ADARECURR: STATE-ADAPTIVE RECURRENT COMPUTATION FOR NEURAL PDE SIMULATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural simulators can substantially reduce the cost of PDE simulation, but autoregressive rollouts typically apply the same amount of inference computation to every physical transition even though different states may benefit differently from additional computation. Some states require only limited recurrent refinement, whereas others continue to improve with deeper refinement. This raises a natural question: can computation be allocated dynamically according to the current physical state as the trajectory evolves?

We introduce AdaRecurr, a state-adaptive recurrent computation framework that turns recurrent depth from a global test-time parameter shared across an entire rollout into an online computation budget selected separately for each physical transition. AdaRecurr builds on a pretrained recurrent neural PDE simulator that can perform a variable number of recurrent refinement iterations for each prediction. Keeping this simulator frozen, AdaRecurr observes the current physical field together with the latent refinement already executed and sequentially decides whether to STOP or CONTINUE. Continuing reuses the current latent state and performs further recurrent refinement, while stopping accepts the current prediction and advances the physical rollout. The controller is trained on self-generated autoregressive trajectories to balance prediction error against additional recurrent computation.

Across three neural PDE benchmarks, AdaRecurr reduces trajectory-level relative L_2 error by 3.6–15.3% over fixed recurrent-depth inference at comparable mean recurrent depth. On viscous Burgers, states receiving more computation exhibit a 36.2% higher mean aggregate estimated shock strength than states receiving less computation. In a controlled compute-demand heterogeneity experiment, we further observe larger gains from adaptive allocation when recurrent-computation requirements vary more strongly across physical states.

These results show that test-time recurrent computation can be allocated dynamically along evolving physical trajectories rather than using a single fixed recurrent depth for the entire rollout.

1 INTRODUCTION

Numerical simulation supports prediction and decision making across science and engineering, from forecasting physical systems to evaluating candidate designs. Many applications require repeated simulation across long trajectories, parameter settings, or design alternatives, making computational efficiency important alongside numerical accuracy. This demand becomes especially pronounced in high-dimensional and large-scale systems, where each simulation may involve fine spatial discretizations and many physical timesteps. Many such systems are described by partial differential equations (PDEs), whose states are typically spatially distributed physical fields such as velocity, pressure, or temperature. Efficient simulation therefore requires useful accuracy to be delivered within the available computational budget (Azizzadenesheli et al., 2024).

Neural simulators offer a way to reduce simulation costs by learning the evolution of physical systems. Using trajectory data, these models learn a mapping from the current physical state to a subsequent state and repeatedly apply this mapping at inference to generate predicted trajectories (Li

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.

et al., 2020b; Kovachki et al., 2023; Pfaff et al., 2021). Learned simulators have demonstrated strong potential in scientific and engineering applications, including weather forecasting and aerodynamic design (Lam et al., 2023; Price et al., 2025; Bleeker et al., 2025). RecurrSim (Abdul Majid et al., 2026) further makes the inference computation of a neural simulator adjustable at test time. For a prediction from the current physical state to the next, the model repeatedly applies a shared recurrent block to progressively refine an internal latent representation. We denote the number of recurrent refinement iterations by the recurrent depth K. A larger K allocates more internal computation to the same prediction, while a smaller K uses less computation. Therefore, the model can adjust K at test time to trade prediction accuracy against inference cost.

However, controlling a single global recurrent depth does not determine how computation should be distributed across an evolving physical trajectory. In autoregressive PDE simulation, the physical state changes over time, and different transitions can differ substantially in prediction difficulty and in how much they benefit from additional refinement. For some transitions, the improvement from further recurrent computation becomes small after only a few iterations, whereas other transitions continue to benefit from deeper refinement. In an autoregressive rollout, each predicted state is fed back as input to predict the next state. Using one fixed K throughout such a rollout therefore assigns the same recurrent computation to every physical transition, regardless of its current demand. The test-time computation problem is thus not only how much computation to use for a rollout, but how to allocate a finite computation budget across its evolving physical states.

State-dependent computational demand is a longstanding principle in numerical PDE simulation. Adaptive methods refine spatial or temporal resolution as solution structures and error indicators evolve, concentrating computation where it contributes most to accuracy (Berger & Oliger, 1984; Mavriplis, 1994; Söderlind, 2002; N'Guessan et al., 2021). For example, sharper gradients, shocks, or rapidly varying structures can require greater local computational resolution. RecurrSim (Abdul Majid et al., 2026) reports an analogous state dependence in latent refinement: different physical states exhibit different refinement behavior as recurrent computation proceeds. These observations suggest that recurrent computation can be treated not only as a global test-time control variable, but as a resource that can be dynamically allocated along a PDE trajectory. The resulting challenge is to determine online, from information already available during inference, whether the current prediction warrants further recurrent refinement.

We introduce AdaRecurr, a state-adaptive recurrent computation framework for pretrained neural PDE simulators that can perform a variable number of recurrent refinement iterations for each prediction. For each physical transition, AdaRecurr does not preassign a fixed recurrent depth; instead, a controller sequentially selects STOP or CONTINUE as refinement proceeds. The controller observes the current physical field together with the executed latent refinement trace. A CONTINUE decision reuses the current latent state and continues recurrent refinement from the computation already performed, while a STOP decision accepts the current prediction as the next physical state and advances the rollout to the next physical timestep. The pretrained simulator remains frozen throughout, enabling state-adaptive computation allocation without retraining the backbone.

To learn these stopping decisions without modifying the pretrained simulator, we train only the controller using self-generated autoregressive rollouts. After a prediction is accepted, that predicted state, rather than the corresponding ground-truth state, becomes the input to the next physical transition. The controller is rewarded for accurate next-state predictions, while each CONTINUE decision incurs a penalty for the additional recurrent iterations it uses. Consequently, later stopping decisions during training are made on model-generated states shaped by earlier predictions and computation-allocation decisions. This exposes the controller during training to the evolving model-generated states encountered at inference. Across three neural PDE benchmarks, AdaRecurr reduces trajectory-level relative L_2 error by 3.6–15.3% over fixed recurrent-depth inference at comparable mean recurrent depth. Further analysis on viscous Burgers shows that states receiving more computation tend to exhibit stronger shock structures. We also find that adaptive allocation achieves larger gains when recurrent-computation requirements vary more strongly across physical states.

Our contributions are as follows:

- We extend recurrent depth from a global test-time control variable for the full rollout to an online computation budget selected separately for each physical transition.

- We formulate computation allocation as a bounded sequential stopping problem over a frozen recurrent simulator. The controller uses the current physical field and executed latent refinement trace to select STOP or CONTINUE, while directly reusing recurrent computation already performed.
- We train the stopping policy on self-generated autoregressive trajectories and evaluate it across three PDE benchmarks, showing lower rollout error at comparable mean recurrent depth and analyzing how computation allocation relates to physical-state structure and compute-demand heterogeneity.

2 BACKGROUND AND RELATED WORK

2.1 PARTIAL DIFFERENTIAL EQUATIONS

Partial differential equations (PDEs) describe the spatial and temporal evolution of physical fields (Evans, 2010). After spatial discretization, we denote the physical state at timestep t by U_t and a trajectory by $U_0, U_1, \ldots, U_T$.

2.2 NEURAL PDE SIMULATORS

Conventional neural PDE simulators. Neural PDE simulators approximate the evolution between physical states with a parameterized model $\mathcal{G}_\theta$,

$$\hat{U}_{t+1} = \mathcal{G}_\theta(\hat{U}_t), \tag{1}$$

and generate trajectories by autoregressively feeding each prediction into the next transition. Convolutional architectures, U-Nets, Fourier neural operators, and graph-based simulators provide widely used parameterizations of this mapping (Gupta & Brandstetter, 2023; Li et al., 2020b; Kovachki et al., 2023; Pfaff et al., 2021). In conventional one-pass neural simulators, each physical transition follows a predetermined computation graph, so different states typically use the same model computation depth.

Recurrent-depth neural simulators. RecurrSim (Abdul Majid et al., 2026) introduces repeatable internal computation within a single physical transition. Given a current state U_t, the model repeatedly applies a shared recurrent block to progressively refine an internal latent representation. We denote by the recurrent depth K the number of recurrent refinement iterations used to predict the next physical state, and write the prediction at depth K as

$$\hat{U}_{t+1}^{(K)}. \tag{2}$$

A larger K allocates more internal computation to the current prediction, while a smaller K uses less computation. The model can therefore adjust K at test time to trade prediction accuracy against inference cost.

RecurrSim further reports that different input states exhibit different refinement behavior as recurrent computation increases (Abdul Majid et al., 2026). This suggests that different states may have different recurrent-computation requirements, providing a basis for state-wise computation allocation.

2.3 RELATED WORK

Adaptive computation in neural networks. Input-dependent computation has been studied through learned halting and iterative inference. Adaptive Computation Time (ACT) learns a halting mechanism that allows recurrent models to execute different numbers of internal computation steps across inputs (Graves, 2016). PonderNet formulates computation depth as a stochastic stopping process and learns when to continue computation or emit the current prediction (Banino et al., 2021). Deep Equilibrium Models construct implicit-depth representations through fixed-point computation (Bai et al., 2019). Together, these approaches show that internal computation depth can be determined from the current input and intermediate representation.

AdaRecurr uses a related sequential-stopping abstraction in a different setting: stopping decisions are embedded within an autoregressive physical trajectory, where each accepted prediction changes

the model-generated state on which subsequent decisions are made. The controller conditions on the current physical field and the recurrent refinement already executed, continuation directly reuses the latent computation already performed, and the recurrent simulator remains frozen. AdaRecurr therefore focuses on trajectory-wise computation allocation for recurrent neural simulators with adjustable depth, where stopping decisions interact through the evolving model-generated physical trajectory.

Adaptive computation in numerical simulation. State-dependent resource allocation is also a central principle in numerical PDE simulation. Adaptive mesh refinement adjusts spatial resolution according to the evolving solution structure, concentrating computation around sharp gradients, interfaces, and other localized features (Berger & Oliger, 1984; Mavriplis, 1994). Adaptive time-stepping varies temporal resolution along the evolving solution (Söderlind, 2002). For multiscale incompressible flow and scalar transport, dynamic mesh adaptation and error control similarly redistribute spatial and temporal computation as the solution evolves (N'Guessan et al., 2021). These methods show that numerical computation can be allocated according to the current physical state rather than kept fixed throughout a trajectory.

Learned approaches extend this principle to data-driven decision policies. Recurrent networks can learn mesh-refinement strategies, while reinforcement learning formulations cast mesh adaptation as sequential resource allocation and learn refinement policies directly from simulation (Bohn & Feischl, 2021; Yang et al., 2023a; Foucart et al., 2023; Yang et al., 2023b; Freymuth et al., 2023; 2026). These works primarily adapt the numerical discretization, such as the spatial mesh or temporal resolution. Recurrent-depth neural simulation instead provides another allocatable resource: the amount of internal recurrent computation used to predict a physical transition.

AdaRecurr targets this computation axis with state-wise allocation. For each physical transition, the controller uses the current physical field and the executed latent refinement trace to select STOP or CONTINUE, determining recurrent depth online. Section 3 presents the stopping formulation, controller observation, training objective, and adaptive inference procedure.

3 METHOD

AdaRecurr turns recurrent-depth selection from a single rollout-level choice into a state-adaptive sequential stopping problem. For each physical transition, a controller observes the current physical field together with the recurrent refinement already executed and decides whether to accept the current prediction or allocate additional recurrent computation. The pretrained simulator remains frozen throughout controller training. Figure 1 summarizes the overall computation flow.

3.1 FROZEN RECURRENT BACKBONE

Let $\hat{U}_t$ denote the simulator state at physical timestep t. We use a pretrained recurrent neural simulator with adjustable depth following RecurrSim (Abdul Majid et al., 2026). Given $\hat{U}_t$, the backbone computes

$$c_t = E_\theta(\hat{U}_t), \qquad z_t^{(k+1)} = F_\theta\left(z_t^{(k)}, c_t\right), \tag{3}$$

and reconstructs a prediction at recurrent depth k as

$$\hat{U}_{t+1}^{(k)} = \mathcal{R}_\theta\left(\hat{U}_t, z_t^{(k)}\right). \tag{4}$$

Here, k denotes the number of recurrent refinement iterations already executed for the same physical transition $\hat{U}_t \to \hat{U}_{t+1}$. Because the shared recurrent block supports prediction at different depths, AdaRecurr treats recurrent depth as an online computation budget while keeping all backbone parameters θ fixed.

3.2 STATE-ADAPTIVE STOPPING CONTROLLER

AdaRecurr selects a final recurrent depth K_t separately for each physical transition. Decisions are made on the discrete depth grid

$$\mathcal{K} = \{k_j = K_{\min} + j\Delta K : j = 0, \ldots, J\}, \qquad K_{\max} = K_{\min} + J\Delta K. \tag{5}$$

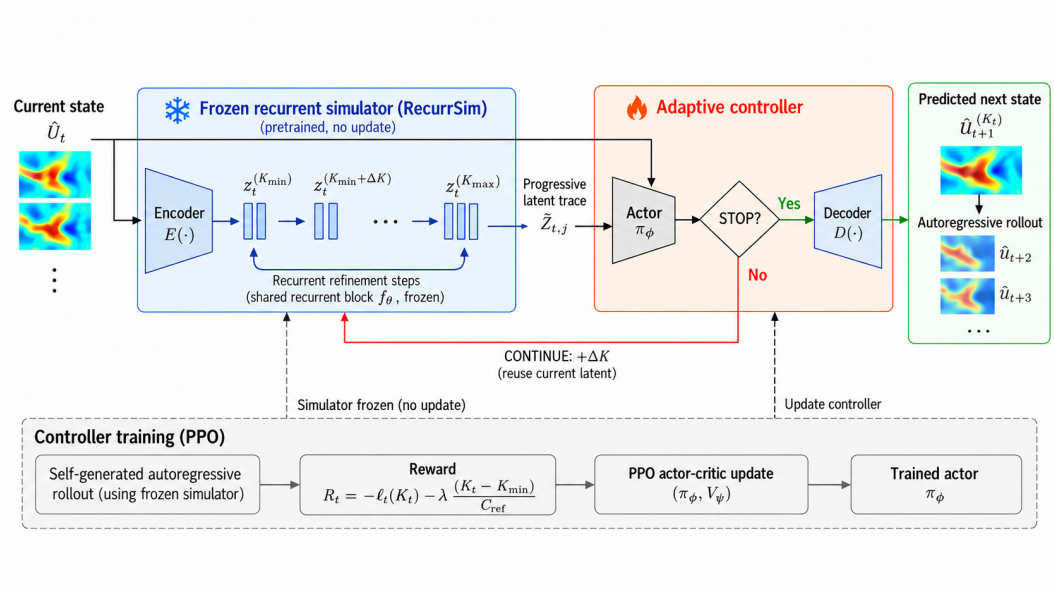

Figure 1: **Overview of AdaRecurr.** For each physical transition, the frozen recurrent simulator produces a progressive latent refinement trace. The controller conditions on the current physical field and the available latent refinement trace to decide whether to stop or allocate additional recurrent iterations. Once stopped, the accepted prediction advances the physical rollout.

Each prediction first executes $K_{\min}$ recurrent iterations. The controller then decides at each checkpoint whether the current refinement is sufficient. As computation proceeds, the available latent history grows: the first decision observes only $z_t^{(K_{\min})}$, and each CONTINUE decision reveals the next computed checkpoint.

We represent this progressive history with a fixed-length trace, zero padding checkpoints that have not yet been executed:

$$\widetilde{\mathcal{Z}}_{t,j} = \left(\tilde{z}_{t,0}^{(j)}, \ldots, \tilde{z}_{t,J}^{(j)}\right), \qquad \tilde{z}_{t,i}^{(j)} = \begin{cases} z_t^{(k_i)}, & i \le j, \\ \mathbf{0}, & i > j. \end{cases} \tag{6}$$

The controller observation is

$$o_{t,j} = \left(\hat{U}_t, \widetilde{\mathcal{Z}}_{t,j}\right). \tag{7}$$

As recurrent refinement continues, each newly computed latent checkpoint fills the next trace slot while previously executed computation is retained.

A spatial encoder maps $\hat{U}_t$ to a state representation, a latent encoder maps the executed latent checkpoints to trace representations, and a sequence encoder aggregates them into a controller representation $q_{t,j}$. The actor defines the binary policy

$$a_{t,j} \sim \pi_\phi(\cdot \mid o_{t,j}), \qquad a_{t,j} \in \{\text{STOP}, \text{CONTINUE}\}, \tag{8}$$

and a critic $V_\psi(o_{t,j})$ estimates the expected return.

If $a_{t,j} = \text{CONTINUE}$, the simulator resumes directly from the current latent state $z_t^{(k_j)}$ and executes recurrent refinement to the next checkpoint. If $a_{t,j} = \text{STOP}$, the current depth is accepted,

$$K_t = k_j, \qquad \hat{U}_{t+1} = \hat{U}_{t+1}^{(K_t)}, \tag{9}$$

and the accepted prediction becomes the input to the next physical timestep. Stopping is forced at $K_{\max}$, so each physical transition has a bounded recurrent-computation budget.

3.3 CONTROLLER TRAINING

The frozen simulator acts as the environment in which the controller learns computation allocation. For a prediction accepted at depth K_t, we define the next-state relative L_2 error

$$\ell_t(K_t) = \frac{\|\hat{U}_{t+1}^{(K_t)} - U_{t+1}\|_2}{\|U_{t+1}\|_2 + \varepsilon}, \tag{10}$$

where $\varepsilon > 0$ is a numerical stabilizer.

We optimize a return that balances prediction error and recurrent computation

$$R_t = -\ell_t(K_t) - \lambda \frac{K_t - K_{\min}}{C_{\text{ref}}}, \tag{11}$$

where $\lambda \geq 0$ controls the computation penalty and $C_{\text{ref}} > 0$ normalizes recurrent cost. This return decomposes naturally over the stopping sequence. Each CONTINUE receives

$$r_{t,j}^{\text{cont}} = -\lambda \frac{\Delta K}{C_{\text{ref}}}, \tag{12}$$

while the final STOP receives

$$r_{t,j}^{\text{stop}} = -\ell_t(K_t). \tag{13}$$

The controller therefore learns whether the expected predictive benefit of further refinement justifies its additional computation cost.

We train the actor and critic with proximal policy optimization (PPO) (Schulman et al., 2017). Optimization hyperparameters are provided in Appendix A, and the complete training procedure is given in Algorithm 1.

Controller training uses self-generated autoregressive rollouts. Starting from $\hat{U}_0 = U_0$, each accepted prediction becomes the next physical input,

$$\hat{U}_0 \xrightarrow{K_0} \hat{U}_1 \xrightarrow{K_1} \hat{U}_2 \xrightarrow{K_2} \cdots \xrightarrow{K_{T-1}} \hat{U}_T. \tag{14}$$

Later stopping decisions are therefore made on rollout states produced by earlier predictions and computation-allocation decisions rather than repeatedly on ground-truth inputs. Each physical transition forms one bounded stopping episode: return and advantage are computed within that transition, while the accepted prediction continues the autoregressive rollout. AdaRecurr thus learns from the state distribution induced by its own stopping decisions without requiring optimal-depth supervision.

3.4 ADAPTIVE INFERENCE

At inference, AdaRecurr uses the frozen simulator and the trained actor. Each physical transition begins with $K_{\min}$ recurrent iterations, after which the actor repeatedly selects

$$a_{t,j} = \arg\max_a \pi_\phi(a \mid o_{t,j}). \tag{15}$$

A CONTINUE decision resumes refinement from the current latent state and fills the next trace slot. A STOP decision accepts the current depth, produces $\hat{U}_{t+1}$, and advances the physical rollout. Consequently, different physical transitions within the same rollout can receive different recurrent depths K_t, each selected online from the current physical state and the refinement progress already observed. The complete adaptive inference procedure is provided in Algorithm 2.

4 EXPERIMENTS

Complete experimental details are provided in Appendix A. Across all benchmarks, AdaRecurr operates on pretrained recurrent neural simulators with adjustable depth whose backbone parameters remain frozen during controller training.

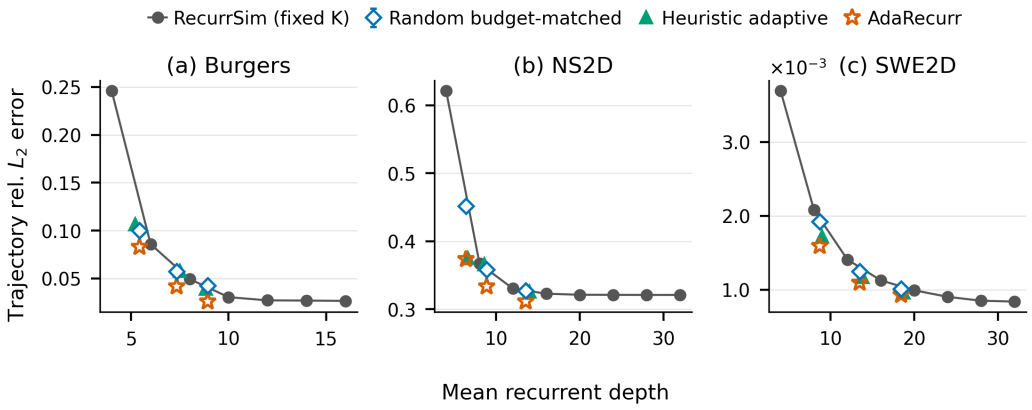

Figure 2: **Accuracy–compute comparison across three PDE benchmarks.** Trajectory-level relative L_2 error is reported against mean recurrent depth for fixed-K RecurrSim, Random Budget-Matched allocation, Heuristic Adaptive stopping, and AdaRecurr.

4.1 Benchmark PDEs

We evaluate AdaRecurr on three PDE benchmarks: viscous Burgers from PDEBench (Takamoto et al., 2022), buoyancy-driven incompressible Navier–Stokes (NS2D) from PDEArena (Gupta & Brandstetter, 2023; Brandstetter et al., 2023), and radial dam-break shallow water (SWE2D) from PDEBench (Takamoto et al., 2022). They exhibit distinct evolving structures: steep fronts in Burgers, buoyancy-driven smoke transport in NS2D, and outward-propagating waves in SWE2D. Governing equations and benchmark details are provided in Appendix A.1.

4.2 Accuracy–Compute Comparison

Using a fixed recurrent depth K assigns the same computation to every physical transition, whereas AdaRecurr selects K_t separately for each state. We therefore test whether state-dependent allocation improves rollout accuracy at comparable mean recurrent depth.

Experimental setup. We compare AdaRecurr with three computation-allocation strategies using the same evaluation trajectories and rollout horizons: fixed-depth RecurrSim, Random Budget-Matched allocation, and Heuristic Adaptive stopping. Fixed-depth RecurrSim uses the same K for every physical transition. Random Budget-Matched varies K_t randomly and independently of the current physical state while matching the target mean computation budget, testing whether simply varying the amount of computation across states is sufficient. Heuristic Adaptive uses no learned controller; instead, it stops according to the change between successive latent refinements, providing a comparison with a simple non-learned stopping rule.

We use mean recurrent depth as the computation axis and trajectory-level relative L_2 error as the accuracy metric. Mean recurrent depth measures recurrent-backbone computation and is the average number of recurrent refinement iterations used per physical transition across an evaluated rollout, so a smaller value indicates lower computation. A smaller relative L_2 error indicates higher prediction accuracy. For fixed-depth RecurrSim, the mean recurrent depth equals the fixed depth K; for adaptive methods, it is averaged over all physical transitions in the evaluated rollouts. Detailed evaluation settings are provided in Appendix A. Each AdaRecurr operating point in Fig. 2 reports the mean over five independent controller-training seeds; the corresponding standard deviations in trajectory-level relative L_2 error are reported in Appendix C.2.

Results. As shown in Fig. 2, AdaRecurr reduces trajectory-level relative L_2 error by 3.6–15.3% relative to fixed recurrent-depth inference at comparable mean recurrent depth. Across the nine target budgets, it further reduces error by 4.9–39.1% relative to Random Budget-Matched alloca-

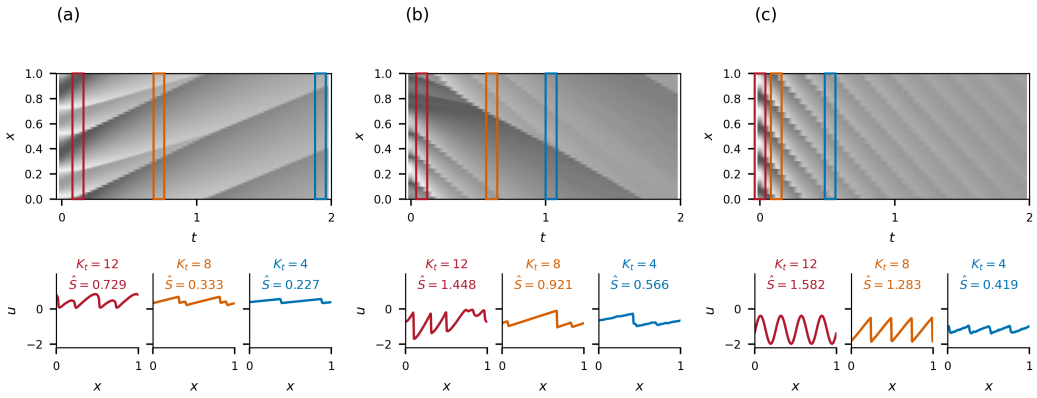

Figure 3: **State-aware computation allocation within Burgers rollouts.** Each panel shows one model-generated autoregressive Burgers rollout in the (t, x) plane. Red, orange, and blue boxes mark complete spatial states assigned $K_t = 12$, $K_t = 8$, and $K_t = 4$, respectively; the corresponding one-dimensional physical fields are shown below in matching colors.

tion and by 0.5–33.8% relative to Heuristic Adaptive stopping, showing that state-aware allocation matters beyond simply varying recurrent depth.

4.3 State-Aware Computation Reallocation

We next analyze what kinds of physical states receive more recurrent computation from AdaRecurr. In the Burgers benchmark, we examine whether the recurrent depth selected by the controller is associated with the strength of shock structures in the current state. Here, a shock refers to a steep front where the solution changes rapidly, and a stronger shock corresponds to a larger state change across the front.

Experimental setup. We analyze the viscous Burgers test set. For each rollout state u_t, we record the recurrent depth K_t selected by the controller.

For each physical state, we detect shock fronts and sum their estimated strengths to obtain the aggregate estimated shock strength $\hat{S}_t$. A larger $\hat{S}_t$ therefore indicates that the state contains stronger shock structure overall. Details of the shock-strength estimation are provided in Appendix A.6.

Using the controller's mean recurrent depth $\bar{K}$ over the test set as the reference, we divide states into low- and high-computation groups according to whether $K_t < \bar{K}$ or $K_t > \bar{K}$. We then compare the aggregate estimated shock strength between the two groups.

We further visualize physical states assigned different recurrent depths within the same autoregressive rollouts in Fig. 3, allowing direct comparison between the controller's computation allocation and the shock structures present in the states.

Results. Among states with valid aggregate shock-strength estimates, the high-computation group has a mean aggregate estimated shock strength of 1.378, compared with 1.011 for the low-computation group, corresponding to a 36.2% increase. Thus, states receiving more recurrent computation also tend to exhibit stronger shock structures, consistent with AdaRecurr adapting computation to the current physical state. Fig. 3 further shows that different physical states within the same rollout can receive different recurrent depths.

4.4 Effect of Compute-Demand Heterogeneity

The preceding experiments show that AdaRecurr can assign different recurrent depths to different physical states. This adaptive allocation is most useful when states genuinely require different amounts of computation: if most states favor similar recurrent depths, a single fixed K can serve

Table 1: Effect of state-wise compute-demand heterogeneity on adaptive recurrent-depth allocation. Gain reports relative error reduction against the lower / upper fixed-K operating points.

	LOW heterogeneity			HIGH heterogeneity		
Method	Avg K	Rel. L_2	Gain	Avg K	Rel. L_2	Gain
Lower Fixed-K	4.00	0.4021	–	8.00	0.3183	–
AdaRecurr	4.49	0.3992	+0.7% / +1.3%	7.73	0.3089	+2.9% / +0.6%
Oracle Adaptive	4.82	0.3979	+1.0% / +1.6%	8.39	0.3065	+3.7% / +1.4%
Upper Fixed-K	8.00	0.4046	–	12.00	0.3108	–

much of the trajectory; if some states need only shallow refinement while others benefit from deeper refinement, there is greater scope for state-adaptive allocation. We therefore ask whether the benefit of AdaRecurr increases as state-wise compute demand becomes more heterogeneous.

Experimental setup. Across the LOW and HIGH settings, we keep the same NS2D recurrent backbone, controller architecture, PPO configuration, λ, and recurrent-depth grid $\mathcal{K} = \{4, 8, \ldots, 32\}$, and vary only the distribution of state-wise compute demand. Intuitively, the compute demand of a state indicates how much recurrent refinement is useful for that state under the same accuracy–compute objective used to train the controller. The exact construction is given in Appendix C.1.

In the LOW setting, 80% of states are associated with $K = 4$, and all states fall within $K \leq 12$, so most states favor similarly shallow computation. In the HIGH setting, 40% of states are associated with $K = 4$, while approximately 40% fall at $K \geq 16$, producing a broader mixture of shallow- and deep-demand states. Grouped depth distributions are reported in Appendix C.1.

We compare AdaRecurr with two nearby fixed-depth operating points. Lower Fixed-K and Upper Fixed-K use the corresponding discrete depths. We also report Oracle Adaptive as a reference: for each state, it selects the candidate recurrent depth that optimizes the same accuracy–compute objective used for controller training. This estimates the benefit available from ideal state-wise allocation rather than serving as a deployable inference baseline. Table 1 reports mean recurrent depth (Avg K), trajectory-level relative L_2 error, and relative error reduction against the two fixed-depth operating points.

Results. In this controlled comparison, adaptive recurrent depth becomes more beneficial as state-wise compute demand becomes more heterogeneous. In the LOW setting, AdaRecurr improves over the neighboring fixed-depth operating points by about 1%, while Oracle Adaptive reaches a maximum gain of 1.6%.

The advantage is larger in the HIGH setting. At an average recurrent depth of 7.73, AdaRecurr reduces error by 2.9% relative to fixed $K = 8$ while using slightly less recurrent computation, and it also reduces error by 0.6% relative to fixed $K = 12$. Oracle Adaptive follows the same trend, with its maximum gain increasing from 1.6% in the LOW setting to 3.7% in the HIGH setting. The larger gains for both AdaRecurr and Oracle Adaptive are consistent with the central hypothesis that state-adaptive allocation has more opportunity to help when recurrent-computation requirements vary more strongly across physical states.

5 CONCLUSION

We introduced AdaRecurr, a state-adaptive recurrent computation framework that turns recurrent depth from a rollout-level test-time parameter into an online budget selected separately for each physical transition over a frozen recurrent neural simulator with adjustable depth.

Across three neural PDE benchmarks, AdaRecurr reduces trajectory-level relative L_2 error by 3.6–15.3% over fixed recurrent-depth inference at comparable mean recurrent depth. On viscous Burgers, higher-computation states exhibit a 36.2% higher mean aggregate estimated shock strength, and controlled NS2D experiments show larger adaptive-allocation gains under stronger compute-demand heterogeneity. These results demonstrate that recurrent computation can be allocated dynamically along evolving physical trajectories rather than fixed for an entire rollout.

AI USE STATEMENT

Generative AI tools were used in an assistive role during the research process, including discussing selected research ideas and implementation approaches, assisting with software debugging, organizing related literature, and improving the manuscript's structure, language, terminology, and readability. The research methodology, experimental design, and final scientific conclusions were determined by the authors and validated through the underlying experiments and source materials. All AI-assisted outputs were reviewed by the authors, who take full responsibility for the methodology, experimental results, analysis, and final content of this work.

REFERENCES

Harris Abdul Majid, Pietro Sittoni, and Francesco Tudisco. Test-time accuracy-cost control in neural simulators via recurrent-depth. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/a3b73551d62d7758bdd99fc018b167a9-Paper-Conference.pdf.

Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics*, 6:320–328, 2024. doi: 10.1038/s42254-024-00712-5. URL https://www.nature.com/articles/s42254-024-00712-5.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models, 2019. URL https://arxiv.org/abs/1909.01377.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder, 2021. URL https://arxiv.org/abs/2107.05407.

Marsha J. Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484–512, 1984. doi: 10.1016/0021-9991(84)90073-1. URL https://www.sciencedirect.com/science/article/pii/0021999184900731.

Maurits Bleeker, Matthias Dorfer, Tobias Kronlachner, Reinhard Sonnleitner, Benedikt Alkin, and Johannes Brandstetter. NeuralCFD: Deep learning on high-fidelity automotive aerodynamics simulations, 2025. URL https://arxiv.org/abs/2502.09692v1.

Jan Bohn and Michael Feischl. Recurrent neural networks as optimal mesh refinement strategies. *Computers & Mathematics with Applications*, 97:61–76, 2021. doi: 10.1016/j.camwa.2021.05.018. URL https://doi.org/10.1016/j.camwa.2021.05.018.

Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K. Gupta. Clifford neural layers for pde modeling. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=okwxL_c4x84.

Lawrence C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, 2 edition, 2010. URL https://bookstore.ams.org/gsm-19-r.

Corbin Foucart, Aaron Charous, and Pierre F. J. Lermusiaux. Deep reinforcement learning for adaptive mesh refinement. *Journal of Computational Physics*, 491:112381, 2023. doi: 10.1016/j.jcp.2023.112381. URL https://doi.org/10.1016/j.jcp.2023.112381.

Niklas Freymuth, Philipp Dahlinger, Tobias Würth, Simon Reisch, Luise Kärger, and Gerhard Neumann. Swarm reinforcement learning for adaptive mesh refinement. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/e85454a113e8b41e017c81875ae68d47-Abstract-Conference.html.

Niklas Freymuth, Philipp Dahlinger, Tobias Würth, Simon Reisch, Luise Kärger, and Gerhard Neumann. Adaptive swarm mesh refinement using deep reinforcement learning with local rewards. *Machine Learning*, 115:169, 2026. doi: 10.1007/s10994-026-07064-4. URL https://link.springer.com/article/10.1007/s10994-026-07064-4.

Alex Graves. Adaptive computation time for recurrent neural networks, 2016. URL https://arxiv.org/abs/1603.08983.

Jayesh K. Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=dPSTDbGtBY.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023. URL https://www.jmlr.org/papers/v24/21-1524.html.

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, Alexander Merose, Stephan Hoyer, George Holland, Oriol Vinyals, Jacklynn Stott, Alexander Pritzel, Shakir Mohamed, and Peter Battaglia. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023. doi: 10.1126/science.adi2336. URL https://www.science.org/doi/10.1126/science.adi2336.

Qin Li, Jian-Guo Liu, and Ruiwen Shu. Sensitivity analysis of burgers' equation with shocks. *SIAM/ASA Journal on Uncertainty Quantification*, 8(4):1493–1521, 2020a. doi: 10.1137/18M1211763. URL https://epubs.siam.org/doi/10.1137/18M1211763.

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

Catherine Mavriplis. Adaptive mesh strategies for the spectral element method. *Computer Methods in Applied Mechanics and Engineering*, 116(1–4):77–86, 1994. doi: 10.1016/S0045-7825(94)80010-3. URL https://www.sciencedirect.com/science/article/pii/S0045782594800103.

Marc-Arthur N'Guessan, Marc Massot, Laurent Séries, and Christian Tenaud. High order time integration and mesh adaptation with error control for incompressible navier–stokes and scalar transport resolution on dual grids. *Journal of Computational and Applied Mathematics*, 387:112542, 2021. doi: 10.1016/j.cam.2019.112542. URL https://www.sciencedirect.com/science/article/pii/S0377042719305473.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Tom R. Andersson, Andrew El-Kadi, Dominic Masters, Timo Ewalds, Jacklynn Stott, Shakir Mohamed, Peter Battaglia, Remi Lam, and Matthew Willson. Probabilistic weather forecasting with machine learning. *Nature*, 637:84–90, 2025. doi: 10.1038/s41586-024-08252-9. URL https://www.nature.com/articles/s41586-024-08252-9.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Gustaf Söderlind. Automatic control and adaptive time-stepping. *Numerical Algorithms*, 31:281–310, 2002. doi: 10.1023/A:1021160023092. URL https://doi.org/10.1023/A:1021160023092.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: An extensive benchmark for scientific machine learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 1596–1611, 2022. doi: 10.52202/068431-0117. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/0a9747136d411fb83f0cf81820d44afb-Paper-Datasets_and_Benchmarks.pdf.

Jiachen Yang, Tarik Dzanic, Brenden Petersen, Jun Kudo, Ketan Mittal, Vladimir Tomov, Jean-Sylvain Camier, Tuo Zhao, Hongyuan Zha, Tzanio Kolev, Robert Anderson, and Daniel Faissol. Reinforcement learning for adaptive mesh refinement. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pp. 5997–6014. PMLR, 2023a. URL https://proceedings.mlr.press/v206/yang23e.html.

Jiachen Yang, Ketan Mittal, Tarik Dzanic, Socratis Petrides, Brendan Keith, Brenden Petersen, Daniel Faissol, and Robert Anderson. Multi-agent reinforcement learning for adaptive mesh refinement. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems*, pp. 14–22. International Foundation for Autonomous Agents and Multiagent Systems, 2023b. URL https://aamas.csc.liv.ac.uk/Proceedings/aamas2023/pdfs/p14.pdf.

A Experimental Details

A.1 Benchmark PDEs

We consider three PDE benchmarks: one-dimensional viscous Burgers, two-dimensional buoyancy-driven incompressible Navier–Stokes, and two-dimensional shallow water.

Burgers equation. The PDEBench Burgers benchmark (Takamoto et al., 2022) is governed by

$$\partial_t u + \partial_x\left(\frac{u^2}{2}\right) = \frac{\nu}{\pi}\,\partial_{xx} u, \qquad \nu = 0.001, \tag{16}$$

on $x \in [0, 1]$ and $t \in [0, 2]$ with periodic boundary conditions.

Incompressible Navier–Stokes equations. For the PDEArena buoyancy-driven Navier–Stokes benchmark (Gupta & Brandstetter, 2023; Brandstetter et al., 2023), let u denote the transported smoke scalar and $\mathbf{v} = (v_x, v_y)^\top$ the velocity field. The dynamics are

$$\begin{aligned} \partial_t u + \mathbf{v}\cdot\nabla u &= 0, \\ \partial_t \mathbf{v} + (\mathbf{v}\cdot\nabla)\mathbf{v} &= -\nabla p + \nu\nabla^2\mathbf{v} + \beta u\,\mathbf{e}_y, \\ \nabla\cdot\mathbf{v} &= 0, \end{aligned} \tag{17}$$

where p is the pressure, $\nu = 0.01$, $\beta = 0.5$, and $\mathbf{e}_y$ is the vertical unit vector. The velocity satisfies homogeneous Dirichlet boundary conditions and the smoke scalar homogeneous Neumann boundary conditions. The smoke scalar is initialized with random Gaussian fluctuations, while the velocity field is initialized at rest (Brandstetter et al., 2023).

Shallow-water equations. The PDEBench radial dam-break benchmark (Takamoto et al., 2022) uses the shallow-water equations on a flat bottom,

$$\begin{aligned} \partial_t h + \partial_x(hu) + \partial_y(hv) &= 0, \\ \partial_t(hu) + \partial_x\left(hu^2 + \frac{1}{2}gh^2\right) + \partial_y(huv) &= 0, \\ \partial_t(hv) + \partial_x(huv) + \partial_y\left(hv^2 + \frac{1}{2}gh^2\right) &= 0, \end{aligned} \tag{18}$$

where h is the water depth, (u, v) are the depth-averaged horizontal velocity components, and $g = 1$. The domain is $\Omega = [-2.5, 2.5]^2$ over $t \in [0, 1]$ with extrapolation boundary conditions. Each trajectory is initialized from still water with a circular depth discontinuity,

$$h(x, y, 0) = \begin{cases} 2, & \sqrt{x^2 + y^2} \le R, \\ 1, & \sqrt{x^2 + y^2} > R, \end{cases} \qquad (u, v)(x, y, 0) = \mathbf{0}, \qquad R \sim \mathcal{U}(0.3, 0.7). \tag{19}$$

Table 2: Dataset and preprocessing summary. Training, validation, and test trajectory counts are reported directly.

Dataset	Spatial grid	States	Train	Validation	Test
Burgers	256	51	1000	100	100
NS2D	128×128	14	5200	1300	800
SWE2D	128×128	101	720	80	200

Table 3: Backbone architecture configurations.

Dataset	Backbone	Hidden width	Fourier modes	In/Out channels	Batch size
Burgers	Recurrent FNO	196	32	1/1	256
NS2D	Recurrent FNO	64	12×12	3/3	8
SWE2D	Recurrent FNO	64	12×12	1/1	8

A.2 Data Details

We use three public PDE datasets: PDEBench Burgers (Takamoto et al., 2022), PDEArena Navier–Stokes 2D (Gupta & Brandstetter, 2023; Brandstetter et al., 2023), and PDEBench shallow water (Takamoto et al., 2022). Table 2 summarizes the effective data resolution and split used after preprocessing. Normalization statistics are fitted exclusively on the corresponding training split.

For Burgers, the original PDEBench trajectories contain 201 temporal states on 1,024 spatial grid points. We uniformly subsample time and space by a factor of four, yielding 51 states, 50 one-step transitions, and 256 spatial points per trajectory, with prediction interval $\Delta t = 0.04$. We use 1,000 training, 100 validation, and 100 test trajectories.

For NS2D, each trajectory contains the smoke scalar u and velocity components (v_x, v_y) on a 128×128 grid. The 14 stored physical states define 13 one-step transitions with prediction interval $\Delta t = 6.0$. The model state consists of the three channels (u, v_x, v_y). Source fields are stored in `float64` and converted to `float32` for model computation. State and residual normalization use per-channel means and standard deviations fitted on the official training split. We use 5,200 training, 1,300 validation, and 800 test trajectories.

For SWE2D, each trajectory contains 101 states of the water-depth field h on a 128×128 grid, corresponding to 100 one-step transitions. We use 720 training, 80 validation, and 200 test trajectories.

A.3 Backbone Architectures and Training

All experiments use a pretrained recurrent neural simulator with adjustable depth as the backbone. The backbone parameters remain frozen throughout AdaRecurr controller training and inference. All three benchmarks use recurrent encoder–decoder architectures based on Fourier Neural Operators, where a shared recurrent block is repeatedly applied to refine the latent representation. The main architectural configurations are summarized in Table 3.

For Burgers, the model input and output are the scalar field u; for NS2D, the model operates on the three-channel physical state (u, v_x, v_y); and for SWE2D, the input and output are the water-depth field h. All backbones include an encoder and decoder and initialize the recurrent latent state from a Gaussian distribution.

The backbones are trained for one-step residual prediction. Given the current physical state U_t, the model predicts

$$\Delta U_t = U_{t+1} - U_t, \tag{20}$$

and forms the next-state prediction as

$$\hat{U}_{t+1} = U_t + \widehat{\Delta U}_t. \tag{21}$$

All three backbones are trained for 100 epochs with AdamW, an initial learning rate of 3×10^{-4}, weight decay 10^{-5}, and a cosine learning-rate schedule decaying to 3×10^{-6}. During training, the recurrent depth is sampled independently for each batch from a Poisson–lognormal-plus-one distribution with $\bar{K} = 32$ and $\sigma = 0.5$. We use 4-step truncated backpropagation through time and

Table 4: AdaRecurr controller training hyperparameters.

Hyperparameter	Value
Recurrent-depth grid	$\{4, 8, 12, 16, 20, 24, 28, 32\}$
PPO optimizer	Adam
Learning rate	3×10^{-4}
Discount factor γ	1.0
GAE parameter	0.95
PPO clip	0.2
Entropy coefficient	0.01
Value coefficient	0.5
PPO epochs / update	10
Decisions / update	4096
Minibatch size	256
Gradient clipping	0.5

gradient-norm clipping at 1. After training, the checkpoint with the lowest validation loss is fixed for all subsequent AdaRecurr controller training and evaluation.

A.4 AdaRecurr Controller Configuration

AdaRecurr makes stopping decisions on the recurrent-depth checkpoints

$$\mathcal{K} = \{4, 8, 12, 16, 20, 24, 28, 32\}. \tag{22}$$

Each physical transition first executes four recurrent iterations. Starting at $K = 4$, the controller selects STOP or CONTINUE at each checkpoint; continuation resumes from the current latent state, and the transition terminates at $K = 32$.

The controller receives the current simulator state $\hat{U}_t$ together with the latent checkpoints computed so far. The checkpoints are stored in a fixed-length trace, with unexecuted slots zero padded. The simulator state and latent checkpoints are encoded separately and aggregated into the controller representation used by the actor and critic.

For a prediction accepted at depth K_t, the reward combines next-state relative L_2 error with recurrent computation cost, with the latter normalized by

$$\frac{K_t - 4}{32}. \tag{23}$$

The computation weight λ determines the accuracy–compute operating point.

Controller training uses autoregressive rollouts, with each accepted prediction becoming the input to the next physical transition. The rollout horizons are 50, 13, and 50 transitions for Burgers, NS2D, and SWE2D, respectively. Actions are sampled from the categorical actor policy during training; validation and inference use the deterministic maximum-probability action.

A.5 Accuracy–Compute Evaluation Protocol

We evaluate all methods with autoregressive rollouts on the benchmark-specific evaluation populations in Table 5. For trajectory n, the trajectory-level relative L_2 error is

$$\mathcal{E}^{(n)} = \frac{1}{T} \sum_{t=1}^{T} \frac{\|\hat{U}_t^{(n)} - U_t^{(n)}\|_2}{\|U_t^{(n)}\|_2 + \varepsilon}, \tag{24}$$

and the reported error is the mean of $\mathcal{E}^{(n)}$ over the evaluation trajectories.

For adaptive methods, computation is measured by the mean recurrent depth

$$\overline{K} = \frac{1}{NT} \sum_{n=1}^{N} \sum_{t=1}^{T} K_t^{(n)}. \tag{25}$$

Fixed-depth RecurrSim uses the same K for every physical transition, whereas AdaRecurr selects K_t online from the checkpoint grid $\{4, 8, 12, 16, 20, 24, 28, 32\}$.

Table 5: Accuracy–compute evaluation settings.

Dataset	Trajectories	Horizon	Reported fixed-K depths
Burgers	100	50	$\{4, 6, 8, 10, 12, 14, 16\}$
NS2D	800	13	$\{4, 8, 12, 16, 20, 24, 28, 32\}$
SWE2D	200	50	$\{4, 8, 12, 16, 20, 24, 28, 32\}$

Random Budget-Matched. For each target mean depth, the random baseline independently samples the two adjacent adaptive-grid depths that bracket the target. Their sampling probabilities are chosen so that the expected recurrent depth matches the target budget. We report the mean and standard deviation over five independently sampled schedules.

Heuristic Adaptive. At decision checkpoint k_j, the heuristic measures the relative change between consecutive latent checkpoints,

$$d_{t,j} = \frac{\|z_t^{(k_j)} - z_t^{(k_{j-1})}\|_2}{\|z_t^{(k_{j-1})}\|_2 + 10^{-12}}. \tag{26}$$

The heuristic stops when $d_{t,j} < \tau$ and otherwise continues to the next checkpoint, with stopping at $K = 32$. Candidate thresholds are obtained from empirical latent-change quantiles on a calibration subset, and for each target budget we use the threshold whose measured mean recurrent depth is closest to that target.

For the fixed-depth improvement percentages reported in Sec. 4.2, each AdaRecurr operating point is compared with the smallest reported fixed depth K satisfying $K \geq \overline{K}$.

A.6 Aggregate Shock-Strength Estimation

For each Burgers rollout state $u_t = u(\cdot, t)$, let M_t denote the number of detected shock fronts. Following the left–right state-jump characterization of an individual shock (Li et al., 2020a), the estimated strength of the m-th shock is

$$\hat{S}_{t,m} = |\hat{u}_{L,t,m} - \hat{u}_{R,t,m}|, \tag{27}$$

where $\hat{u}_{L,t,m}$ and $\hat{u}_{R,t,m}$ are estimated from the left and right states outside the corresponding shock transition layer.

A single physical state may contain multiple shock fronts. We therefore define the aggregate estimated shock strength of the complete spatial state as

$$\hat{S}_t = \sum_{m=1}^{M_t} \hat{S}_{t,m}. \tag{28}$$

On the discrete periodic grid, we identify distinct compressive fronts, determine the transition layer associated with each front, and estimate the corresponding left and right states outside that layer. Overlapping candidates associated with the same transition layer are merged before aggregation.

For the grouped analysis in Sec. 4.3, we use the controller's mean recurrent depth $\bar{K}$ to define

$$K_{\text{low}} = \{t : K_t < \bar{K}\}, \qquad K_{\text{high}} = \{t : K_t > \bar{K}\}. \tag{29}$$

Let V denote the set of states for which a valid aggregate shock-strength estimate is obtained, and define

$$V_{\text{low}} = K_{\text{low}} \cap V, \qquad V_{\text{high}} = K_{\text{high}} \cap V. \tag{30}$$

The reported group means are

$$\bar{S}_{\text{low}} = \frac{1}{|V_{\text{low}}|} \sum_{t \in V_{\text{low}}} \hat{S}_t, \qquad \bar{S}_{\text{high}} = \frac{1}{|V_{\text{high}}|} \sum_{t \in V_{\text{high}}} \hat{S}_t. \tag{31}$$

A.7 Hardware and Runtime Environment

All experiments are run on NVIDIA GPUs, primarily RTX 3090 and RTX 4090. Experiments are implemented with PyTorch 2.5.1 and CUDA 12.1, using single-precision floating-point computation for model training and evaluation. We report mean recurrent depth as the computation measure.

B Training and Inference Algorithms

The procedures below summarize the execution of AdaRecurr during controller training and adaptive inference.

Algorithm 1 Training AdaRecurr

Require: Frozen simulator f_θ; actor π_ϕ; critic V_ψ; reference trajectories $U_{0:T}$; depth grid $\mathcal{K} = \{k_0, \dots, k_J\}$; computation weight λ; reference cost C_{ref}

Initialize ϕ and ψ
repeat
  Initialize PPO buffer $\mathcal{B}$
  for each sampled trajectory $U_{0:T}$ **do**
    $\hat{U}_0 \leftarrow U_0$
    for $t = 0, \dots, T-1$ **do**
      Initialize the recurrent state for $\hat{U}_t$
      Execute k_0 recurrent iterations and obtain $z_t^{(k_0)}$
      $j \leftarrow 0$; $\widetilde{\mathcal{Z}}_t \leftarrow (z_t^{(k_0)}, \mathbf{0}, \dots, \mathbf{0})$
      while true **do**
        $o_{t,j} \leftarrow (\hat{U}_t, \widetilde{\mathcal{Z}}_t)$
        if $j = J$ **then**
          $a_{t,j} \leftarrow$ Stop
        else
          Sample $a_{t,j} \sim \pi_\phi(\cdot \mid o_{t,j})$
        end if
        if $a_{t,j} =$ Continue **then**
          Store $(o_{t,j}, a_{t,j}, -\lambda(k_{j+1} - k_j)/C_{\mathrm{ref}})$ in $\mathcal{B}$
          Continue recurrent refinement from $z_t^{(k_j)}$ to $z_t^{(k_{j+1})}$
          $j \leftarrow j + 1$; $\widetilde{\mathcal{Z}}_t[j] \leftarrow z_t^{(k_j)}$
        else
          $K_t \leftarrow k_j$; decode $\hat{U}_{t+1}$ from $z_t^{(K_t)}$
          Compute next-state relative L_2 error $\ell_t(K_t)$
          Store terminal transition $(o_{t,j}, a_{t,j}, -\ell_t(K_t))$ in $\mathcal{B}$
          break
        end if
      end while
    end for
  end for
  Compute within-transition returns and advantages from $\mathcal{B}$
  Update actor and critic parameters with PPO
until training terminates

Algorithm 2 AdaRecurr Inference

Require: Initial state U_0; frozen simulator f_θ; trained actor π_ϕ; depth grid $\mathcal{K} = \{k_0, \dots, k_J\}$; horizon T
Ensure: Predicted trajectory $\hat{U}_{1:T}$

$\hat{U}_0 \leftarrow U_0$
for $t = 0, \dots, T-1$ **do**
  Initialize the recurrent state for $\hat{U}_t$
  Execute k_0 recurrent iterations and obtain $z_t^{(k_0)}$
  $j \leftarrow 0$; $\widetilde{\mathcal{Z}}_t \leftarrow (z_t^{(k_0)}, \mathbf{0}, \dots, \mathbf{0})$
  while true **do**
    $o_{t,j} \leftarrow (\hat{U}_t, \widetilde{\mathcal{Z}}_t)$
    if $j = J$ **then**
      $a_{t,j} \leftarrow$ STOP
    else
      $a_{t,j} \leftarrow \arg\max_a \pi_\phi(a \mid o_{t,j})$
    end if
    if $a_{t,j} =$ CONTINUE **then**
      Continue recurrent refinement from $z_t^{(k_j)}$ to $z_t^{(k_{j+1})}$
      $j \leftarrow j + 1$; $\widetilde{\mathcal{Z}}_t[j] \leftarrow z_t^{(k_j)}$
    else
      $K_t \leftarrow k_j$; decode $\hat{U}_{t+1}$ from $z_t^{(K_t)}$
      break
    end if
  end while
end for
return $\hat{U}_{1:T}$

C Additional Analyses

C.1 Compute-Demand Heterogeneity Analysis

Section 4.4 studies whether adaptive recurrent-depth allocation becomes more useful when different physical states favor different amounts of recurrent computation. We construct two controlled NS2D settings, LOW and HIGH, while keeping the recurrent backbone, controller architecture, PPO configuration, computation penalty λ, and recurrent-depth grid

$$\mathcal{K} = \{4, 8, 12, 16, 20, 24, 28, 32\}$$

fixed.

To characterize state-wise compute demand, we first use a separate meta set. For each physical state in this meta set, the pretrained RecurrFNO backbone is kept frozen and evaluated independently at every recurrent depth $K \in \mathcal{K}$. Let

$$\ell_t(K) = \frac{\|\hat{U}_{t+1}^{(K)} - U_{t+1}\|_2}{\|U_{t+1}\|_2 + \varepsilon}$$

denote the next-state relative L_2 error obtained for state t at depth K. From this depth sweep, we assign each state a reference recurrent depth K_t^{req}, defined as the shallowest candidate depth that reaches the state's best-performing regime over the candidate depths. This reference depth is used only to characterize how much recurrent computation the state requires for the heterogeneity analysis; it is not provided to the AdaRecurr controller.

We then select states from the meta set according to their K_t^{req} labels to construct two populations with different amounts of compute-demand heterogeneity. In the LOW setting, 80% of selected states have $K_t^{\mathrm{req}} = 4$, while the remaining 20% have $K_t^{\mathrm{req}} \in \{8, 12\}$; no selected state has $K_t^{\mathrm{req}} > 12$. In the HIGH setting, 40% of selected states have $K_t^{\mathrm{req}} = 4$, approximately 20% have $K_t^{\mathrm{req}} \in \{8, 12\}$, and approximately 40% have $K_t^{\mathrm{req}} \geq 16$. Thus, the HIGH setting contains a substantially broader mixture of shallow- and deep-demand states.

Table 6: Grouped state-wise reference-depth distributions used to construct the controlled heterogeneity settings.

Setting	$K_t^{\mathrm{req}} = 4$	$K_t^{\mathrm{req}} \in \{8, 12\}$	$K_t^{\mathrm{req}} \geq 16$
LOW	80%	20%	0%
HIGH	40%	$\approx 20\%$	$\approx 40\%$

Table 7: Multi-seed AdaRecurr results corresponding to the operating points in Fig. 2. Each row aggregates five independent controller-training seeds.

Dataset	Mean Avg K	Trajectory Rel. L_2 (mean $\pm$ std.)
Burgers	5.425	0.08287 ± 0.002
Burgers	7.322	0.04191 ± 0.001
Burgers	8.922	0.02599 ± 0.004
NS2D	6.379	0.37343 ± 0.022
NS2D	8.854	0.33373 ± 0.027
NS2D	13.528	0.31128 ± 0.024
SWE2D	8.670	0.001595 ± 0.0001
SWE2D	13.482	0.001096 ± 0.0002
SWE2D	18.432	0.000930 ± 0.0002

The LOW and HIGH populations are deliberately constructed controlled subsets of the meta set. Their proportions are chosen to isolate the effect of state-wise compute-demand heterogeneity and are not intended to estimate the naturally occurring recurrent-depth distribution of NS2D.

For the Oracle Adaptive reference reported in Sec. 4.4, each evaluated state is assigned the candidate recurrent depth that optimizes the same accuracy–compute objective used for controller training. This oracle has access to the target next state when selecting its depth and therefore represents an offline reference for the opportunity available from ideal state-wise allocation, not a deployable inference method. AdaRecurr, by contrast, selects its depth online from the current physical state and the recurrent refinement trace available at inference.

C.2 MULTI-SEED CONTROLLER RESULTS

Each AdaRecurr operating point in Fig. 2 reports the mean over five independent controller-training seeds. Table 7 reports the corresponding mean recurrent depth and the mean $\pm$ standard deviation of trajectory-level relative L_2 error across the five runs.