes the starting point for the next round.

Evaluation. For both NS studies, the objective is to minimize the mean relative ℓ_2 velocity error over M test problems under a prescribed execution budget:

$$\mathrm{Error} = \frac{1}{2M}\sum_{i=1}^{M}\sum_{k=1}^{2}\frac{\|u_{i,k}^{\mathrm{pred}} - \widetilde{u}_{i,k}\|_2}{\|\widetilde{u}_{i,k}\|_2}. \tag{9}$$

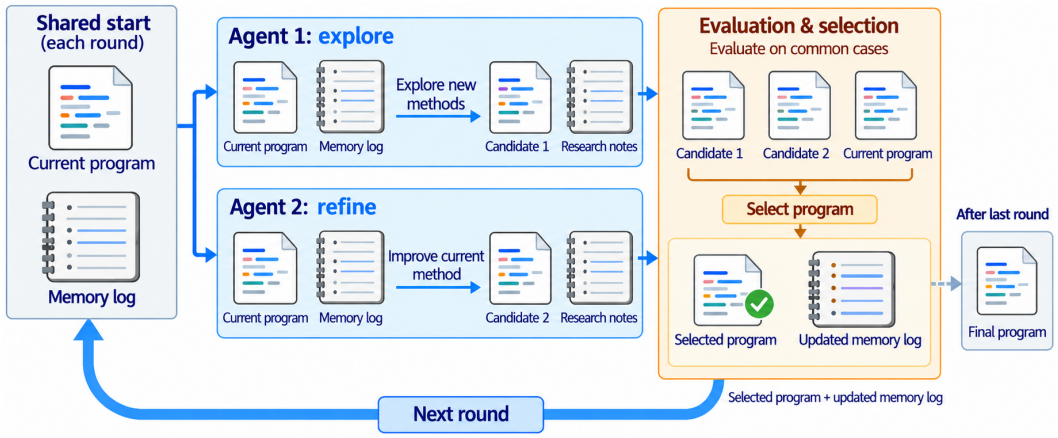

Figure 2: **Multi-turn setting.** The selected program and updated memory log pass to the next round; research notes from both agents are retained regardless of the selection.

Here $u_{i,k}^{\mathrm{pred}}$ and $\widetilde{u}_{i,k}$ denote the predicted and reference values of velocity component k at the evaluation points for problem i.

For VPFP, we average relative ℓ_2 errors over density, vector momentum, and kinetic-energy density. The reference moments are computed from the generated solutions. Scores are averaged across test problems. Appendix C.2 describes program selection, test sets, and evaluation points.

4.2 Program development across PDE tasks

We study how agents develop programs with GAV across PDE tasks and data sources. We use *GAV-Full*, with guided generation and perturbation enabled, on the two NS datasets (numerical simulations and cylinder-wake PIV measurements) and the VPFP simulation dataset. All three experiments use the agent setting in Section 4.1 over five rounds.

Figure 3 shows that GAV supports continued program improvement across PDE families, data sources, and tasks, from NS solvers to VPFP reduced-order models. Appendix D.3 illustrates how agents use feedback to revise their programs.

We construct independent numerical NS and VPFP test problems directly from the original data sources, allowing us to assess performance in the physical regimes of interest (Appendix D.1). The results are consistent with our main evaluation, providing evidence against overfitting to the learned Generator's problem distribution and supporting the programs' accuracy in these regimes.

We visualize the solutions and their errors in Figure 4 and Appendix D.4. Figure 4 compares the first-round (R1) and final (R5) programs on one predetermined PIV-derived manufactured problem. The final program substantially reduces the spatial velocity errors at the five displayed times, with the full-window vector relative ℓ_2 error decreasing from 0.0154% to 0.00172%.

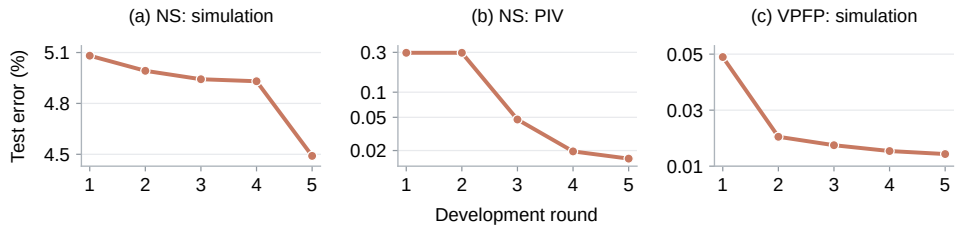

Figure 3: **Program development with GAV-Full.** Mean test error (%) over five rounds on 768 fixed problems per dataset: velocity error for NS and mean error in density, vector momentum, and kinetic-energy density for VPFP. Lower is better; (b) uses a log scale.

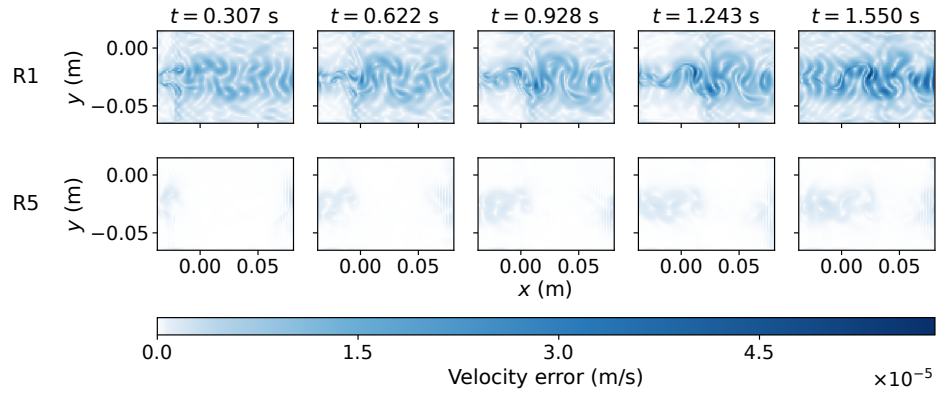

Figure 4: **Velocity error on a PIV-derived manufactured problem.** Rounds 1 (top) and 5 (bottom) use the same generated reference at five times. Color shows $\|\mathbf{u}^{\text{pred}} - \widetilde{\mathbf{u}}\|_2$ (m/s) on one full-window linear scale: white at zero, darker blue for larger errors.

4.3 GAV WITH EXISTING PDE AGENTS

GAV is an experimentation environment designed to support PDE agents with different models and development strategies. We next evaluate whether GAV improves the performance of existing PDE agents. We adapt CodePDE (Li et al., 2025), AutoNumerics (Du et al., 2026), and PDE-SHARP (Fazliani & Udell, 2025) to the NS task with numerical simulation data and follow each method's original development strategy. Within each method, we compare its original setup (Original) with two configurations of our environment. Original uses a fixed collection of 64 problems with inputs sampled on prescribed grids. GAV additionally allows agents to request new problems and query inputs at chosen coordinates. GAV-Full also enables guided generation and perturbation. We evaluate all final programs on the same 96 held-out problems. Appendix C.3 describes how each method uses the environment.

GAV and GAV-Full achieve lower test error than Original for all three PDE agents (Figure 5). GAV reduces mean test error by more than an order of magnitude. Both variants also use less total agent active time than Original (Appendix C.3, Table 6).

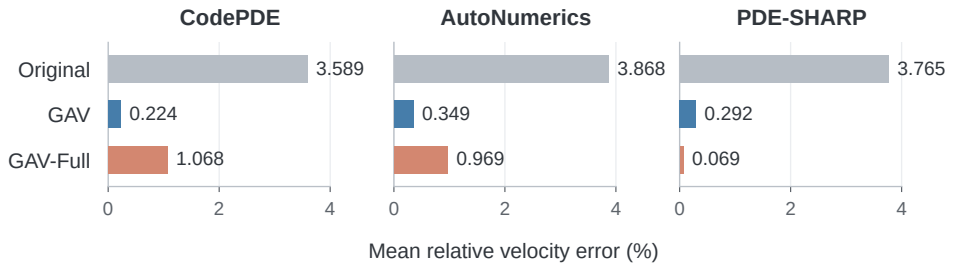

Figure 5: Mean relative velocity error (%) on 96 numerical NS test problems; lower is better.

4.4 ABLATION STUDIES

Three key features support experimentation in GAV: learning from data to represent the target physical regime, MMS construction to provide exact references without a separate numerical solve, and continuous input queries at locations and resolutions chosen by the Agent. We examine how each feature contributes to program development.

Learning from data. We replace the learned reference distribution with a Gaussian-process prior while keeping MMS and continuous input queries. This tests the role of learning the reference distribution from the target scientific data.

MMS construction. MMS constructs problem inputs from an exact reference solution, avoiding a separate numerical solve. We keep the learned problem distribution unchanged and numerically solve each generated problem to obtain an approximate reference for development feedback. This requires additional computation time and introduces numerical error.

Continuous input queries. Without continuous input queries, agents receive initial velocity and forcing values only on a fixed 64×64 spatial grid during development. They cannot directly query the input fields at locations or spatial resolutions of their choice, but may interpolate the provided grid values. We keep the learned Generator and MMS.

We compare GAV with the three ablation variants on the NS task using the multi-turn setting in Section 4.1. All runs start from the same initial program and use three development rounds with a two-hour budget per agent per round. Table 1 reports errors on the same 96 numerical NS test problems used in Section 4.3. Appendix C.4 gives the implementation details.

Table 1: Mean relative velocity error (%) after three rounds on 96 common NS test problems. Error / GAV is the ratio to GAV; lower is better.

Environment	Mean error (%)	Error / GAV
GAV	**0.441**	1.0×
w/o learning from data	1.468	3.3×
w/o MMS construction	0.988	2.2×
w/o continuous input queries	5.211	11.8×

Compared with GAV, the conditions without learning from data, MMS construction, or continuous input queries have 3.3, 2.2, and 11.8 times the mean error, respectively. These results support GAV's core combination of learning reference solutions from data, constructing matching problems through MMS, and providing continuous input queries.

5 Related work

PDE agents. Recent work uses language model agents to automate PDE solver development through numerical reasoning, code generation, and iterative refinement (Li et al., 2025; Fazliani & Udell, 2025; Dong et al., 2026; Du et al., 2026). Related systems automate simulation workflows using established scientific software, including OpenFOAM and finite-element tools (Chen et al., 2024; Yue et al., 2025; Deotale et al., 2026). Within solver development, Li et al. (2025) study inference-time search and refinement, while Fazliani & Udell (2025) use mathematical analysis and solver synthesis to organize candidate generation and evaluation. Dong et al. (2026) explicitly represent the solver strategy and revise it using numerical evidence. Our work enhances the capabilities of existing PDE agents through the design of an experimentation environment.

Environments for scientific agents. Recent work has developed environments and benchmarks to support and evaluate agents on scientific tasks (Huang et al., 2024; Chen et al., 2025; Nathani et al., 2025; Bragg et al., 2026). Within this direction, Gandhi et al. (2025) and Kabra et al. (2026) study how agents choose experiments and use the resulting observations to revise scientific models. For scientific code generation, Zeng et al. (2026) connect testing to program improvement by jointly refining test cases and candidate programs. GAV advances this direction by turning scientific data into a continuing source of PDE experiments with reliable numerical feedback, without running a reference solver for every new problem.

6 Discussion

We introduced GAV, a scientific experimentation environment for PDE agents. It turns scientific data into new PDE problems with reliable numerical feedback, supporting agents as they test ideas, investigate failures, and refine their programs. We establish theoretical guarantees for evaluation and guided development under stated conditions, and demonstrate substantial gains for existing PDE agents. These results highlight environment design as a way to enhance agents' ability to solve scientific problems.

We view GAV as a first step toward scientific experimentation environments for agents across disciplines. Future work will extend the Generator–Agent–Verifier framework beyond PDEs, with the Generator providing scientific problems, the Agent developing candidate solutions, and the Verifier supplying reliable feedback. Our broader goal is to enhance scientific agents by designing environments in which they can test ideas and improve their solutions.

AI USE STATEMENT

As this work studies AI agents for PDE tasks, we use `gpt-5.6-sol` as the agent model in our experiments. We also used AI tools to polish the manuscript and prepare scientific figures.

REPRODUCIBILITY STATEMENT

Appendix A describes the PDEs and data sources, and Appendix B details the environment construction and training settings. Appendix C provides the agent settings, prompt templates, development budgets, and protocols for program selection and evaluation. Proofs of the theoretical results are provided in Appendix E. Anonymized code is provided in the supplementary material.

REFERENCES

Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston, Anton Kast, Cory Y. McLean, Peter Norgaard, Zahra Shamsi, David Smalling, James Thompson, Subhashini Venugopalan, Brian P. Williams, Chujun He, Sarah Martinson, Martyna Plomecka, Lai Wei, Yuchen Zhou, Qian-Ze Zhu, Matthew Abraham, Erica Brand, Anna Bulanova, Jeffrey A. Cardille, Chris Co, Scott Ellsworth, Grace Joseph, Malcolm Kane, Ryan Krueger, Johan Kartiwa, Dan Liebling, Jan-Matthis Lueckmann, Paul Raccuglia, Xuefei Julie Wang, Katherine Chou, James Manyika, Yossi Matias, John C. Platt, Lizzie Dorfman, Shibl Mourad, and Michael P. Brenner. An AI system to help scientists write expert-level empirical software. *Nature*, 654:909–916, 2026. doi: 10.1038/s41586-026-10658-6. URL https://www.nature.com/articles/s41586-026-10658-6.

Rangel Baldasso, Alan Pereira, and Guilherme Reis. Large deviations for interacting diffusions with path-dependent McKean–Vlasov limit. *The Annals of Applied Probability*, 32(1):665–695, 2022. doi: 10.1214/21-AAP1692. URL https://arxiv.org/abs/1912.00440v2.

Jonathan Bragg, Mike D'Arcy, Nishant Balepur, Dan Bareket, Bhavana Dalvi, Sergey Feldman, Dany Haddad, Jena D. Hwang, Peter Jansen, Varsha Kishore, Bodhisattwa Prasad Majumder, Aakanksha Naik, Sigal Rahamimov, Kyle Richardson, Amanpreet Singh, Harshit Surana, Aryeh Tiktinsky, Rosni Vasu, Guy Wiener, Chloe Anastasiades, Stefan Candra, Jason Dunkelberger, Dan Emery, Rob Evans, Malachi Hamada, Regan Huff, Rodney Kinney, Matt Latzke, Jaron Lochner, Ruben Lozano-Aguilera, Cecile Nguyen, Smita Rao, Amber Tanaka, Brooke Vlahos, Peter Clark, Doug Downey, Yoav Goldberg, Ashish Sabharwal, and Daniel S. Weld. AstaBench: Rigorous benchmarking of AI agents with a scientific research suite. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.21652v2.

Yuxuan Chen, Xu Zhu, Hua Zhou, and Zhuyin Ren. MetaOpenFOAM: An LLM-based multi-agent framework for CFD. arXiv preprint arXiv:2407.21320, 2024. URL https://arxiv.org/abs/2407.21320v1.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. ScienceAgentBench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/f12b4df26344f3be803c06b555252efe-Abstract-Conference.html.

Rushikesh Deotale, Adithya Srinivasan, Mahmoud Golestanian, Yuan Tian, Tianyi Zhang, Pavlos Vlachos, and Hector Gomez. ALL-FEM: Agentic large language models fine-tuned for finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 457:118985, 2026. doi: 10.1016/j.cma.2026.118985. URL https://doi.org/10.1016/j.cma.2026.118985.

Huanshuo Dong, Keyao Zhang, Hong Wang, Zhezheng Hao, Zhiwei Zhuang, Ziyan Liu, Jiacong Wang, Gengyuan Liu, and Xin Jin. AutoPDE: Reliable agentic PDE solving via explicitly represented solver strategies. arXiv preprint arXiv:2606.10752, 2026. URL https://arxiv.org/abs/2606.10752v1.

Jianda Du, Youran Sun, and Haizhao Yang. AutoNumerics: An autonomous, PDE-agnostic multi-agent pipeline for scientific computing. arXiv preprint arXiv:2602.17607, 2026. URL https://arxiv.org/abs/2602.17607v1.

Shaghayegh Fazliani and Madeleine Udell. PDE-SHARP: PDE solver hybrids through analysis and refinement passes. arXiv preprint arXiv:2511.00183, 2025. URL https://arxiv.org/abs/2511.00183v2.

Ruiqi Feng, Chenglei Yu, Wenhao Deng, Peiyan Hu, and Tailin Wu. On the guidance of flow matching. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 16993–17029. PMLR, 2025. URL https://proceedings.mlr.press/v267/feng25s.html.

Kanishk Gandhi, Michael Y. Li, Lyle Goodyear, Agam Bhatia, Louise Li, Aditi Bhaskar, Mohammed Zaman, and Noah D. Goodman. BoxingGym: Benchmarking progress in automated experimental design and model discovery. In *NeurIPS Workshop on Scaling Environments for Agents*, 2025. URL https://arxiv.org/abs/2501.01540v2.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 20271–20309. PMLR, 2024. URL https://proceedings.mlr.press/v235/huang24y.html.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9902–9915. PMLR, 2022. URL https://proceedings.mlr.press/v162/janner22a.html.

Sanchit Kabra, Nikhil Abhyankar, Saaketh Desai, Prasad P. Iyer, and Chandan K. Reddy. LLM-AutoSciLab: Closed-loop scientific discovery via active experimentation with LLMs. arXiv preprint arXiv:2605.24043, 2026. URL https://arxiv.org/abs/2605.24043v1.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022.

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

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*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XVjTT1nw5z.

D. Moxey, C. D. Cantwell, Y. Bao, A. Cassinelli, G. Castiglioni, S. Chun, E. Juda, E. Kazemi, K. Lackhove, J. Marcon, G. Mengaldo, D. Serson, M. Turner, H. Xu, J. Peiró, R. M. Kirby, and S. J. Sherwin. Nektar++: Enhancing the capability and application of high-fidelity spectral/hp element methods. *Computer Physics Communications*, 249:107110, 2020. doi: 10.1016/j.cpc.2019.107110.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. MLGym: A new framework and benchmark for advancing AI research agents. arXiv preprint arXiv:2502.14499, 2025. URL https://arxiv.org/abs/2502.14499v1.

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/2506.13131v1.

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

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco 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*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

Kambiz Salari and Patrick Knupp. Code verification by the method of manufactured solutions. Technical Report SAND2000-1444, Sandia National Laboratories, 2000. URL https://digital.library.unt.edu/ark:/67531/metadc702130/.

Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, 2000. doi: 10.1137/1.9780898719598. URL https://doi.org/10.1137/1.9780898719598.

Jonathan H. Tu, Clarence W. Rowley, J. Nathan Kutz, and Jessica K. Shang. Spectral analysis of fluid flows using sub-nyquist-rate PIV data. *Experiments in Fluids*, 55(9):1805, 2014. doi: 10.1007/s00348-014-1805-6. URL https://doi.org/10.1007/s00348-014-1805-6.

Jonas Wildberger, Maximilian Dax, Simon Buchholz, Stephen Green, Jakob H. Macke, and Bernhard Schölkopf. Flow matching for scalable simulation-based inference. In *Advances in Neural Information Processing Systems*, volume 36, pp. 16837–16864, 2023. doi: 10.52202/075280-0737. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/3663ae53ec078860bb0b9c6606e092a0-Paper-Conference.pdf.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist-v2: Workshop-level automated scientific discovery via agentic tree search. arXiv preprint arXiv:2504.08066, 2025. URL https://arxiv.org/abs/2504.08066v1.

Haotian Ye, Haowei Lin, Jingyi Tang, Yizhen Luo, Rahul Thapa, Caiyin Yang, Chang Su, Rui Yang, Ruihua Liu, Rundao Li, Zeyu Li, Pengwei Sun, Chong Gao, Dachao Ding, Guangrong He, Miaolei Zhang, Lina Sun, Wenyang Wang, Yuchen Zhong, Zhuohao Shen, Puheng Li, Pan Lu, Bianxiao Cui, Di He, Jianzhu Ma, Junfeng Li, Hexi Baoyin, Yejin Choi, Stefano Ermon, Xiaowen Chu, Tongyang Li, Yuzhi Xu, and James Zou. Structured scaling of AI discovery across diverse scientific domains. arXiv preprint arXiv:2604.19341, 2026. URL https://arxiv.org/abs/2604.19341v2.

Ling Yue, Nithin Somasekharan, Tingwen Zhang, Yadi Cao, and Shaowu Pan. Foam-Agent: An end-to-end composable multi-agent framework for automating CFD simulation in OpenFOAM. arXiv preprint arXiv:2509.18178, 2025. URL https://arxiv.org/abs/2509.18178v1.

Zihang Zeng, Jiaquan Zhang, Pengze Li, Yuan Qi, and Xi Chen. AI-for-science low-code platform with bayesian adversarial multi-agent framework. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Cug26Y0RlT.

A PDEs AND DATA SOURCES

A.1 NAVIER–STOKES EQUATIONS

On a two-dimensional fluid domain Ω, the velocity $\mathbf{u}(t, \mathbf{x})$ and pressure $p(t, \mathbf{x})$ satisfy

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \nu \Delta \mathbf{u} + \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0. \tag{10}$$

Here $\nu > 0$ is the kinematic viscosity, $\mathbf{f}(t, \mathbf{x})$ is the forcing, and the divergence constraint enforces incompressibility. Each problem specifies the domain, viscosity, initial velocity $\mathbf{u}(0, \mathbf{x}) = \mathbf{u}_0(\mathbf{x})$, and boundary conditions.

Numerical data. We obtain discrete solution data by numerically solving 100,000 problems in a unit square containing a fixed kite-shaped obstacle and a triangular baffle (Figure 6a). The outer walls and both obstacles impose the no-slip condition $\mathbf{u} = 0$, and the viscosity is $\nu = 0.01$. Initial velocities and forcing vary across problems, with ten families of forcing that include shear, rotating vortices, and localized pulses. We integrate the equations with a finite-difference projection method, choosing time steps from advective and diffusive stability limits. For each problem, we sample velocity and pressure on a 64×64 cell-centered grid at 32 uniformly spaced times in $[0, 0.12]$.

We then use MMS to construct matching forcing from the sampled velocity and pressure, approximating derivatives by finite differences. This gives a dataset of 100,000 problems, each paired with the discrete solution data used to construct it.

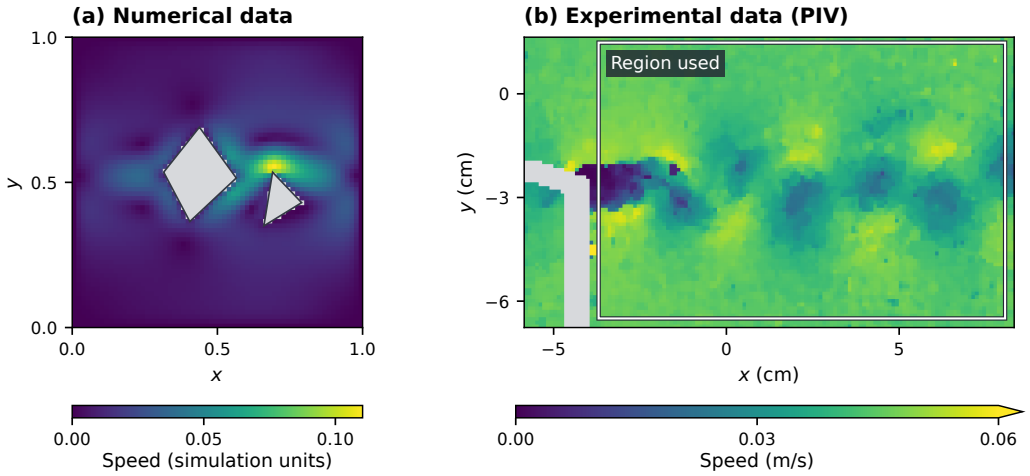

Figure 6: **Navier–Stokes data sources.** Examples of discrete solution data used to construct the environment. Color shows fluid speed at a single time. (a) Numerical simulation in a square domain with two solid obstacles (gray). (b) PIV measurements of flow behind a cylinder. The white rectangle marks the measurement region used to construct problems; gray areas indicate missing measurements. Each panel uses its own color scale; PIV speeds above 0.06 m/s use the top color.

Experimental data. We use planar particle image velocimetry (PIV) measurements of flow past a cylinder in a water channel at Reynolds number 413, described in Tu et al. (2014). The dataset contains 8,000 velocity fields sampled at 20 Hz, each with two velocity components on a 135×80 grid. For problems constructed from these measurements, we use a kinematic viscosity of $\nu \approx 1.0 \times 10^{-6}\,\mathrm{m}^2/\mathrm{s}$.

To construct problems from these measurements, we first retain every second grid point along each spatial direction. We then select the largest rectangle whose grid points all have valid measurements at every time. The selected region contains 57×39 measurement locations and no solid obstacles (Figure 6b). For each problem, velocity measurements at 32 consecutive sampling times provide the discrete solution data over a 1.55 s interval.

Pressure and forcing are not measured. To form the paired problem inputs in Section 2, we take initial and time-dependent boundary velocities from each observation window and compute matching forcing from the measured velocity, using zero pressure as a construction convention.

A.2 Vlasov–Poisson–Fokker–Planck equations

The VPFP equations describe a particle distribution $F(t, \mathbf{x}, \mathbf{v}) \geqslant 0$ coupled to a self-consistent electric field and subject to collisions. In dimensionless form, the system is

$$\partial_t F + \mathbf{v} \cdot \nabla_{\mathbf{x}} F + \mathbf{E} \cdot \nabla_{\mathbf{v}} F = \nu_{\mathrm{c}} \nabla_{\mathbf{v}} \cdot \big(\mathbf{v} F + T_{\mathrm{v}} \nabla_{\mathbf{v}} F \big) + S, \tag{11}$$

$$-\Delta_{\mathbf{x}} \phi = \rho - \rho_{\mathrm{b}}, \qquad \mathbf{E} = -\nabla_{\mathbf{x}} \phi, \qquad \rho = \int_{\mathbb{R}^2} F \, \mathrm{d}\mathbf{v}. \tag{12}$$

Here ρ is the particle density, $\rho_{\mathrm{b}} = 1$ is the uniform neutralizing background, and S is a source term. The electric field is written as an acceleration, with the charge convention absorbed into its definition. The parameter $\nu_{\mathrm{c}} > 0$ is the collision rate and $T_{\mathrm{v}} > 0$ is the temperature parameter of the linear Fokker–Planck operator, which combines velocity drag and diffusion.

The VPFP study uses two spatial and two velocity dimensions, with $\mathbf{x} \in [0, 2\pi)^2$, $\mathbf{v} \in \mathbb{R}^2$, $\nu_{\mathrm{c}} \in [10, 30]$, and $T_{\mathrm{v}} \in [0.8, 1.2]$. We use periodic spatial boundaries, a zero-mean electric field, and $t \in [0, 1]$. Finite velocity windows used to sample inputs do not impose boundaries on the reference distribution, whose velocity moments are defined over $\mathbb{R}^2$.

Program inputs and outputs. The program receives the equation parameters, initial distribution $F(0, \mathbf{x}, \mathbf{v}) = F_0(\mathbf{x}, \mathbf{v})$, and source $S(t, \mathbf{x}, \mathbf{v})$, and predicts the requested macroscopic fields. The initial distribution is supplied as density times a normalized conditional velocity distribution. The outputs are density, the two components of momentum density, and kinetic-energy density:

$$\rho = \int F \, \mathrm{d}\mathbf{v}, \qquad \mathbf{m} = \int \mathbf{v} F \, \mathrm{d}\mathbf{v}, \qquad K = \frac{1}{2} \int |\mathbf{v}|^2 F \, \mathrm{d}\mathbf{v}. \tag{13}$$

The Agent chooses the computational method and internal representation, as in Section 2.

Numerical data. We draw the collision rate and temperature independently and uniformly from the ranges above. The initial density is a normalized exponential Fourier field with four random modes, wave numbers up to four, and Gaussian coefficients of standard deviation 0.1 in log density. The conditional velocity distribution is a spatially varying mixture of four to ten anisotropic Gaussian components, with the component count sampled uniformly. Their base centers are uniform on $[-2, 2]^2$, their base standard deviations are log-uniform on $[0.65, 1.8]$, and their correlations are uniform on $[-0.6, 0.6]$. Periodic Fourier fields modulate their weights, centers, and widths. The simulation source has twelve signed Gaussian–Hermite wave packets, with velocity polynomial degree at most three and a common amplitude scale sampled uniformly from $[5 \times 10^{-4}, 5 \times 10^{-3}]$. The numerical solver uses Fourier differentiation in space, conservative finite-volume velocity fluxes with a Chang–Cooper discretization of the collision term, and fourth-order Runge–Kutta time integration. Each trajectory is sampled at 33 uniformly spaced times on a 16×16 spatial grid and a 21×21 velocity grid covering $[-8, 8]^2$. The internal time step is capped at 10^{-3} and further restricted by the stability estimate with CFL factor 0.18. The numerical trajectories are projected into the continuous representation described in Appendix B.1.2. The projected distribution then determines matching source terms through MMS. Generator fitting and evaluation use separate trajectory splits, specified in that appendix.

B Environment construction

B.1 Reference solutions and problem generation

We construct a separate Generator for each data source. For each data source, we construct continuous reference solutions informed by the data, define a sampling distribution, and use MMS to construct new problems from sampled reference solutions, as in Equation (2).

B.1.1 NS GENERATOR

Data splits. For the numerical data in Appendix A.1, we use $80{,}000$ problems to fit the Generator. We draw validation data from a separate set of $10{,}000$ problems to choose its settings. The remaining $10{,}000$ problems are excluded from Generator fitting.

For PIV, velocity measurements at the first $3{,}600$ sampling times provide training data, and those at the next 800 provide validation data for choosing the Generator's settings. Each problem uses measurements at 32 consecutive times; shifting the start by four sampling intervals yields 893 training problems and 193 validation problems. The training and validation sets use disjoint time intervals, although problems within each set share measurements. Later measurements are excluded from Generator fitting and selection. Table 2 gives the Generator settings for the two NS studies.

Continuous representation. We represent the velocity through a streamfunction ψ, with $\widetilde{\mathbf{u}} = (\partial_y \psi, -\partial_x \psi)$, so that incompressibility holds by construction (Raissi et al., 2019). We use a spectral representation (Trefethen, 2000): tensor products of spatial cosine functions and temporal Chebyshev polynomials for both the streamfunction and pressure. The spatial basis includes the constant function. We evaluate the spatial basis in coordinates normalized to $[0, 1]$, while taking velocity derivatives with respect to physical coordinates. For problems generated from the numerical NS data, the domain extends to the outermost grid centers, with coordinate bounds $[1/128, 127/128]^2$.

For the numerical data, we multiply the streamfunction by a boundary mask that enforces zero velocity on the outer walls and obstacle surfaces. The mask rises from zero at a solid boundary to one in the fluid interior, with a transition width of 0.05 in normalized coordinates. Both the mask and its first spatial derivatives vanish on each solid boundary, so differentiating the masked streamfunction gives zero velocity there. For PIV, the observation rectangle lies within the fluid and allows inflow and outflow. We extend the cosine basis interval to $[-0.25, 1.25]$ along each normalized spatial axis to allow nonzero boundary velocities. The observation rectangle remains $[0, 1]^2$, and its boundary velocities are taken from the reference solution and may vary with time.

For problem i, let $\widehat{\mathbf{u}}_i(t_j, \mathbf{x}_m)$ denote the observed velocity at time t_j and location $\mathbf{x}_m$. We omit spatial basis functions whose velocity contribution at the observation points is zero or numerically negligible. At each sampled time, we fit the remaining spatial coefficients by

$$a_j = \arg\min_a \left\{ \sum_m \|\widetilde{\mathbf{u}}_a(\mathbf{x}_m) - \widehat{\mathbf{u}}_i(t_j, \mathbf{x}_m)\|_2^2 + \eta \|a\|_2^2 \right\}.$$

Here $\widetilde{\mathbf{u}}_a$ is the velocity obtained from the streamfunction with spatial coefficients a; the sum runs over observed fluid locations. The penalty $\eta\|a\|_2^2$ discourages large coefficients. For each spatial basis function used in this fit, we evaluate its velocity contribution with unit coefficient and sum the squared components over the observation points. In both studies, we set η to 10^{-6} times the median of these sums, so that the penalty scales with the velocity basis. We then fit the coefficients a_j across sampled times with Chebyshev polynomials by regularized least squares, with the penalty computed from the temporal basis in the same way. This gives a streamfunction that is continuous in both space and time. For the numerical data, pressure is fitted after removing its spatial mean at each time. For PIV, pressure is set to zero, following the convention in Appendix A.1.

Learning the distribution. Let w_i be the vector of K fitted space–time streamfunction coefficients for training problem i, with N training problems. We subtract the training mean and divide by a shared root-mean-square scale:

$$\bar{w} = \frac{1}{N} \sum_{i=1}^{N} w_i, \qquad \sigma = \left(\frac{1}{NK} \sum_{i=1}^{N} \|w_i - \bar{w}\|_2^2 \right)^{1/2}, \qquad \widetilde{w}_i = \frac{w_i - \bar{w}}{\sigma}.$$

Here $\bar{w}$ averages each coefficient across problems, while the scalar σ pools all K coefficients and all N problems. For the numerical data, we normalize pressure separately using its own mean and scale; PIV pressure remains zero. This expresses each solution relative to the training mean and balances the two fields while preserving amplitude differences across problems.

We concatenate the scaled field coefficients and apply principal component analysis (PCA), retaining the directions with the largest variation across training problems. We choose the smallest tested

Table 2: **NS Generator settings.**

Setting	Numerical data	PIV data
Cosine basis functions per spatial axis	17	
Chebyshev polynomial degree in time	12	
Retained principal components	256	32
Batch size	512	128
Training steps	30,000	5,000
Checkpoint used (training step)	15,000	5,000

number of components whose median validation velocity reconstruction error is within 5% of the best value. Finally, we divide each retained PCA coordinate by its training standard deviation to obtain a whitened vector z. We learn the distribution of these vectors with rectified flow (Liu et al., 2023), training a flow velocity V by minimizing

$$\mathbb{E}\Big[\|V(z_s, s) - (z - \epsilon)\|_2^2\Big], \qquad z_s = (1-s)\epsilon + sz, \tag{14}$$

Here z is sampled from the whitened training vectors, ϵ is a standard Gaussian vector of the same dimension, and $s \sim \mathrm{Unif}[0,1]$; these draws are independent. The flow time s is distinct from physical time t. At generation, we integrate the learned flow from Gaussian noise at $s = 0$ to $s = 1$ using 256 Heun steps. We undo the whitening, reconstruct the coefficients in the retained PCA basis, and restore the field scales and training mean. The resulting coefficients define a continuous reference solution.

The flow model is a residual multilayer perceptron with eight blocks of width 768, SiLU activations, and layer normalization, conditioned on flow time. We use AdamW with a peak learning rate of 2×10^{-4}, $1{,}000$ warmup steps, and cosine decay to one tenth of the peak rate. The Generator is fitted before program development and remains fixed during development; guided generation modifies sampling as described in Appendix B.2.

B.1.2 VPFP Generator

Data splits. We use $2{,}048$ numerical trajectories for Generator fitting and 128 to monitor validation denoising loss. Three further sets of 128 trajectories each are held out for calibration and reference-distribution diagnostics. The splits contain distinct equation seeds, and coefficient normalization uses only the training trajectories.

Continuous representation. We write $\widetilde{F} = \widetilde{\rho}\,\widetilde{g}$, where $\widetilde{g}$ is a normalized conditional velocity density. For $\widetilde{\rho}$, we fix the constant Fourier mode to one and rescale the nonconstant coefficients so that the magnitude of the density fluctuation stays below one. This preserves positivity and spatial mean density one. For $\widetilde{g}$, we square an expansion in 120 tensor-product Hermite functions of total degree at most 14 and divide the result by its integral over $\mathbb{R}^2$. Orthonormality makes this integral the squared norm of the Hermite coefficient vector. We mix the normalized density with a Maxwellian of weight 10^{-5} to keep it strictly positive. We fit the Hermite amplitude to the square root of the discrete conditional velocity density by weighted least squares. Spatial Fourier expansions then represent the fitted coefficient fields, with mode orders six for density and four for the Hermite coefficients. Piecewise cubic Hermite interpolation through 33 control times supplies continuous time dependence, with slopes computed from neighboring control values.

Learning the distribution. We train a conditional diffusion model with EDM preconditioning (Karras et al., 2022). The model is conditioned on the collision rate, temperature parameter, and complete initial density and Hermite coefficient fields. It denoises coefficient values at the 32 future control times; the initial field remains fixed during sampling. We normalize coefficient groups using the training set and whiten their future temporal coordinates conditional on the initial values. The denoiser has four residual space–time convolutional blocks of width 256. We use AdamW with learning rate 2×10^{-4}, zero weight decay, and batch size four, and retain the fixed checkpoint after $40{,}000$ updates. The loss weights the density- and Hermite-coefficient denoising terms equally.

Training noise levels satisfy $\log\sigma \sim \mathcal{N}(-0.6, 1.8^2)$, clipped to $\sigma \in [0.002, 80]$, with EDM data scale one and gradient-norm clipping at one. At generation, we sample the same initial-distribution law used for the numerical data and project its initial field into the Fourier–Hermite representation. We generate the future coefficients with 64 Heun steps, using the EDM noise schedule with exponent seven and no added sampling noise, then invert the normalization and construct the interpolation slopes. The source is derived afterwards from the generated field; it is not a conditioning input to the diffusion model.

Problem construction. We obtain the density from the generated distribution and solve the periodic Poisson equation in Fourier space for the zero-mean electric field. We define the source by substituting that same distribution into Equation (11):

$$S = \partial_t \widetilde{F} + \mathbf{v}\cdot\nabla_{\mathbf{x}}\widetilde{F} + \widetilde{\mathbf{E}}\cdot\nabla_{\mathbf{v}}\widetilde{F} - \nu_{\mathrm{c}}\nabla_{\mathbf{v}}\cdot\big(\mathbf{v}\widetilde{F} + T_{\mathrm{v}}\nabla_{\mathbf{v}}\widetilde{F}\big). \tag{15}$$

Initial data are the reference distribution at $t = 0$, and spatial periodicity is built into the representation. The resulting problem and its reference therefore come from one continuous distribution. The Verifier computes VPFP reference moments from the Hermite representation over $\mathbb{R}^2$.

B.2 Guided generation

The ideal guidance step in Theorem 1 samples from the loss-tilted distribution in Equation (6), which gives more weight to problems on which the current program has larger errors. For the rectified-flow Generator in Appendix B.1.1, we approximate this step by adapting the return guidance of Janner et al. (2022). Their method learns a return predictor and uses its gradient to guide sampling; here we predict program error and add its gradient to the flow velocity.

Given a program and a set of unguided, unperturbed problems, we compute its error on each problem as the mean of the two velocity components' relative ℓ_2 errors, as in Equation (9). For feedback problem i, let z_i be its whitened PCA vector. We standardize the errors by subtracting their mean and dividing by their standard deviation across the feedback problems, and denote the resulting target by e_i. We train a scalar predictor $h(z_s, s)$ by minimizing

$$\mathbb{E}_{i,\epsilon,s}\Big[\big(h(z_s,s) - e_i\big)^2\Big], \qquad z_s = (1-s)\epsilon + s z_i.$$

Here i is sampled uniformly from the feedback problems, independently of the Gaussian noise ϵ and uniform flow time s in Equation (14). This gives a differentiable prediction of standardized program error at intermediate flow states.

During generation, we evaluate the following velocity at the current whitened PCA vector z and flow time s:

$$V_{\mathrm{guided}}(z,s) = V(z,s) + \gamma(z,s)\nabla_z h(z,s).$$

The nonnegative factor $\gamma(z,s)$ controls the norm of the added guidance relative to the original flow velocity $V(z,s)$; Table 4 gives the target ratio and its upper bound. The gradient is taken with respect to the whitened PCA coordinates and points toward increasing predicted error. We convert the final PCA vector back to a continuous reference solution as in Appendix B.1.1, then use MMS (Equation (2)) to construct the matching problem inputs.

NS. For the NS studies in Section 4.2, we fit the predictor for the program entering each round using its errors on 64 unguided, unperturbed problems. Before round one, we evaluate the initial program on a new feedback set; later rounds reuse the unguided problems from the preceding selection set. Both agents share this fixed predictor during the round. For our CodePDE, AutoNumerics, and PDE-SHARP adaptations (Section 4.3), GAV-Full uses a common predictor fitted before development and held fixed.

VPFP. Before each round, we fit a new guidance model using the current program's errors on the first 80 unguided feedback problems in their recorded order. We split these problems into 64 fitting and 16 validation problems before adding diffusion noise. The predictor has two residual space–time convolutional blocks of width 32 and takes the noisy future coefficients, noise level, physical parameters, and complete initial field as inputs. We train it for 2,000 AdamW updates with learning rate 2×10^{-4}, weight decay 0.01, batch size four, and gradient-norm limit one. Validation

is diagnostic; we use the final checkpoint and share it between both agents for the round. The base Generator remains fixed.

For a valid feedback error e_i, define the bounded target $r_i = \log(1 + e_i)/(1 + \log(1 + e_i))$ and standardize it using the valid fitting targets' mean and population standard deviation to obtain $\widehat{r}_i$. The scalar predictor h is trained by squared-error regression of $\exp(h)$ onto weights proportional to $\exp(0.5\widehat{r}_i)$, normalized to mean one over the 64 fitting problems. Invalid evaluations have regression target zero; an auxiliary validity-classification loss has weight 0.1 and does not enter the sampling score. For noisy future coefficients z_σ, let c denote the physical parameters and fixed initial field. Guided sampling replaces the EDM denoiser by

$$D_{\mathrm{G}}(z_\sigma, \sigma, c) = D(z_\sigma, \sigma, c) + \lambda_\sigma \sigma^2 \operatorname{clip}_1(\nabla_{z_\sigma} h(z_\sigma, \sigma, c)),$$

where clip_1 limits the gradient's Euclidean norm to one. Guidance acts only for $\sigma \in [0.002, 2]$ and leaves the initial field fixed. The gain targets a ratio of 0.1 between the RMS guidance drift and the base drift $(z_\sigma - D)/\sigma$, capped at 0.25, with $\lambda_\sigma \leqslant 10^6$. Numerically vanishing gradients or base drift disable the update. We retain the base model's 64-step Heun sampler. This implements a learned, rescaled guidance rule; it does not evaluate the ideal loss-tilted distribution exactly.

B.3 Perturbation

Perturbation lets the Agent explore variations beyond the Generator's learned problem distribution. We add a small perturbation to a generated continuous reference solution, choosing its form to preserve the constraints required by the PDE. We then use MMS (Equation (2)) to construct matching problem inputs, so the perturbed solution is the exact reference for the new problem.

NS. The Generator in Appendix B.1.1 uses cosine mode indices $0, \ldots, 16$ along each spatial axis. We retain these coefficients and add a streamfunction perturbation $\delta\psi$ using modes with nonnegative integer indices k_x, k_y satisfying $17 \leqslant \max(k_x, k_y) \leqslant 20$. The streamfunction representation gives the corresponding velocity change:

$$\psi \longmapsto \psi + \delta\psi, \qquad \widetilde{\mathbf{u}} \longmapsto \widetilde{\mathbf{u}} + (\partial_y \delta\psi, -\partial_x \delta\psi).$$

Pressure is unchanged, and the added spatial coefficients are drawn independently from a Gaussian distribution. The perturbation's time dependence is proportional to $\left[(t - t_0)/(t_1 - t_0)\right]^2$ on the problem's physical time interval $[t_0, t_1]$. We normalize the space–time coefficient vector of $\delta\psi$ to unit ℓ_2 norm, then scale it to the amplitude in Table 4.

The streamfunction construction preserves incompressibility, and the quadratic time factor preserves the initial velocity and its first time derivative. For the numerical domain, the same geometry mask preserves the no-slip boundaries. For PIV, we recompute the boundary velocities from the perturbed reference solution. Appendix D.3 illustrates how perturbation feedback informs revisions to an NS solver.

VPFP. We embed an unguided or guided reference distribution into an expanded representation, increasing the density Fourier order from six to eight, the spatial order of the Hermite coefficient fields from four to six, and the velocity Hermite total degree from 14 to 18. We add six new density Fourier modes and twelve kinetic coefficient directions outside the original representation. Their time controls and first derivatives vanish at $t = 0$, preserving the initial distribution. The bounded density and normalized squared-Hermite construction retain positivity, spatial mean density one, and conditional velocity normalization. We then recompute the electric field and source from the perturbed distribution.

Kinetic time controls sample normalized $t^2 \sin(\pi k t + \varphi)$, with k uniform on $\{1, \ldots, 5\}$ and φ uniform on $[-\pi, \pi]$. For each density mode's sine–cosine coefficient pair, we use a normalized cubic path $t^2 R_\varphi(1 + at, bt)$, where R_φ is a planar rotation, a is uniform on $[-0.45, 0.45]$, and b has a random sign and magnitude uniform on $[0.15, 0.75]$. This pair stays nonzero for $t > 0$. The density and kinetic direction scales are $0.025/\sqrt{6}$ and $0.055/\sqrt{12}$, respectively. Starting at amplitude one, we halve the amplitude at most nine times and accept only a perturbation satisfying the checks in Table 3; a failed direction is rejected, with at most four direction attempts.

The checks use 256 uniformly sampled space–time probes and 512 phase-space probes, with velocity coordinates sampled from $\mathcal{N}(0, T_{\mathrm{v}} I)$. Macroscopic displacement is the joint relative ℓ_2 change

in $(\rho, \mathbf{m}, K)$, normalized by the parent fields. Source displacement is normalized by the larger of the parent and child source norms. Coefficient RMS quantities are square roots of the mean squared Euclidean coefficient norm over probes; the time-derivative change is additionally multiplied by the interval length. These are acceptance checks at finite probe sets, not global error bounds.

Table 3: **VPFP perturbation acceptance settings.** All listed checks must pass for an accepted child problem. Initial-state checks use the same spatial probes at $t = 0$.

Quantity	Accepted range or bound
Joint relative macroscopic change	$[0.002, 0.20]$
Hermite-coefficient RMS change	$[0.005, 0.35]$
Relative source change	$[10^{-4}, 0.80]$
Mean squared norm of added Hermite coefficients	$\geqslant 10^{-5}$
RMS change in coefficient time derivative	$\leqslant 4$
Initial relative macroscopic change	$\leqslant 2 \times 10^{-6}$
Initial Hermite-coefficient RMS change	$\leqslant 2 \times 10^{-6}$

Table 4: **NS guidance and perturbation settings.** The numerical and PIV studies share these settings. Here $[t_0, t_1]$ is the physical time interval.

Parameter	Value
Error predictor h	
Residual blocks	6
Hidden width	512
Embedding dimension for flow time s	128
Training steps	5,000
Batch size	256
Optimizer	AdamW
Learning rate	2×10^{-4}
Weight decay	10^{-4}
Gradient clipping threshold (ℓ_2)	1
Guided sampling	
Target guidance-to-flow norm ratio	0.1
Maximum guidance-to-flow norm ratio	0.25
Streamfunction perturbation	
Spatial cosine mode indices	$17 \leqslant \max(k_x, k_y) \leqslant 20$
Physical time factor	$\left[(t - t_0)/(t_1 - t_0)\right]^2$
Norm of added coefficients (ℓ_2)	2.862×10^{-6}

C Experimental details

C.1 Agent settings and prompts

We use the same multi-turn setting for the NS and VPFP studies (Figure 2). Both agents use `gpt-5.6-sol` with maximum reasoning effort. At the start of each round, they receive the same current program and cumulative memory log. One agent explores alternative approaches, while the other refines the current approach; both are free to change methods in response to numerical feedback. The selected program, both agents' research notes, and the evaluation results pass to the next round. Program selection is described in Appendix C.2. For the two NS studies and the VPFP study in Section 4.2, we use five rounds with a four-hour budget per agent per round, including tool calls and waiting.

The prompt templates below separate the shared task instructions from the two roles. Bracketed fields specify the study's equation and task, public inputs, required outputs, evaluation criterion, and development budget.

> **Shared task prompt (template).** Develop a program for [PDE and task] that maps [public inputs] to [required outputs]. Improve its performance under [evaluation criterion] within [development budget]. Use the current program and memory log as context. Choose your methods and experiments using the problems and tools available in this environment. Use numerical feedback to assess your changes. Before the round ends, submit your strongest program and leave concise research notes describing the methods tried, observations, and useful next steps.
>
> **Exploration role.** Focus primarily on exploring new approaches to the task. Pursue the ideas you judge most promising for producing a stronger program. Use the current program and available evidence as context, and follow useful findings wherever they lead. You have full freedom to choose methods, representations, and experiments, including combining new ideas with improvements to existing approaches.
>
> **Refinement role.** Focus primarily on improving the current approach. Use the current program and available evidence to identify where further work could produce a stronger program. You decide what to retain, change, or replace, and follow useful findings wherever they lead. You have full freedom to choose methods, representations, and experiments, including a different approach when it offers a better path forward.

VPFP initial program and resources. The VPFP experiment starts from a fixed, previously calibrated Hermite-moment program. It projects the public initial distribution and source onto velocity Hermite polynomials of total degree at most six, uses Fourier spatial derivatives, and advances the moment system with fourth-order Runge–Kutta integration. Its existing empirical source and moment corrections are retained. It samples 13 times, a 33×33 spatial grid, and a 9×9 velocity grid over $[-5, 5]^2$, then returns macroscopic values by linear time interpolation and periodic bilinear spatial interpolation. These grids belong to the initial program; subsequent programs choose their own discretizations.

The VPFP run uses seed 101. Each agent may acquire at most 32 new development problems per round, including four initial unguided problems and all perturbed children. Each agent explicitly nominates one program for selection. Program preparation has a $180\,\mathrm{s}$ budget per problem: solver computation, numerical input-query execution, and data transfer count, while shared queue and lock waiting do not. A separate $900\,\mathrm{s}$ execution limit applies to each problem. The campaign uses allocations with three NVIDIA L40S GPUs, 24 CPU cores, and $128\,\mathrm{GB}$ of host memory.

C.2 Program selection and final evaluation

Program selection. At the end of each round in the setting of Appendix C.1, we compare the current program and the two agents' candidates on the same selection problems, using the same evaluation points and execution budget per problem. We retain the program with the lowest mean error among those that complete the evaluation with finite errors. If neither candidate improves on the current program, we retain it for the next round.

Final evaluation. After the last development round, we evaluate the retained program on a separate test set. Within each study, all programs use the same test problems, evaluation points, and execution budget per problem, with the task-specific error measure defined in Section 4.1. For Figure 3, we apply this test evaluation to the saved program from each round.

NS evaluation. For generated NS problems, evaluation times are sampled uniformly over the problem's time interval and spatial locations uniformly over the fluid domain. The NS studies use 96 evaluation points per selection problem and 128 per test problem, shared by the programs being compared. The per-problem execution limit is $900\,\mathrm{s}$ for the two NS studies in Section 4.2 and $300\,\mathrm{s}$ for the comparisons in Sections 4.3 and 4.4. Evaluation against discrete solution data uses the available fluid locations and, for PIV, the measurement times.

VPFP evaluation. Evaluation points are sampled uniformly over the full space–time domain, with both momentum components evaluated at the same points. We use 128 points per development prob-

lem and 256 per selection or test problem. The program prepares its solution before the evaluation coordinates are revealed. The relative ℓ_2 errors use a floor of 10^{-30} on the squared reference norm. Guided selection problems target the program entering each round. The test set is fixed before development, with guided problems targeting the initial program.

Table 5 summarizes the problem sets used for initial development, selection, and testing.

Table 5: Problem sets for initial development, program selection, and evaluation. Counts are per dataset. Unguided problems are sampled directly from the learned Generator. NS perturbations use unguided parents; the VPFP rows distinguish unguided and guided parents. The NS and VPFP selection sets in Section 4.2 are refreshed each round; their test sets are fixed. The other listed NS sets are fixed.

Study	Use	Size	Composition
NS studies (Section 4.2)	Selection per round	192	64 unguided, 64 guided, 64 perturbed
	Test	768	256 unguided, 256 guided, 256 perturbed
Existing PDE agents (Section 4.3)	Initial development	64	Generated NS problems shared across environments
NS comparisons (Sections 4.3 and 4.4)	Selection	24	8 unguided, 8 guided, 8 perturbed
	Test	96	32 unguided, 32 guided, 32 perturbed
Interpolated NS (Appendix D.1)	Test	96	Interpolated numerical solutions
VPFP (Section 4.2)	Selection per round	256	86 unguided, 85 guided, 85 perturbed unguided
	Test	768	256 unguided, 256 guided, 128 perturbed unguided, 128 perturbed guided

C.3 Adapting existing PDE agents

GAV provides the experimentation environment for the three methods in Section 4.3 to develop NS solvers. The agents use this environment to request new problems, query initial velocity and forcing, and obtain numerical feedback during their existing development stages. Our NS reimplementations preserve each method's procedure for proposing, revising, and selecting programs, with the same workflow settings across its three environment variants. All use `gpt-5.6-sol` with maximum reasoning effort. Each method completes its prescribed stages without a fixed wall-clock development limit; per-problem execution limits are given in Appendix C.2.

Original, GAV, and GAV-Full start from the same fixed collection of 64 NS problems sampled from the learned Generator in Appendix B.1.1. Each problem provides initial velocity on a 64×64 cell-centered grid, together with forcing and reference velocities on that grid at 32 uniformly spaced times in $[0, 0.12]$. Original uses this collection throughout development; the GAV variants additionally let agents request new problems and query initial velocity and forcing at chosen coordinates. GAV-Full also provides guided generation and perturbation. Only the Verifier has access to the reference solutions of newly generated problems. It evaluates submitted programs against these solutions and returns numerical error feedback to the agents.

CodePDE. CodePDE (Li et al., 2025) independently generates solver candidates, repairs execution failures, and refines programs using successful implementations and their measured errors. Our NS implementation follows the authors' released refinement code. We use three initial candidate branches followed by three refinement branches, each receiving the same fixed pool of valid initial candidates and their evaluation results. In Original, generation, debugging, and refinement use the fixed data collection and candidate evaluation results. With GAV, these same stages can request

further problems, query inputs at chosen coordinates, and evaluate revised programs before submission.

AutoNumerics. AutoNumerics (Du et al., 2026) uses specialized agents to formulate a PDE, propose and select numerical plans, and implement and repair the chosen plans. Our implementation follows the authors' released planning and execution workflow. It proposes ten numerical plans, selects up to five, and tests each implementation on coarse and fine grids. An agent selects the final solver from these execution reports, prioritizing accuracy and stability, and a final reasoning stage explains the choice. In Original, coding and repair use the fixed sampled inputs. With GAV, these stages can acquire further problems and query inputs at the coordinates required by the chosen spatial and temporal discretization.

PDE-SHARP. PDE-SHARP (Fazliani & Udell, 2025) begins with mathematical analysis and independent solver candidates. Judge agents nominate candidates, receive numerical feedback, and revise or combine their approaches. Our NS implementation follows the paper's algorithm and appendix prompts, with four initial candidates, three judges, and two rounds of refinement. In Original, candidate generation and refinement use the fixed problem collection and shared evaluation feedback. With GAV, both stages can request additional problems and conduct numerical experiments before nominating a program, allowing the judges to use these observations when deciding what to retain or change.

The common selection set in Table 5 supplies numerical feedback for each method's selection procedure. CodePDE and PDE-SHARP retain the valid candidate with the lowest mean error; AutoNumerics uses its final selection agent. We then evaluate all selected programs on the common test set.

Agent active time. We record total agent active time across all development stages. Table 6 reports these times for the runs in Section 4.3. For all three methods, both GAV and GAV-Full achieve lower test error (Figure 5) with less total agent active time than Original.

Table 6: Total agent active time (hours).

Method	Original	GAV	GAV-Full
CodePDE	11.77	5.59	8.52
AutoNumerics	13.22	9.71	8.61
PDE-SHARP	20.80	13.44	12.36

C.4 Ablation study details

We describe the implementations of the ablation conditions in Section 4.4. In every condition, agents can request newly generated problems throughout development. The studies use the same development strategy as the main experiments (Appendix C.1), with three development rounds and a two-hour budget per agent per round. GAV is the reference condition: its learned Generator samples continuous reference solutions (Appendix B.1.1), MMS constructs their matching problem inputs, and agents can query these inputs at arbitrary coordinates. Program selection follows Appendix C.2; the problem sets are listed in Table 5.

Without learning from data. To examine the effect of a mismatch with the target data distribution, we replace GAV's learned Generator with a Gaussian-process prior that is not fitted to the data. We construct continuous reference solutions using the spatial cosine and temporal Chebyshev bases in Appendix B.1.1, drawing independent Gaussian coefficients for the streamfunction and pressure. We apply the same boundary mask to the streamfunction, enforcing zero velocity on the outer walls and obstacle surfaces. MMS then constructs each problem's initial velocity and forcing from the sampled reference solution. Agents can query these inputs at chosen coordinates.

Without MMS construction. To test the role of MMS, we use an *input-first* procedure while retaining GAV's learned problem distribution. We obtain each problem's initial velocity and forcing

from the same learned Generator and retain continuous input queries. We then solve each problem with the incompressible Navier–Stokes solver in Nektar++ (Moxey et al., 2020) to obtain the Verifier's reference solution for development feedback. This takes additional computation time and yields an approximate reference subject to spatial and temporal discretization error.

Without continuous input queries. This condition tests the value of querying problem inputs beyond a fixed grid. Problem generation and program development follow the same procedure as in GAV. For each problem, the initial velocity and forcing available to agents are restricted to values sampled on a fixed 64×64 grid. The restriction applies during development; program selection and final evaluation use the common continuous input interface.

D ADDITIONAL RESULTS AND CASE STUDIES

D.1 EVALUATION ON INTERPOLATED PROBLEMS

Numerical NS problems. To check for overfitting to the learned Generator's problem distribution, we evaluate the final numerical NS programs from the main experiments on a separate set of 96 interpolated problems. We test each program exactly as it was at the end of the main experiments.

We start from 192 numerical solutions unused during agent development and pair similar solutions within the same flow scenario. We fit continuous streamfunctions and pressure fields to the sampled solution data so that the derivatives required by MMS can be evaluated. We interpolate each pair by taking a weighted combination of these fields to obtain a new reference solution. MMS provides the corresponding forcing. This construction preserves the geometry and continuous representation used in the main experiments but does not sample the learned Generator. This local interpolation keeps the test solutions close to the original data, providing an independent check of accuracy in the same flow regimes.

The GAV-Full program from Section 4.2 achieves a mean relative velocity error of 2.64% on these problems. Both GAV variants improve all three existing PDE agents compared with Original (Table 7).

Table 7: Mean relative velocity error (%) of the final numerical NS programs from the main experiments on 96 interpolated test problems. Lower is better.

Agent setting	Environment	Mean error (%)
Multi-turn setting	GAV-Full	2.6375
CodePDE	Original	3.1231
	GAV	0.1890
	GAV-Full	1.0555
AutoNumerics	Original	3.4970
	GAV	0.3086
	GAV-Full	0.9567
PDE-SHARP	Original	3.5203
	GAV	0.2744
	GAV-Full	0.0738

Table 8 gives the corresponding results for the environment comparison in Section 4.4. The ordering of the four conditions is unchanged: GAV has the lowest error, while the condition without continuous input queries has an error 13.5 times that of GAV. These results indicate that GAV helps agents develop more accurate programs, with gains that transfer to problems constructed independently of the learned Generator.

Table 8: Mean relative velocity error (%) for the environment comparison on 96 interpolated NS test problems. Lower is better.

Environment	Mean error (%)
GAV	**0.440**
w/o learning from data	1.561
w/o MMS construction	0.864
w/o continuous input queries	5.915

VPFP problems. We test the initial VPFP program S0 and the programs retained after rounds 1–5 on eight problems constructed independently of the learned Generator. The programs are unchanged, and these problems are not used for program selection. We sample one problem in each of four collision-rate bins crossed with two temperature bins, spanning $\nu_{\mathrm{c}} \in [10, 30]$ and $T_{\mathrm{v}} \in [0.8, 1.2]$. For each problem, a coarse numerical solve supplies 65 time slices on $[0, 1]$, with 5×5 spatial points

on $[0, 2\pi)^2$ and 9×9 velocity points on $[-6, 6]^2$. We interpolate the square root of the sampled distribution using Gaussian radial basis functions of width 1.2 in velocity, Fourier interpolation in space, and cubic splines in time. Squaring and normalizing this interpolant gives a nonnegative continuous reference with unit spatial mean density. MMS derives a new forcing from this reference, replacing the forcing used in the coarse solve. The errors therefore measure accuracy on the resulting manufactured problems, rather than convergence to the original coarse-solve equations.

Each program is evaluated at the same 256 hidden space–time points per problem. The initial program retains its own 13 time nodes, 33×33 spatial grid, and 9×9 velocity grid on $[-5, 5]^2$ (Appendix C.1); the coarse reference grid does not prescribe its discretization. All six programs complete all eight problems. Table 9 shows that the final program reduces mean macro error from 8.415% for S0 to 0.01162% on this independent problem family. The mean error is not monotone across rounds; R4 has the lowest value on this bank (0.01097%).

Table 9: Mean relative ℓ_2 errors (%) of the frozen VPFP programs on eight independent interpolated problems. Each component column averages per-problem errors; momentum is evaluated as a vector. Mean error is the arithmetic mean of the density, momentum, and kinetic-energy errors. All programs use the same 256 evaluation points per problem. Lower is better.

Program	Density	Momentum	Kinetic energy	Mean error
S0	1.867	15.15	8.229	8.415
R1	0.01088	0.1344	0.07499	0.07344
R2	0.001756	0.02068	0.01252	0.01165
R3	0.001523	0.01929	0.01224	0.01102
R4	0.001484	0.01923	0.01220	0.01097
R5	0.001671	0.02062	0.01258	0.01162

D.2 Problem difficulty under guided generation

We examine whether guided generation supplies harder problems for the current program over five rounds of development on numerical NS data. At each update, we train the guidance model using error feedback from the current round's program. Keeping that program fixed, we compare its mean error on the current round's problems with that on new problems generated using the updated guidance.

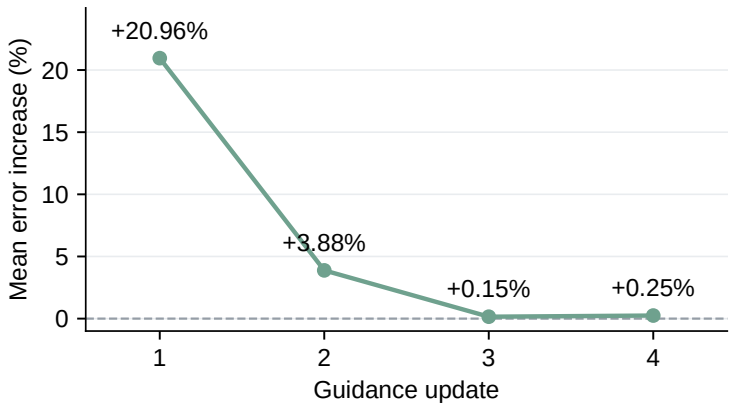

Figure 7: **Percentage increase in mean error after guidance updates.** Each comparison fixes the program and measures the joint relative ℓ_2 velocity error on the old and new problem sets.

The mean error is higher on the new problems at all four updates (Figure 7). These observations show that guided generation can supply more challenging problems for the current program as development proceeds.

D.3 PROGRAM EVOLUTION OVER FIVE ROUNDS

We trace the numerical NS and VPFP experiments with GAV-Full in Section 4.2. Figure 8 shows the submitted and retained programs over five rounds in both studies. The Agents use new problems, input queries, and numerical feedback from GAV to decide which revisions to keep.

Using feedback to isolate a useful change. In round 2, the Explorer tests a proposed boundary correction on additional problems from GAV. The Verifier reports a scalar relative ℓ_2 error score for each velocity component (horizontal and vertical). Using these scores, the Explorer compares parts of the correction on the same problem and retains only the change to diffusion near the walls.

Testing methods on different meshes. The Explorer queries GAV for initial velocities and forcing on both a finite-element mesh and the existing finite-difference grid. The Verifier evaluates the solvers and their combinations on the same problems and points, summarizing errors over those points separately for horizontal and vertical velocity. These two scores lead the Explorer to combine horizontal predictions while keeping the finite-difference prediction for vertical velocity.

Reusing earlier exploration. The shared research history lets the Explorer recover a stabilization method from its unselected round-4 proposal. In round 5, it combines this idea with a rotated mesh and the finite-difference solver, then compares the new program with the current one on fresh problems from GAV. The Verifier reports lower errors for both velocity components, and the combination becomes the final retained program. The dashed arrow in Figure 8a traces how an earlier branch contributes to this revision.

Learning from perturbation feedback. In the final round, the Refiner uses a problem and its perturbed counterpart from GAV to check a proposed change to the horizontal velocity prediction. The Verifier shows that a gain on the original problem reverses on its counterpart, prompting the Agent to investigate the vertical component. Further comparisons on a broader set of unguided, guided, and perturbed problems support changing that component. The submitted program consequently preserves the previous horizontal calculation and revises the vertical one.

Program development on VPFP problems. For VPFP, the programs in Figure 8b predict density, momentum, and kinetic-energy density by evolving velocity Hermite moments.

Diagnosing errors through input queries. In round 2, the Refiner queries the public source on successively finer velocity grids, using GAV's ability to evaluate inputs at chosen coordinates. The comparison exposes inaccurate source integration on a perturbed problem and motivates denser velocity sampling. The Refiner then tests the revised program against the current program on shared development problems; lower errors from the Verifier support retaining the revision.

Choosing model complexity from feedback. In round 3, the Explorer compares the current model with a larger system of Hermite moments on the same problems from GAV, including guided problems. The Verifier returns lower errors for the larger system on problems with lower collision rates, while a guided problem with a higher rate favors the current model. The Agent therefore makes the number of moments depend on the collision rate, adding complexity where the development comparisons support it.

Allocating input queries using feedback. In round 5, the Explorer queries the source through GAV at fewer velocity points and more times, using a finer velocity grid for corrections at selected times. Source probes determine the correction times; the total query count stays close to that of the current program. The Verifier compares the revised and current programs on fresh problems, including perturbed counterparts. The errors favor the new allocation at lower collision rates and the previous treatment at higher rates. The submitted program uses the collision rate to choose between them.

(a) Numerical NS

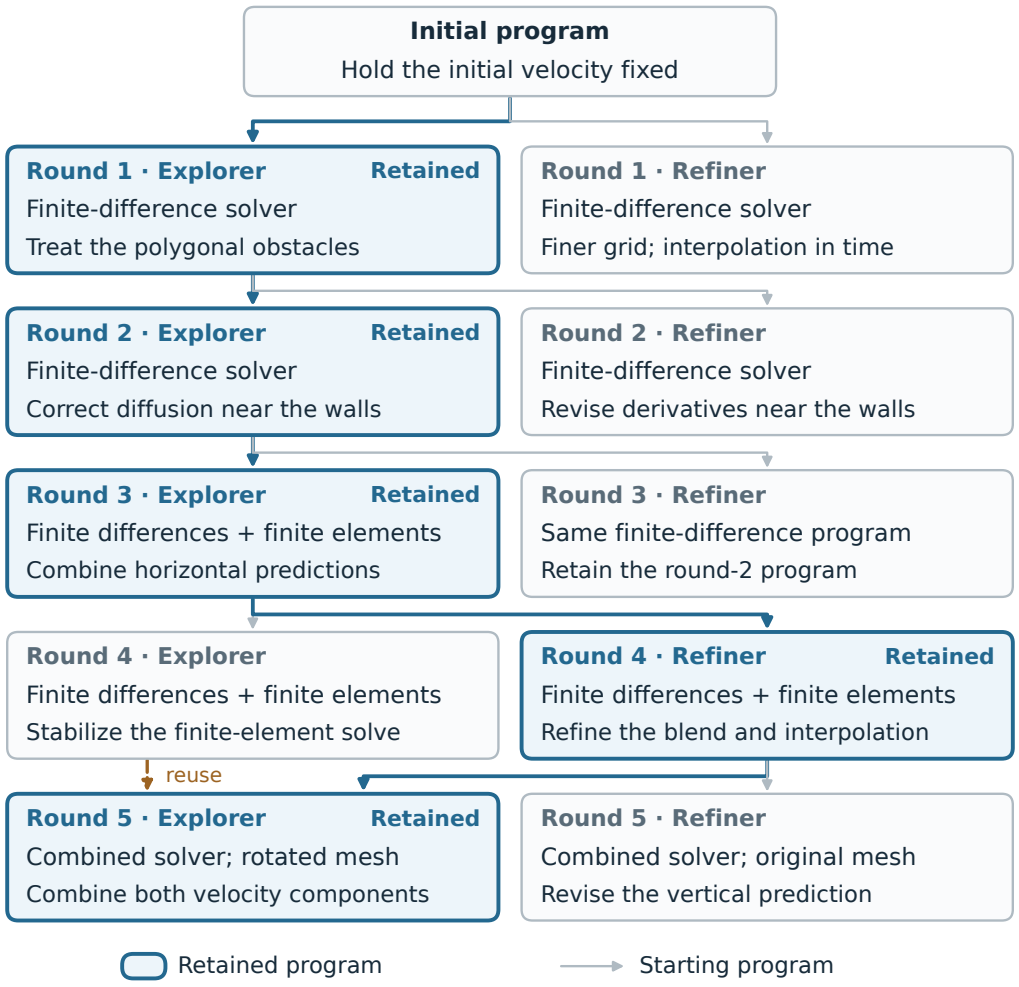

Figure 8: **Program evolution over five rounds.** (a) Numerical NS. Each round shows the Explorer's and Refiner's final submissions. Blue marks retained programs. Solid arrows connect each submission to its starting program; the dashed arrow shows reuse of a method from the shared research history.

(b) VPFP

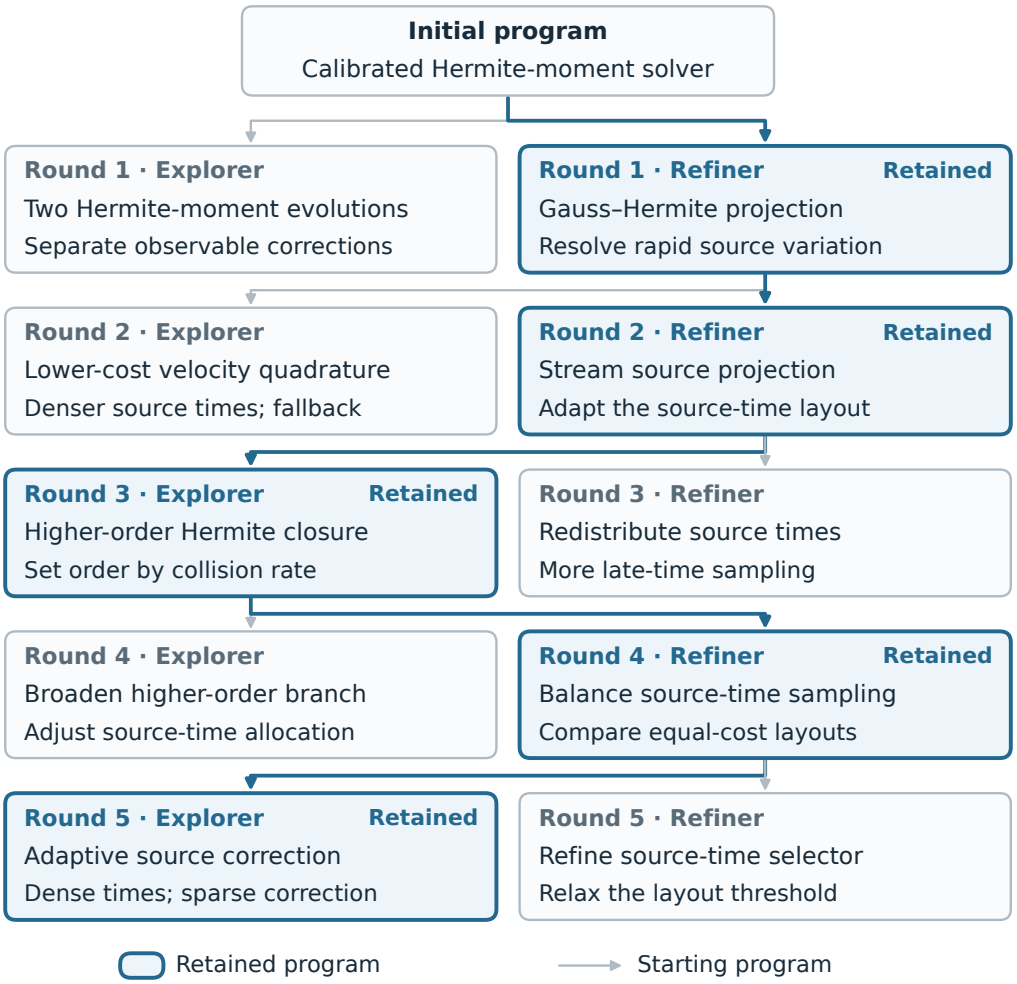

Figure 8: **Program evolution over five rounds (continued).** (b) VPFP. Each branch shows an Agent's final proposal; blue marks the program retained after evaluation on that round's 256 selection problems. Arrows identify the starting program for each branch. Proposals evolve Hermite moments, refining their order and the sampling of the public source in response to numerical feedback. Test errors are reported in Figure 3c.

D.4 SOLUTION FIELDS AND ERROR EVOLUTION

We visualize the first unguided problem in each archived PIV and VPFP test bank using the frozen programs retained after each development round. Both cases were fixed before examining their visualization errors. The continuous manufactured references and saved predictions are shared throughout each comparison; plotting adds no interpolation or smoothing. These single-problem diagnostics illustrate spatial and temporal behavior, while Figure 3 reports performance over the full test banks.

Relative ℓ_2 curves reduce over the spatial grid at each time; full-window values reduce over all saved space–time samples. These diagnostics use unweighted sums without a denominator floor. Zero error against a zero reference is assigned zero; nonzero error against a zero reference is undefined. Positive errors use logarithmic axes.

PIV-derived NS problem. The example covers $[0, 1.55]$ s at 193 times on a 96×64 spatial grid. Its reference solution is sampled from the Generator fitted to the PIV measurements. The rectangular wake region contains no solid obstacle; its outer boundary velocities follow the manufactured reference, and velocity diagnostics include these boundary points. We compute vorticity, $\omega = \partial_x u_2 - \partial_y u_1$, by applying the same finite-difference operator to reference and predicted velocities on complete interior stencils. The vector velocity diagnostic combines both components in one norm, whereas Equation 9 averages their separate relative errors.

Figure 9 retains all 193 times, including brief R5 error peaks near both ends of the window. R2 retains R1 and produces identical predictions. R4 has a smaller full-window vector relative ℓ_2 error than R5 on this problem (0.00141% versus 0.00172%); this single-case ordering need not match the mean over the full test bank. Figures 10 and 11 show the corresponding fields and round-by-round spatial errors.

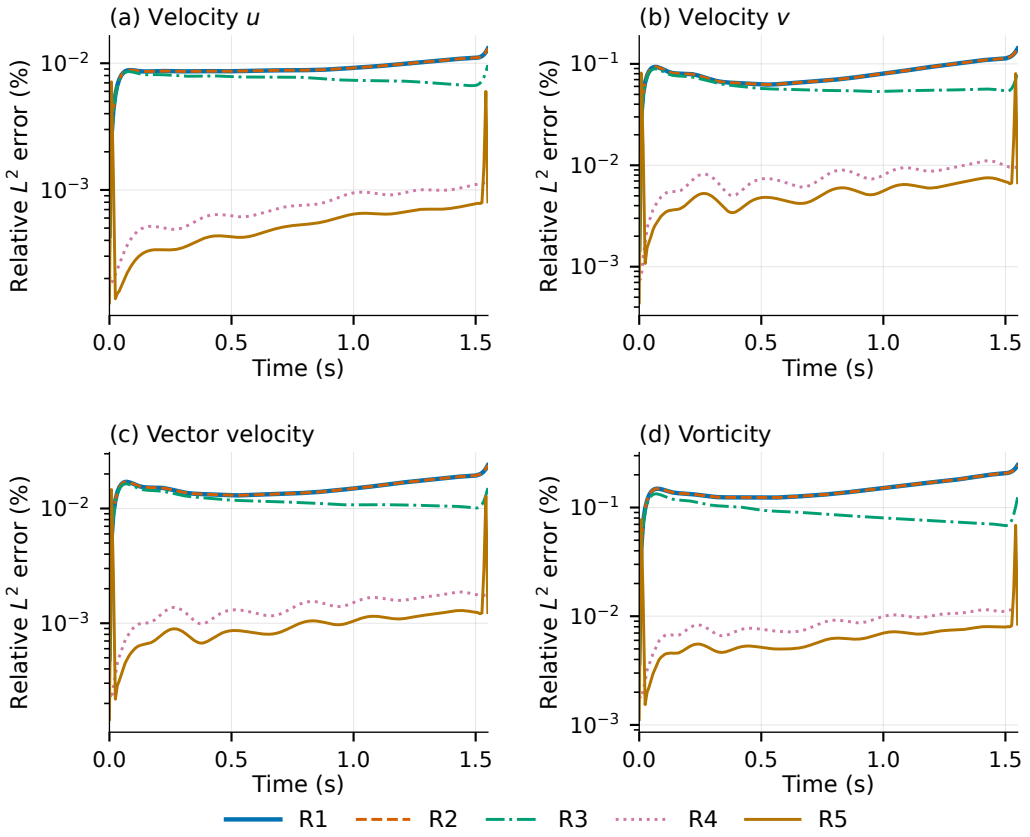

Figure 9: **PIV errors over the complete time window.** Spatial relative ℓ_2 errors (%) for each velocity component, the joint velocity vector, and vorticity at all 193 query times. R2 coincides with R1. These are single-problem grid diagnostics.

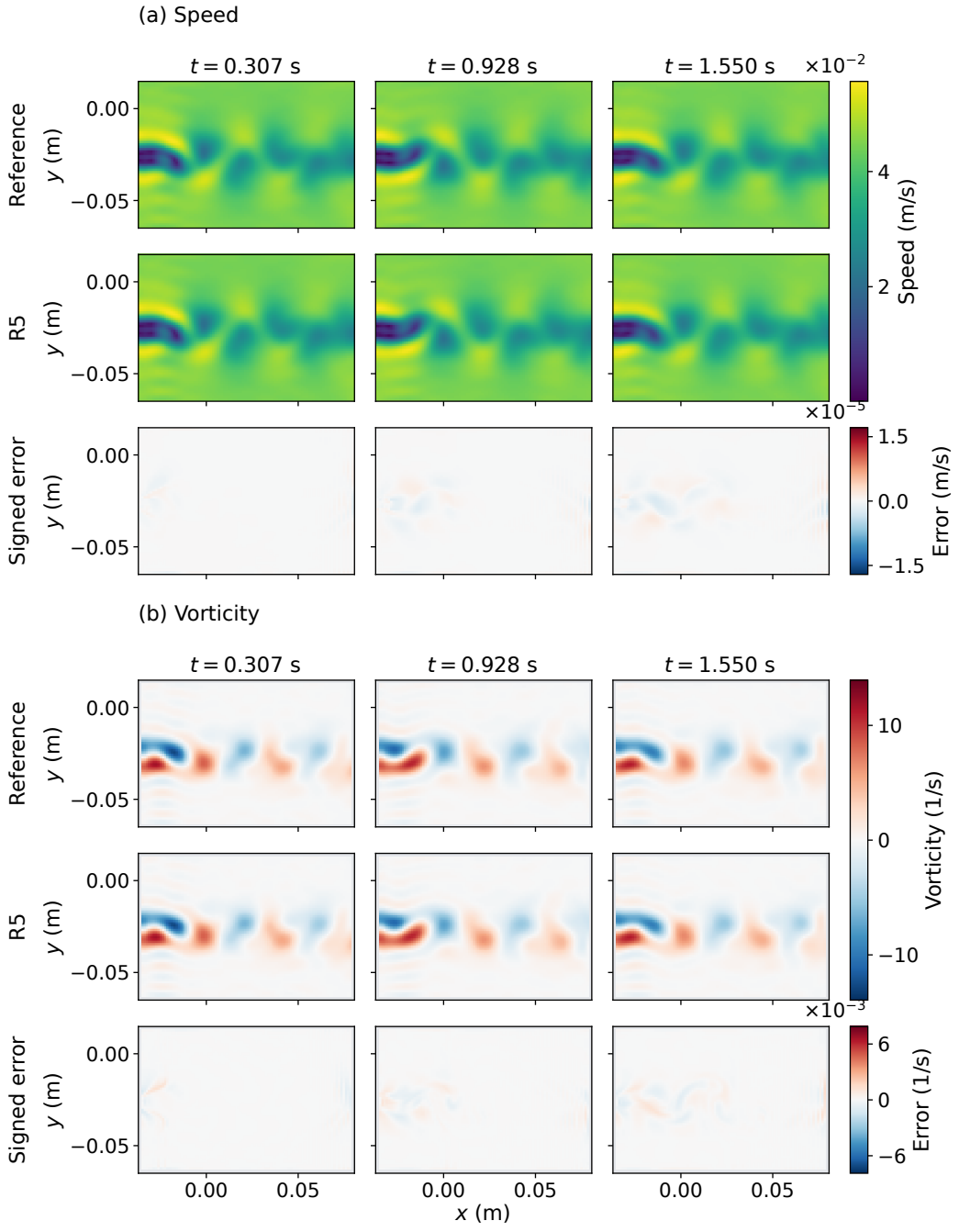

Figure 10: **Final-program fields on the PIV-derived example.** Reference, R5 prediction, and signed error (prediction minus reference) for speed and vorticity at $t = 0.307$, 0.928, and 1.55 s. Each field uses one scale for reference and prediction and a separate symmetric error scale, fixed over the full time window. Gray vorticity pixels lie outside the common interior-stencil support.

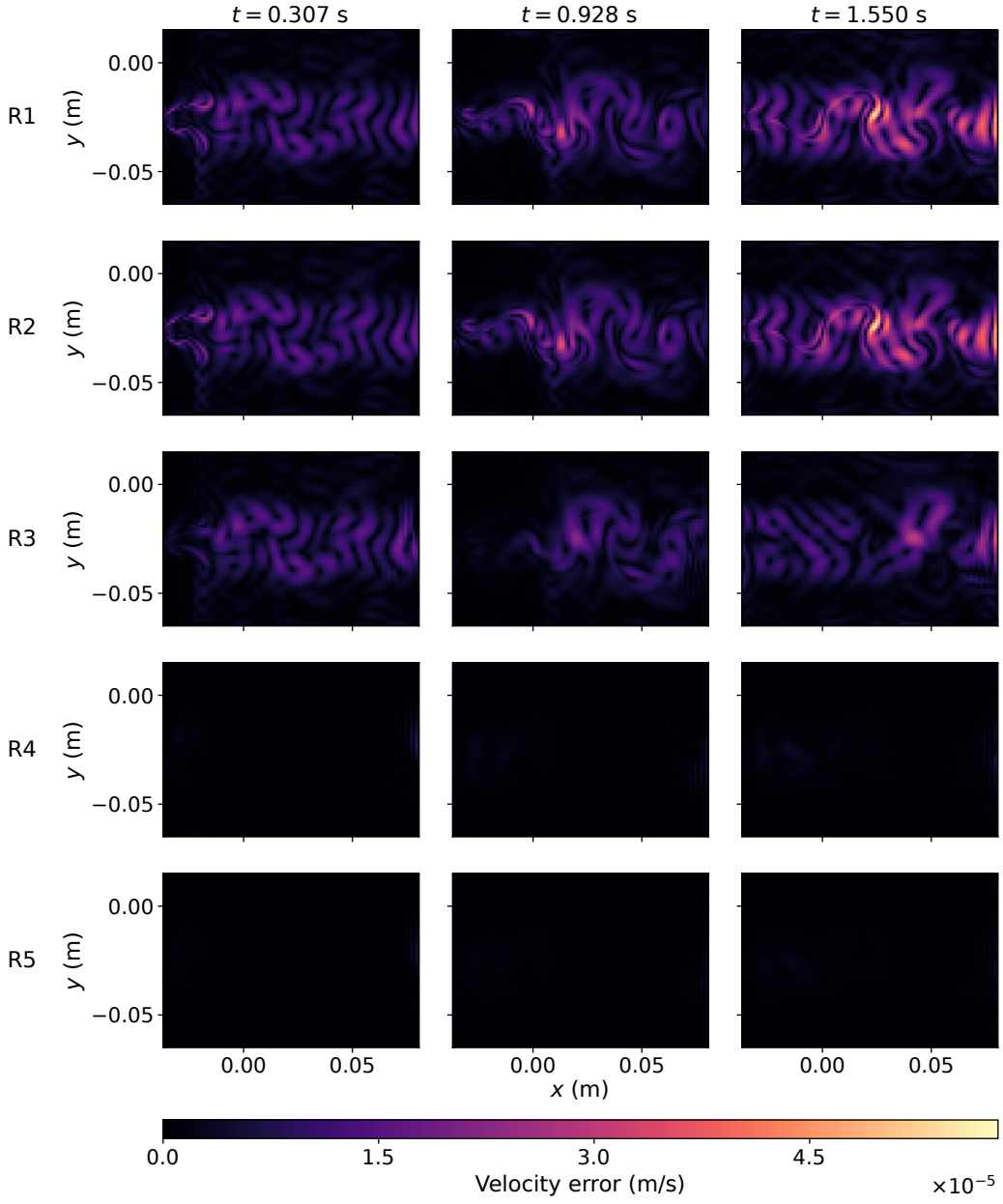

Figure 11: **Velocity error across PIV development rounds.** Pointwise vector error in m/s for the retained programs R1–R5 on the same manufactured problem at three times. All panels share one linear scale determined over all five programs and all 193 times. R2 repeats R1 because its program is unchanged.

VPFP problem. The example uses 101 times in $[0, 1]$ and a periodic 64×64 grid, with $\nu_c = 21.1478$ and $T_v = 0.9558$. We display density, the two momentum components, and kinetic-energy density separately. Their scalar relative errors in Figure 12 differ from the test metric, which averages errors over density, vector momentum, and kinetic-energy density at the test evaluation points. R5 has the smallest full-window error in each displayed scalar field on this problem; R2 and R3 are close but are different programs. This does not imply a uniform ordering at every location or time.

Figures 13–15 use scales fixed over the full time interval to compare field and error evolution. As the momentum amplitude decays, these scales obscure late-time structure. Figure 16 therefore provides a separate view with explicitly labeled final-time scales.

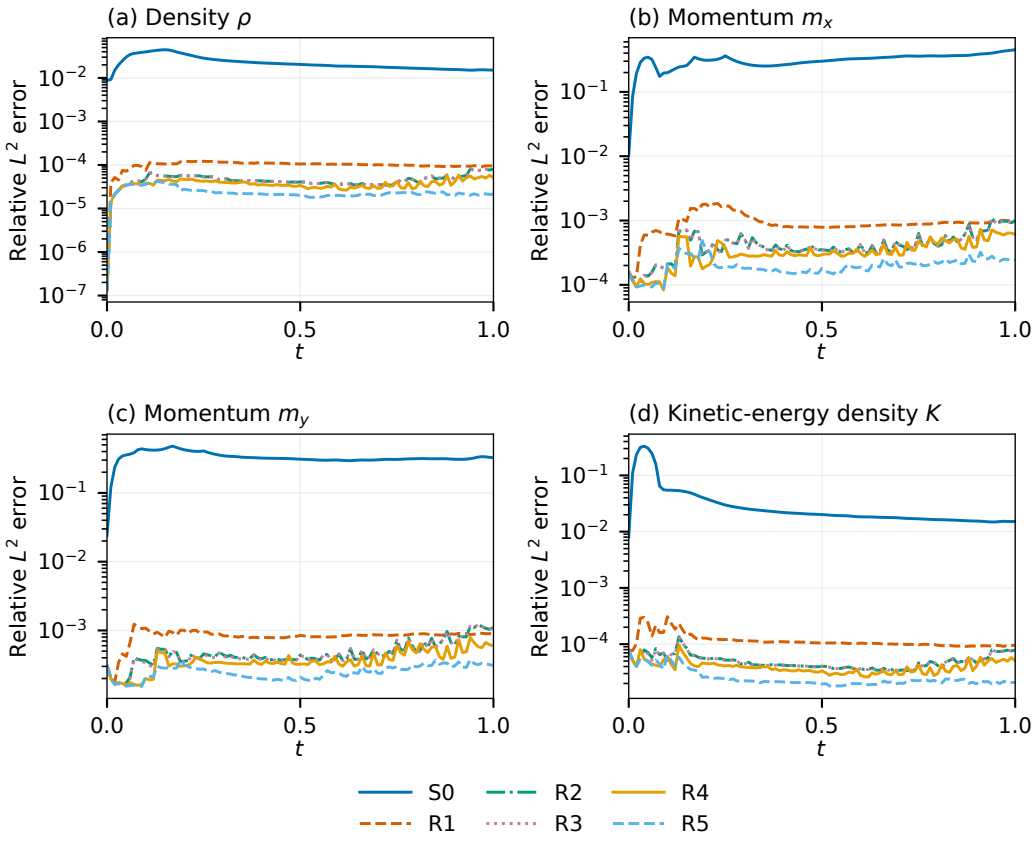

Figure 12: **VPFP errors over the complete time interval.** Spatial relative ℓ_2 errors for density, each momentum component, and kinetic-energy density on the same problem, comparing the initial program S0 and the programs retained after rounds 1–5. Each point uses all 64×64 spatial samples at one of 101 times. Values are fractions on logarithmic axes.

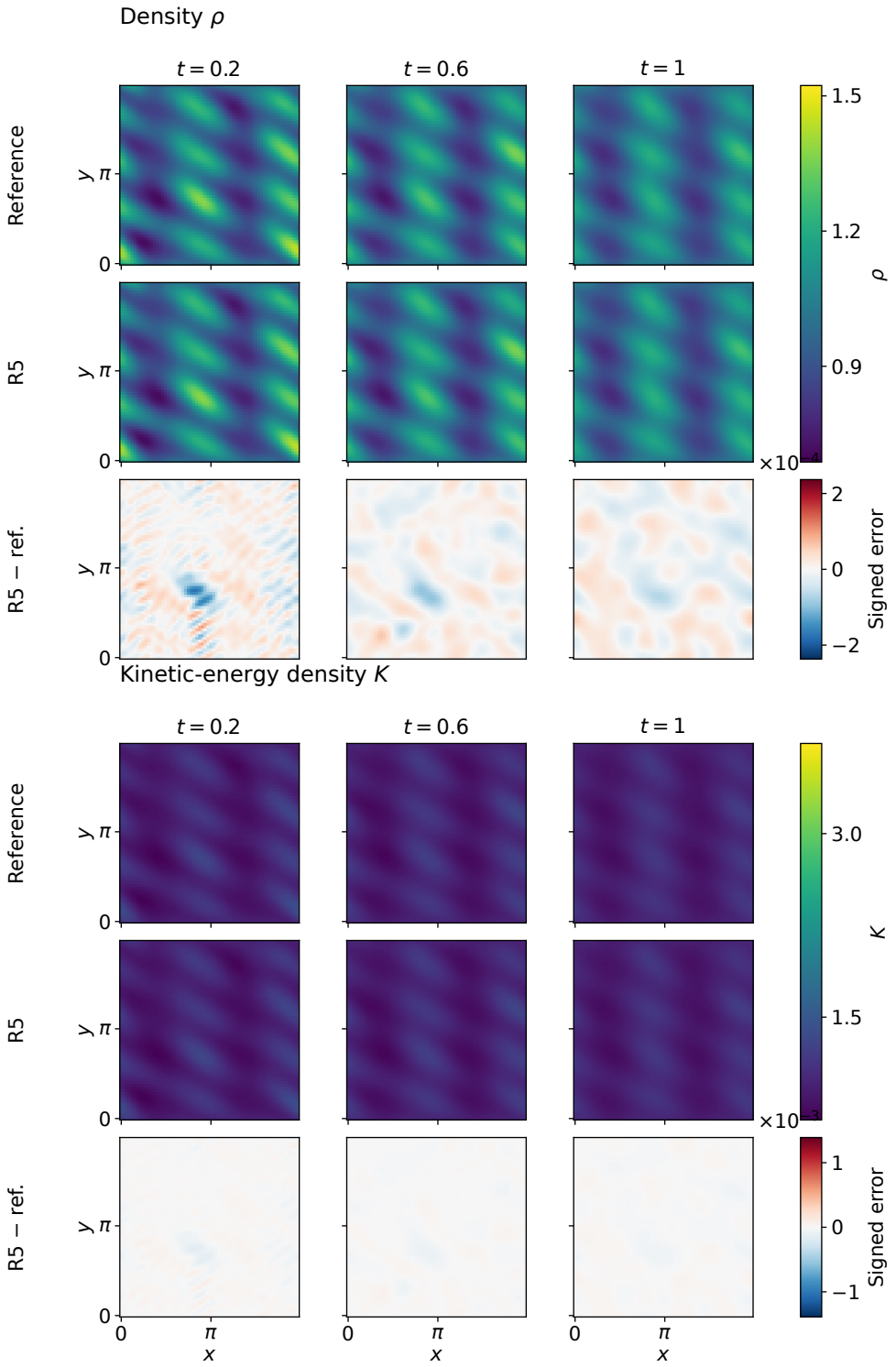

Figure 13: **Density and kinetic-energy density on the VPFP example.** Each field shows the continuous manufactured reference, R5 prediction, and signed error at $t = 0.2$, 0.6, and 1. Reference and prediction share a scale; errors use a separate symmetric scale. Each scale is fixed over all 101 times for its field.

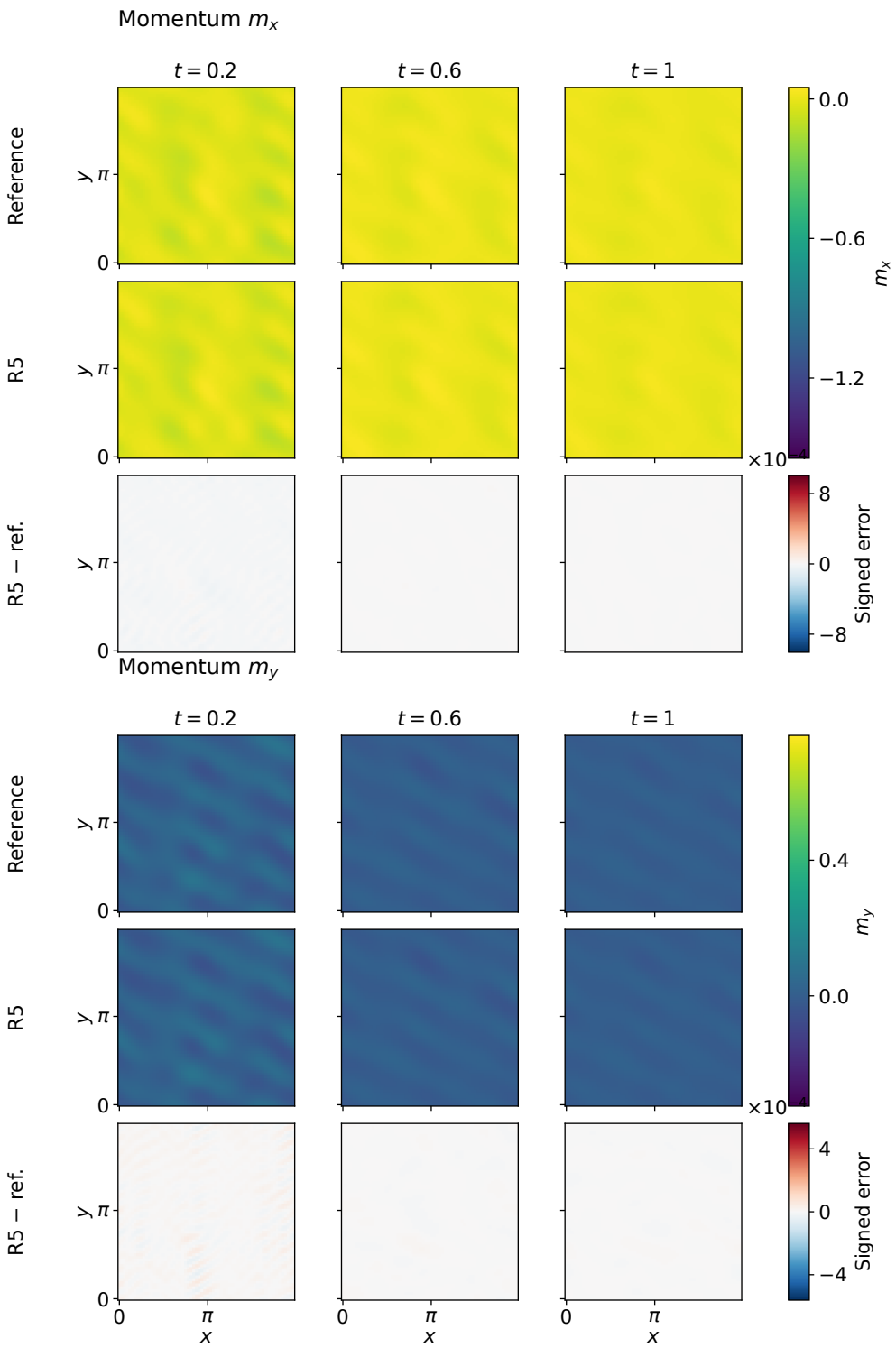

Figure 14: **Momentum components on the VPFP example.** Reference, R5 prediction, and signed error for m_x and m_y at $t = 0.2$, 0.6, and 1. For each component, reference and prediction share a scale fixed over all 101 times, with a separate symmetric error scale.

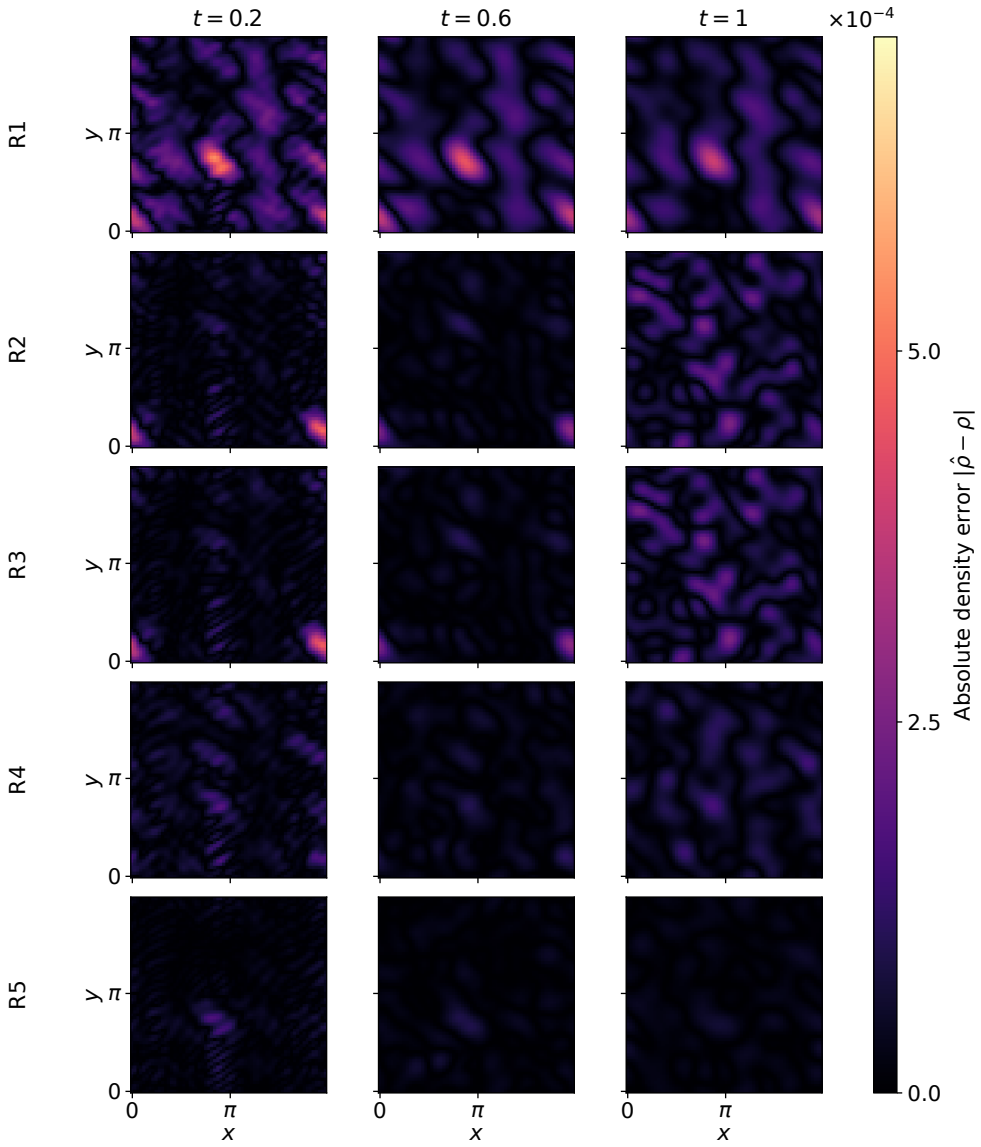

Figure 15: **Density error across VPFP development rounds.** Absolute density error of the retained programs R1–R5 on the same manufactured problem at $t = 0.2$, 0.6, and 1. All panels share one linear scale set by the errors of these five programs over all 101 times.

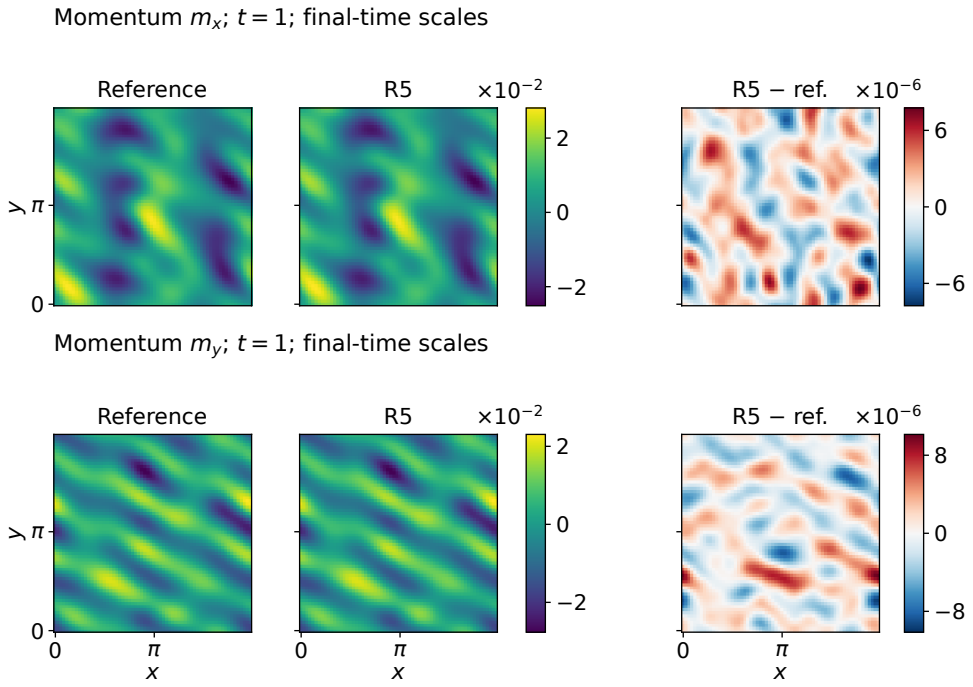

Figure 16: **VPFP momentum at the final time.** Reference, R5 prediction, and signed error for the two momentum components at $t = 1$. Each row uses a common reference/prediction scale and a separate symmetric error scale, both determined only at this time. These scales differ from the full-window scales in Figure 14.

E PROOFS

E.1 PROOF OF PROPOSITION 1

Proof. Fix the learned problem distribution $\widehat{P}$ and a program A chosen without using the test set. All probabilities below are with respect to the M independent test problems from $\widehat{P}$. By Assumption 1(ii), the expected loss under $\widehat{P}$ satisfies $\mu_{\widehat{P}} = \mathbb{E}_{c\sim\widehat{P}}[\ell_A(c)] \in (0,\infty)$. Write $X(c) = \ell_A(c)/\mu_{\widehat{P}} - 1$, so that $\mathbb{E}_{\widehat{P}}X = 0$ and $X \geqslant -1$. We first consider $\nu > 0$ and treat $\nu = 0$ at the end. Throughout the proof, $1/\kappa = \infty$ when $\kappa = 0$.

We first compare μ_P and $\mu_{\widehat{P}}$. By the entropy inequality (Baldasso et al., 2022, Lemma 2.1), for any random variable Z with $\mathbb{E}_P|Z| < \infty$ and $\mathbb{E}_{\widehat{P}}e^Z < \infty$, we have

$$\mathbb{E}_P Z \leqslant D_{\mathrm{KL}}(P\|\widehat{P}) + \log \mathbb{E}_{\widehat{P}}e^Z. \tag{16}$$

We begin by verifying that μ_P is finite. Fix $0 < s < 1/\kappa$. For each integer $n \geqslant 1$, apply (16) to the bounded truncation $Z = s\min\{X+1, n\}$. Since $\min\{X+1,n\} \leqslant X+1$, Assumption 1 gives

$$s\mathbb{E}_P \min\{X+1,n\} \leqslant \varepsilon + \log \mathbb{E}_{\widehat{P}}e^{s(X+1)} \leqslant \varepsilon + s + \frac{\nu^2 s^2}{2}.$$

Since $0 \leqslant \min\{X+1,n\} \uparrow X+1$, monotone convergence gives $\mathbb{E}_P(X+1) \leqslant \varepsilon/s + 1 + \nu^2 s/2 < \infty$. The bound $|X| \leqslant (X+1)+1$ then implies $\mathbb{E}_P|X| < \infty$. Since $\ell_A = \mu_{\widehat{P}}(X+1)$ and $\mu_{\widehat{P}} < \infty$ by Assumption 1(ii), we also have $\mu_P = \mu_{\widehat{P}}\mathbb{E}_P(X+1) < \infty$.

For $0 < s < 1/\kappa$, both sX and $-sX$ are now integrable under P, and (3) bounds their exponential moments under $\widehat{P}$. Applying (16) to these two variables gives

$$-\left(\varepsilon + \frac{\nu^2 s^2}{2}\right) \leqslant s\mathbb{E}_P X \leqslant \varepsilon + \frac{\nu^2 s^2}{2}.$$

Dividing by $s > 0$ and using $\mathbb{E}_P X = \mu_P/\mu_{\widehat{P}} - 1$, we obtain

$$\left|\frac{\mu_P}{\mu_{\widehat{P}}} - 1\right| = |\mathbb{E}_P X| \leqslant \frac{\varepsilon}{s} + \frac{\nu^2 s}{2}, \qquad 0 < s < 1/\kappa.$$

If $\varepsilon = 0$, letting $s \downarrow 0$ gives $\mu_P = \mu_{\widehat{P}}$. If $\varepsilon > 0$, choose

$$s = \frac{\sqrt{2\varepsilon}}{\nu + \kappa\sqrt{2\varepsilon}}.$$

This choice satisfies $0 < s < 1/\kappa$ and $s \leqslant \sqrt{2\varepsilon}/\nu$. Substituting into the preceding bound gives

$$|\mathbb{E}_P X| \leqslant \nu\sqrt{\frac{\varepsilon}{2}} + \kappa\varepsilon + \nu\sqrt{\frac{\varepsilon}{2}} = a_\varepsilon.$$

Thus, in both cases,

$$|\mu_P - \mu_{\widehat{P}}| \leqslant a_\varepsilon \mu_{\widehat{P}}. \tag{17}$$

Next, we bound the deviation of the empirical mean loss $\widehat{\mu}_M$ from the expected loss $\mu_{\widehat{P}}$. Let $\overline{X}_M = M^{-1}\sum_{j=1}^M X(\widetilde{c}_j) = \widehat{\mu}_M/\mu_{\widehat{P}} - 1$. For $|s| < M/\kappa$, we have $|s/M| < 1/\kappa$, so independence of the test problems and (3) give

$$\log \mathbb{E}e^{s\overline{X}_M} = \sum_{j=1}^M \log \mathbb{E}_{\widehat{P}}e^{(s/M)X} \leqslant \frac{\nu^2 s^2}{2M}.$$

Set $x = \log(2/\delta)$. For any $0 < s < M/\kappa$, applying Markov's inequality to $e^{s\overline{X}_M}$ yields

$$\mathbb{P}\left(\overline{X}_M > \frac{x}{s} + \frac{\nu^2 s}{2M}\right) \leqslant e^{-x-\nu^2 s^2/(2M)}\mathbb{E}e^{s\overline{X}_M} \leqslant e^{-x}.$$

Replacing s by $-s$ in the bound for $\log \mathbb{E}e^{s\overline{X}_M}$ gives $\log \mathbb{E}e^{-s\overline{X}_M} \leqslant \nu^2 s^2/(2M)$. Applying Markov's inequality to $e^{-s\overline{X}_M}$ therefore gives the same probability bound for $-\overline{X}_M$. Choose

$$s = \frac{\sqrt{2Mx}}{\nu + \kappa\sqrt{2x/M}}.$$

Then $0 < s < M/\kappa$ and $s \leqslant \sqrt{2Mx}/\nu$, so

$$\frac{x}{s} + \frac{\nu^2 s}{2M} \leqslant \nu\sqrt{\frac{x}{2M}} + \frac{\kappa x}{M} + \nu\sqrt{\frac{x}{2M}} = b_M.$$

Thus the threshold in each tail bound is at most b_M. Taking a union bound, we obtain

$$\mathbb{P}(|\overline{X}_M| > b_M) \leqslant \mathbb{P}(\overline{X}_M > b_M) + \mathbb{P}(-\overline{X}_M > b_M) \leqslant 2e^{-x} = \delta.$$

Equivalently, with probability at least $1 - \delta$,

$$|\widehat{\mu}_M - \mu_{\widehat{P}}| \leqslant b_M \mu_{\widehat{P}}. \tag{18}$$

Combining (17) and (18) by the triangle inequality, we obtain, with probability at least $1 - \delta$,

$$|\widehat{\mu}_M - \mu_P| \leqslant |\widehat{\mu}_M - \mu_{\widehat{P}}| + |\mu_{\widehat{P}} - \mu_P| \leqslant \mu_{\widehat{P}}(a_\varepsilon + b_M).$$

Moreover, (17) and $a_\varepsilon < 1$ give

$$\mu_P \geqslant (1 - a_\varepsilon)\mu_{\widehat{P}} > 0.$$

Dividing the preceding error bound by μ_P yields

$$\frac{|\widehat{\mu}_M - \mu_P|}{\mu_P} \leqslant \frac{a_\varepsilon + b_M}{1 - a_\varepsilon},$$

as claimed.

Finally, suppose $\nu = 0$. For any $0 < s < 1/\kappa$, Jensen's inequality and (3) give

$$1 = e^{s\mathbb{E}_{\widehat{P}}X} \leqslant \mathbb{E}_{\widehat{P}}e^{sX} \leqslant e^0 = 1.$$

Since the exponential function is strictly convex, equality in Jensen's inequality implies that X is constant almost surely under $\widehat{P}$. The identity $\mathbb{E}_{\widehat{P}}X = 0$ shows that this constant is zero. The finite KL divergence gives $P \ll \widehat{P}$, so $X = 0$ also holds almost surely under P. Consequently, $\widehat{\mu}_M = \mu_P = \mu_{\widehat{P}} > 0$ with probability one, which proves the bound in this case and completes the proof. □

E.2 Proof of Theorem 1

The following lemma shows that the guided loss agrees with the robust objective at the current parameter and controls their difference after a parameter change. It supplies the quadratic bound used in the convergence proof.

Lemma 1. *Suppose Assumption 2(ii) holds, and let $\bar{\beta} \geqslant 0$ be finite. For $\rho > 0$, let R_t be the exponential tilt in* (6)*, calibrated to $D_{\mathrm{KL}}(R_t\|\widehat{P}) = \rho$, with $0 < \beta_t \leqslant \bar{\beta}$ for every $t \geqslant 0$. For $\rho = 0$, take $R_t = \widehat{P}$. Then, for every $t \geqslant 0$ and $\theta \in \Theta$,*

$$0 \leqslant F(\theta) - f_t(\theta) \leqslant \frac{\bar{\beta}G^2}{2}\|\theta - \theta_t\|^2. \tag{19}$$

In particular, $f_t(\theta_t) = F(\theta_t)$.

See Appendix E.2.1 for the proof.

Proof. We first verify that the minima in (7) and in the definition of $F^\star$ are attained. For any distribution R with $D_{\mathrm{KL}}(R\|\widehat{P}) \leqslant \rho$, finite KL divergence implies $R \ll \widehat{P}$. Thus the boundedness and Lipschitz conditions in Assumption 2(ii) also hold R-almost surely. The uniform bound on the losses makes F and every f_t finite. Moreover, for any $\theta, \theta' \in \Theta$,

$$\left|\mathbb{E}_R\ell_\theta - \mathbb{E}_R\ell_{\theta'}\right| \leqslant \mathbb{E}_R|\ell_\theta - \ell_{\theta'}| \leqslant G\|\theta - \theta'\|.$$

Taking $R = R_t$ shows that each f_t is G-Lipschitz. The same inequality also gives $\mathbb{E}_R \ell_\theta \leqslant F(\theta') + G\|\theta - \theta'\|$. Taking the supremum over R and then interchanging θ and θ', we obtain

$$|F(\theta) - F(\theta')| \leqslant G\|\theta - \theta'\|.$$

Consequently, F and the objective in (7) are continuous on the nonempty compact set Θ, so their minima are attained. Fix $\theta^\star \in \Theta$ such that $F(\theta^\star) = F^\star$.

We next prove that $F(\theta_{t+1}) \leqslant F(\theta_t)$ for every $t \geqslant 0$. Fix $t \geqslant 0$ and set $C = \bar{\beta} G^2$. By the definition of idealized guidance, $D_{\mathrm{KL}}(R_t \| \widehat{P}) \leqslant \rho$, so $f_t(\theta) \leqslant F(\theta)$ for all $\theta \in \Theta$. For any such θ, we have

$$\begin{aligned} F(\theta_{t+1}) + \frac{\tau - C}{2}\|\theta_{t+1} - \theta_t\|^2 &\leqslant f_t(\theta_{t+1}) + \frac{\tau}{2}\|\theta_{t+1} - \theta_t\|^2 \\ &\leqslant f_t(\theta) + \frac{\tau}{2}\|\theta - \theta_t\|^2 \\ &\leqslant F(\theta) + \frac{\tau}{2}\|\theta - \theta_t\|^2. \end{aligned} \tag{20}$$

The first inequality follows from Lemma 1 at θ_{t+1}, and the second follows from the minimizing property in (7). Taking $\theta = \theta_t$ in (20) and using $\tau > C$, we obtain

$$F(\theta_{t+1}) \leqslant F(\theta_t) - \frac{\tau - C}{2}\|\theta_{t+1} - \theta_t\|^2 \leqslant F(\theta_t).$$

This proves the first claim.

It remains to prove (8). Write $D = \operatorname{diam}(\Theta)$ and $\Delta_t = F(\theta_t) - F^\star \geqslant 0$. For any $\alpha \in [0, 1]$, convexity of Θ gives $(1 - \alpha)\theta_t + \alpha\theta^\star \in \Theta$. Substituting this parameter into (20) and using $\tau > C$, we obtain

$$\begin{aligned} F(\theta_{t+1}) &\leqslant F\big((1 - \alpha)\theta_t + \alpha\theta^\star\big) + \frac{\tau\alpha^2}{2}\|\theta^\star - \theta_t\|^2 \\ &\leqslant (1 - \alpha)F(\theta_t) + \alpha F^\star + \frac{\tau\alpha^2 D^2}{2}, \end{aligned}$$

where the last inequality follows from convexity of F and $\|\theta^\star - \theta_t\| \leqslant D$. Subtracting $F^\star$ from both sides gives

$$\Delta_{t+1} \leqslant (1 - \alpha)\Delta_t + \frac{\tau\alpha^2 D^2}{2}. \tag{21}$$

For each $t \geqslant 0$, choose $\alpha = 2/(t + 2) \in (0, 1]$. Then (21) becomes

$$\Delta_{t+1} \leqslant \frac{t}{t + 2}\Delta_t + \frac{2\tau D^2}{(t + 2)^2}.$$

Multiplying by $(t + 1)(t + 2)$ and rearranging yields

$$(t + 1)(t + 2)\Delta_{t+1} - t(t + 1)\Delta_t \leqslant 2\tau D^2 \frac{t + 1}{t + 2}.$$

Summing over $t = 0, \ldots, T - 1$, the intermediate terms on the left cancel and the coefficient of Δ_0 is zero. Hence

$$\begin{aligned} T(T + 1)\Delta_T &= \sum_{t=0}^{T-1} \big((t + 1)(t + 2)\Delta_{t+1} - t(t + 1)\Delta_t\big) \\ &\leqslant 2\tau D^2 \sum_{t=0}^{T-1} \frac{t + 1}{t + 2} \leqslant 2\tau D^2 T, \end{aligned}$$

where the last inequality uses $(t+1)/(t+2) \leqslant 1$. Since $T \geqslant 1$, dividing by $T(T+1)$ and recalling the definitions of D and Δ_T proves (8). □

E.2.1 PROOF OF LEMMA 1

Proof. If $\rho = 0$, the only feasible distribution is $\widehat{P} = R_t$, so $F = f_t$ and the claim holds. Suppose $\rho > 0$, and fix $t \geqslant 0$ and $\theta \in \Theta$. Write $\beta = \beta_t$ and $d(c) = \ell_\theta(c) - \ell_{\theta_t}(c)$. The exponential tilt in (6) is equivalent to $\widehat{P}$, and its log density is bounded by the boundedness of the losses. For any feasible distribution R, changing the reference measure in the KL divergence therefore gives

$$\begin{aligned} D_{\mathrm{KL}}(R\|R_t) &= D_{\mathrm{KL}}(R\|\widehat{P}) - \beta\mathbb{E}_R\ell_{\theta_t} + \log\mathbb{E}_{\widehat{P}}e^{\beta\ell_{\theta_t}} \\ &= D_{\mathrm{KL}}(R\|\widehat{P}) - \rho - \beta\big(\mathbb{E}_R\ell_{\theta_t} - \mathbb{E}_{R_t}\ell_{\theta_t}\big), \end{aligned}$$

where the second equality uses $D_{\mathrm{KL}}(R_t\|\widehat{P}) = \rho$. The first equality also shows that R_t uniquely maximizes the KL-regularized objective: its value at any feasible R is $\beta^{-1}\log\mathbb{E}_{\widehat{P}}e^{\beta\ell_{\theta_t}} - \beta^{-1}D_{\mathrm{KL}}(R\|R_t)$. This identity holds for every R with finite $D_{\mathrm{KL}}(R\|\widehat{P})$, and the second equality shows that calibration makes R_t a constrained maximizer as well. Since $D_{\mathrm{KL}}(R\|\widehat{P}) \leqslant \rho$, it follows that

$$\mathbb{E}_R\ell_\theta - \mathbb{E}_{R_t}\ell_\theta \leqslant \mathbb{E}_R d - \mathbb{E}_{R_t}d - \frac{1}{\beta}D_{\mathrm{KL}}(R\|R_t) \leqslant \frac{1}{\beta}\log\mathbb{E}_{R_t}e^{\beta(d-\mathbb{E}_{R_t}d)}.$$

For the last inequality, apply nonnegativity of relative entropy to the probability law with density $e^{\beta d}/\mathbb{E}_{R_t}e^{\beta d}$ relative to R_t. All these quantities are finite because the losses are bounded.

To bound the centered exponential moment, set $a = G\|\theta - \theta_t\|$. The common Lipschitz bound gives $|d| \leqslant a$ almost surely under R_t. For $s \in \mathbb{R}$, define $K(s) = \log\mathbb{E}_{R_t}e^{s(d-\mathbb{E}_{R_t}d)}$. Boundedness of d permits differentiation under the expectation, and $K''(s)$ is its variance under the probability law proportional to $e^{sd}\,dR_t$. Under any such law, $|d| \leqslant a$, so $K''(s) \leqslant a^2$. Since $K(0) = K'(0) = 0$, integration over $[0, \beta]$ gives $K(\beta) \leqslant \beta^2a^2/2$. Consequently, taking the supremum over feasible R yields

$$F(\theta) - f_t(\theta) \leqslant \frac{\beta G^2}{2}\|\theta - \theta_t\|^2 \leqslant \frac{\bar{\beta}G^2}{2}\|\theta - \theta_t\|^2.$$

Feasibility of R_t gives the lower bound in (19); setting $\theta = \theta_t$ gives equality of the two objectives there, completing the proof. □