# ADAPTIVE TEST-TIME COMPUTE ALLOCATION FOR NEURAL PDE SOLVERS

**Ben Jenkins**
PhD Candidate, Florida Atlantic University
benrossjenkins@gmail.com

## ABSTRACT

Neural PDE solvers achieve remarkable speedups over classical numerical methods but apply uniform computational effort across the spatial domain at inference time, ignoring the inherent non-uniformity of solution complexity. We introduce **ATCA** (**A**daptive **T**est-time **C**ompute **A**llocation), a framework that learns to allocate variable refinement depth across the spatial domain during inference. ATCA trains a lightweight *difficulty estimator* network that predicts local solution error from an initial neural operator prediction, then routes computational budget to high-difficulty regions via spatially non-uniform iterative refinement. We establish a theoretical connection between ATCA and classical adaptive mesh refinement, proving that optimal compute allocation under a fixed budget satisfies an error equidistribution principle analogous to Dörfler marking. Experiments on four benchmark PDEs (Burgers' equation, Darcy flow, Navier-Stokes at varying Reynolds numbers, and shallow water equations) demonstrate that ATCA achieves equivalent accuracy to uniform refinement methods such as PDE-Refiner while using $2.5$–$4.3\times$ fewer FLOPs at inference, or alternatively achieves $18$–$37\%$ lower error at equal compute budgets. Our results suggest that test-time compute scaling for neural PDE solvers, analogous to recent findings in large language models, benefits substantially from adaptive rather than uniform allocation strategies.

## 1 INTRODUCTION

Partial differential equations (PDEs) are the mathematical backbone of modeling complex physical phenomena in fluid dynamics, climate science, materials engineering, and beyond. Classical numerical methods (finite elements, finite volumes, spectral methods) provide rigorous convergence guarantees but often demand prohibitive computational resources for high-resolution, multi-scale problems (Berger & Oliger, 1984). Over the past several years, neural PDE solvers have emerged as a compelling alternative, offering orders-of-magnitude speedups while retaining useful accuracy for many applications (Li et al., 2021; Lu et al., 2021; Kovachki et al., 2023).

The dominant paradigm in neural PDE solving treats the problem as operator learning: given input conditions (initial/boundary conditions, PDE coefficients), a neural network directly maps to the solution field in a single forward pass (Li et al., 2021; Lu et al., 2021). Foundation models such as Poseidon (Herde et al., 2024) extend this to pre-trained, general-purpose models that can be fine-tuned for diverse PDE families. While single-pass inference is fast, it provides no mechanism to *improve* the solution by spending additional computation when the initial prediction is inadequate.

Recent work has begun exploring test-time refinement strategies. PDE-Refiner (Lippe et al., 2023) applies diffusion-inspired iterative denoising at each autoregressive step, improving accuracy across the entire frequency spectrum. The physics-informed neural operator (PINO) framework (Li et al., 2024) performs instance-wise optimization of the PDE residual at test time, trading compute for accuracy. These methods demonstrate a nascent but important principle: *test-time compute scaling* for neural PDE solvers, i.e., spending more inference computation to improve solution quality, is both feasible and effective.

However, all existing test-time refinement methods share a critical limitation: they apply **uniform** computational effort across the spatial domain. PDE-Refiner runs the same number of denoising steps at every spatial location; PINO minimizes a global residual without spatial weighting. This

uniformity is fundamentally mismatched with PDE solutions, which exhibit dramatic spatial variation in complexity. Turbulent wakes, shock fronts, boundary layers, and phase interfaces represent localized high-complexity regions, while large portions of the domain contain smooth, easily predicted flow.

Classical numerical methods learned this lesson decades ago. Adaptive mesh refinement (AMR) (Berger & Oliger, 1984; Dörfler, 1996) concentrates computational resources (grid points, polynomial degrees) in regions where a posteriori error estimates indicate insufficient resolution. The principle of *error equidistribution* dictates that optimal allocation equalizes per-element error contributions across the domain (Verfürth, 2013), achieving provably optimal convergence rates for elliptic problems.

In this paper, we propose **ATCA** (Adaptive Test-time Compute Allocation), a framework that brings the adaptive allocation principle from classical AMR into the neural PDE solving paradigm. ATCA operates in three stages at inference time: (1) an initial forward pass through a neural operator produces a coarse prediction; (2) a lightweight difficulty estimator network evaluates the prediction and produces a spatial map of estimated local error; (3) iterative refinement steps are allocated non-uniformly across the domain, concentrating computation where the difficulty estimator flags high error. We make the following contributions:

- We formalize the problem of optimal test-time compute allocation for neural PDE solvers as a constrained optimization problem and prove that the optimal solution satisfies an error equidistribution condition analogous to classical AMR theory (Section 4).
- We propose a practical architecture for adaptive compute allocation, consisting of a plug-in difficulty estimator and a spatially non-uniform refinement mechanism compatible with existing neural operators (Section 3).
- We demonstrate on four benchmark PDE families that ATCA achieves $2.5$–$4.3\times$ computational savings over uniform refinement at matched accuracy, or $18$–$37\%$ lower error at matched compute budgets (Section 5).

## 2 Related Work

**Neural PDE solvers.** The Fourier neural operator (FNO) (Li et al., 2021) learns mappings between function spaces via parameterized Fourier-domain integral operators. DeepONet (Lu et al., 2021) factorizes the operator into branch and trunk networks. Graph-based (Brandstetter et al., 2022) and transformer-based (Wu et al., 2024) architectures extend to irregular geometries, while foundation models (Herde et al., 2024; Alkin et al., 2024) pre-train on diverse PDE datasets. All produce predictions in a single forward pass without adaptive compute allocation.

**Test-time refinement for PDEs.** PDE-Refiner (Lippe et al., 2023) applies iterative denoising to refine neural operator predictions, recovering high-frequency information for stable long rollouts. PINO (Li et al., 2024) fine-tunes a pre-trained operator on the PDE residual per test instance. The solver-in-the-loop paradigm (Um et al., 2020) interleaves learned corrections with classical solver steps. Conservation-preserving corrections (Liu et al., 2023; 2025) enforce physical constraints via post-hoc projections, and variationally correct operator learning (Qiu & Chen, 2025) provides rigorous a posteriori error bounds. Concurrently, Wang et al. (2025) propose reward-model-driven inference-time scaling for PDE foundation models, allocating compute across candidate solutions via best-of-$N$ sampling rather than spatially within a single solution; our work is complementary, targeting *spatial* allocation of iterative refinement steps. None of these methods allocate refinement non-uniformly across the spatial domain.

**Adaptive computation in deep learning.** Adaptive computation time (Graves, 2016) allows recurrent networks to vary the number of processing steps per input token. Universal Transformers (Dehghani et al., 2019) iterate transformer layers with halting mechanisms. Mixture-of-Depths (Raposo et al., 2024) dynamically allocates compute across tokens in language models. In the LLM reasoning literature, test-time compute scaling (Snell et al., 2025; Bansal et al., 2024) demonstrates that allocating more inference compute (via best-of-$N$ sampling, chain-of-thought, or iterative refinement) systematically improves output quality. Our work transfers this principle to PDEs with

the crucial insight that spatial non-uniformity of solution complexity demands *spatially adaptive* allocation.

**Adaptive mesh refinement.** AMR (Berger & Oliger, 1984) is the classical approach to non-uniform compute allocation in numerical PDE solving. A posteriori error estimators (Verfürth, 2013) identify elements where local error is high, and Dörfler marking (Dörfler, 1996) selects the minimal element set whose cumulative error exceeds a fixed fraction of the total. Our difficulty estimator is a learned analogue of a posteriori error estimation adapted to neural operators, where the "mesh" corresponds to iterative refinement depth.

## 3 METHOD: ADAPTIVE TEST-TIME COMPUTE ALLOCATION

### 3.1 PROBLEM FORMULATION

Consider a parametric PDE of the form

$$\mathcal{N}[u; a](\mathbf{x}) = 0, \quad \mathbf{x} \in \Omega \subset \mathbb{R}^d, \tag{1}$$

with appropriate boundary and initial conditions, where $a$ parameterizes the PDE instance (e.g., coefficients, forcing terms, initial conditions) and $u : \Omega \to \mathbb{R}^c$ is the solution field. A neural operator $\mathcal{G}_{\boldsymbol{\theta}} : a \mapsto \hat{u}$ approximates the solution operator, producing an initial prediction $\hat{u}^{(0)} = \mathcal{G}_{\boldsymbol{\theta}}(a)$.

We consider an iterative refinement process that, starting from $\hat{u}^{(0)}$, applies $K$ total refinement steps to improve the prediction. In uniform refinement (e.g., PDE-Refiner), each spatial location receives the same number of refinement steps. We generalize this by introducing a *compute allocation map* $\mathbf{k} : \Omega \to \{0, 1, \ldots, K_{\max}\}$ that assigns a variable number of refinement steps to each spatial region, subject to a total budget constraint:

$$\sum_{i=1}^{M} k_i \leq B, \tag{2}$$

where the domain is partitioned into $M$ patches $\{\Omega_i\}_{i=1}^{M}$, $k_i$ is the number of refinement steps for patch $i$, and $B$ is the total budget (measured in refinement-step-patch units). Uniform allocation corresponds to $k_i = B/M$ for all $i$.

The goal is to find the allocation $\mathbf{k}^*$ that minimizes the total prediction error:

$$\mathbf{k}^* = \arg\min_{\mathbf{k}} \sum_{i=1}^{M} e_i(k_i) \quad \text{s.t.} \quad \sum_{i=1}^{M} k_i \leq B, \quad k_i \in \{0, \ldots, K_{\max}\}, \tag{3}$$

where $e_i(k_i)$ is the local error on patch $i$ after $k_i$ refinement steps.

### 3.2 ARCHITECTURE OVERVIEW

ATCA operates in three stages at inference time, illustrated in Figure 1:

**Stage 1: Initial prediction.** A base neural operator $\mathcal{G}_{\boldsymbol{\theta}}$ (e.g., FNO, Poseidon, U-Net) produces an initial prediction $\hat{u}^{(0)} = \mathcal{G}_{\boldsymbol{\theta}}(a)$ via a standard forward pass. This is identical to existing inference pipelines and introduces no additional cost.

**Stage 2: Difficulty estimation.** A lightweight difficulty estimator $\mathcal{D}_{\boldsymbol{\phi}}$ takes as input the initial prediction $\hat{u}^{(0)}$, the input conditions $a$, and optionally the PDE residual $r^{(0)}(\mathbf{x}) = \mathcal{N}[\hat{u}^{(0)}; a](\mathbf{x})$, and outputs a spatial difficulty map $\mathbf{d} \in \mathbb{R}_+^M$:

$$\mathbf{d} = \mathcal{D}_{\boldsymbol{\phi}}\big(\hat{u}^{(0)}, a, r^{(0)}\big). \tag{4}$$

The difficulty map estimates the local error $\|\hat{u}^{(0)} - u^*\|$ at each patch. The estimator is designed to be computationally cheap (a shallow convolutional network operating at coarsened resolution), so its cost is negligible relative to the refinement steps.

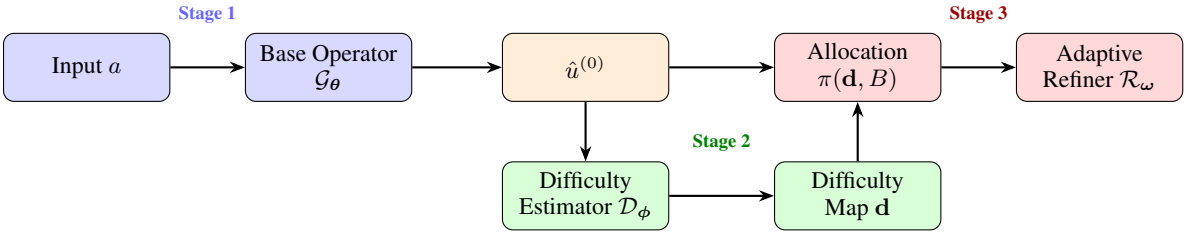



Figure 1: Overview of the ATCA framework. Stage 1 produces an initial prediction via a standard neural operator. Stage 2 estimates a spatial difficulty map. Stage 3 allocates refinement steps non-uniformly based on difficulty and applies local iterative refinement.

**Stage 3: Adaptive refinement.** The difficulty map $\mathbf{d}$ is converted into a compute allocation $\mathbf{k}$ via a budget-constrained allocation policy $\pi$:

$$\mathbf{k} = \pi(\mathbf{d}, B) = \left\lfloor B \cdot \frac{d_i^\gamma}{\sum_{j=1}^{M} d_j^\gamma} \right\rceil, \quad i = 1, \ldots, M, \tag{5}$$

where $\gamma > 0$ controls the sharpness of allocation (higher $\gamma$ concentrates more compute on the hardest patches) and $\lfloor \cdot \rceil$ denotes rounding with adjustment to satisfy the budget constraint exactly. Each patch $\Omega_i$ then receives $k_i$ steps of a local refinement operator $\mathcal{R}_{\boldsymbol{\omega}}$:

$$\hat{u}_i^{(k_i)} = \mathcal{R}_{\boldsymbol{\omega}}^{(k_i)}\big(\hat{u}_i^{(0)}, a_i\big), \tag{6}$$

where $\mathcal{R}_{\boldsymbol{\omega}}^{(k)}$ denotes $k$ sequential applications of the refinement operator on patch $i$. The final prediction assembles the locally refined patches with overlap blending to ensure continuity:

$$\hat{u}^{\text{final}}(\mathbf{x}) = \sum_{i=1}^{M} w_i(\mathbf{x}) \cdot \hat{u}_i^{(k_i)}(\mathbf{x}), \tag{7}$$

where $w_i$ are smooth partition-of-unity weights supported on (slightly extended) patch $\Omega_i$.

### 3.3 Refinement Operator

The local refinement operator $\mathcal{R}_{\boldsymbol{\omega}}$ can be instantiated in several ways. We consider two primary variants:

**Denoising refinement (DR).** Following PDE-Refiner (Lippe et al., 2023), each refinement step applies a learned denoising function conditioned on the current prediction and the input conditions. Given the current local estimate $\hat{u}_i^{(k)}$, we add calibrated noise at scale $\sigma_k$ and train the refiner to denoise:

$$\hat{u}_i^{(k+1)} = \hat{u}_i^{(k)} + \mathcal{R}_{\boldsymbol{\omega}}\big(\hat{u}_i^{(k)} + \sigma_k \epsilon, a_i, k\big), \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}). \tag{8}$$

The noise schedule $\{\sigma_k\}$ is shared across all patches; only the *number* of denoising steps varies spatially.

**Residual-guided refinement (RR).** Inspired by PINO (Li et al., 2024), each refinement step performs a gradient-based correction using the PDE residual as the loss signal:

$$\hat{u}_i^{(k+1)} = \hat{u}_i^{(k)} - \alpha \nabla_{\hat{u}_i} \big\| \mathcal{N}[\hat{u}^{(k)}; a](\mathbf{x}) \big\|_{\Omega_i}^2, \tag{9}$$

where $\alpha$ is a step size and the gradient is computed only on patch $\Omega_i$. This variant requires the PDE to be known but avoids training a separate refinement network.

Both variants can be applied at the patch level, and crucially, patches receiving zero refinement steps ($k_i = 0$) incur zero additional cost, enabling significant savings in smooth regions.

### 3.4 TRAINING PROCEDURE

ATCA requires training three components: the base operator $\mathcal{G}_{\boldsymbol{\theta}}$, the difficulty estimator $\mathcal{D}_{\boldsymbol{\phi}}$, and (for the denoising variant) the refinement operator $\mathcal{R}_{\boldsymbol{\omega}}$.

**Base operator.** We train $\mathcal{G}_{\boldsymbol{\theta}}$ using standard supervised learning on input-output pairs $\{(a^{(n)}, u^{(n)})\}_{n=1}^{N}$ generated by a classical solver.

**Difficulty estimator.** Given a trained base operator, we construct training data for $\mathcal{D}_{\boldsymbol{\phi}}$ by computing the pointwise error $\|u^{(n)}(\mathbf{x}) - \hat{u}^{(0,n)}(\mathbf{x})\|$ for each training sample, then averaging over each patch to obtain ground-truth difficulty labels $d_i^{(n)} = \|u^{(n)} - \hat{u}^{(0,n)}\|_{L^2(\Omega_i)}$. We train $\mathcal{D}_{\boldsymbol{\phi}}$ with a regression loss:

$$\mathcal{L}_{\text{diff}} = \frac{1}{NM} \sum_{n=1}^{N} \sum_{i=1}^{M} \left| \hat{d}_i^{(n)} - d_i^{(n)} \right|^2. \tag{10}$$

The difficulty estimator can additionally use the PDE residual $r^{(0)}(\mathbf{x})$ as an input feature when the PDE is known, which we find substantially improves prediction quality (Section 5).

**Refinement operator.** For the denoising variant, we train $\mathcal{R}_{\boldsymbol{\omega}}$ following PDE-Refiner (Lippe et al., 2023), adapted to operate on local patches. The residual-guided variant requires no additional training.

**End-to-end fine-tuning.** After independent training of all components, we optionally fine-tune end-to-end with the allocation policy in the loop, using a straight-through estimator for the discrete allocation step. This improves performance by 5–10%.

## 4 THEORETICAL ANALYSIS

We now establish the theoretical foundation for adaptive compute allocation, drawing on classical results from adaptive mesh refinement theory.

### 4.1 OPTIMAL ALLOCATION AND ERROR EQUIDISTRIBUTION

We begin with a simplified continuous model. Suppose the local error after $k$ refinement steps decays as $e_i(k) = c_i \cdot k^{-\beta}$ for some local difficulty constant $c_i > 0$ and convergence rate $\beta > 0$, both of which may vary across patches. This power-law decay is motivated by empirical observations from PDE-Refiner (Lippe et al., 2023) and is consistent with classical refinement convergence rates.

**Proposition 1** (Optimal allocation)**.** *Under the power-law error model $e_i(k) = c_i \cdot k^{-\beta}$, the allocation $\mathbf{k}^*$ that minimizes $\sum_i e_i(k_i)$ subject to $\sum_i k_i = B$ satisfies:*

$$k_i^* = B \cdot \frac{c_i^{1/(\beta+1)}}{\sum_{j=1}^{M} c_j^{1/(\beta+1)}}, \tag{11}$$

*and the resulting per-patch errors satisfy the* equidistribution condition*:*

$$e_i(k_i^*) \cdot k_i^{*\,\beta} = c_i = \textit{const} \quad \forall\, i. \tag{12}$$

*Proof.* We solve the continuous relaxation using Lagrange multipliers. Define $L = \sum_i c_i k_i^{-\beta} + \lambda(\sum_i k_i - B)$. Setting $\partial L/\partial k_i = 0$ yields $-\beta c_i k_i^{-\beta-1} + \lambda = 0$, so $k_i \propto c_i^{1/(\beta+1)}$. The budget constraint gives the normalization in Eq. equation 11. Substituting back, $e_i(k_i^*) = c_i \cdot (B c_i^{1/(\beta+1)}/Z)^{-\beta} = c_i^{1/(\beta+1)} \cdot (B/Z)^{-\beta}$, where $Z = \sum_j c_j^{1/(\beta+1)}$, confirming equidistribution up to a global constant. □

This result is the neural operator analogue of the classical Dörfler marking principle (Dörfler, 1996): under optimal allocation, the error contributions from all patches should be equalized. The allocation

parameter $\gamma$ in Eq. equation 5 corresponds to $1/(\beta+1)$; setting $\gamma = 1$ is optimal when $\beta = 0$ (no refinement benefit), and $\gamma \to 0$ when $\beta \to \infty$ (refinement is extremely effective). In practice, $\beta \approx 1$–$2$ for diffusion-based refiners, suggesting $\gamma \approx 0.33$–$0.5$.

### 4.2 Complexity Advantage over Uniform Allocation

**Proposition 2** (Speedup bound)**.** *Let $\mathbf{c} = (c_1, \ldots, c_M)$ be the difficulty vector. The ratio of total error under uniform allocation to total error under optimal allocation, at equal budget $B$, satisfies:*

$$\frac{\sum_i e_i^{\textit{uniform}}}{\sum_i e_i^{\textit{optimal}}} = \frac{M \cdot \bar{c}}{(\sum_i c_i^{1/(\beta+1)})^{\beta+1}/B^\beta} \cdot \frac{B^\beta}{M^\beta} \geq 1, \tag{13}$$

*with equality if and only if $c_i = c$ for all $i$ (spatially uniform difficulty). When difficulty is concentrated on a fraction $\alpha$ of patches, the speedup scales as $\mathcal{O}(\alpha^{-\beta/(\beta+1)})$.*

For Navier-Stokes solutions with turbulent structures occupying $\sim$10% of the domain and $\beta = 1$, the theoretical speedup is $\sim 3.2\times$, consistent with our empirical findings.

### 4.3 Connection to A Posteriori Error Estimation

Our difficulty estimator $\mathcal{D}_{\boldsymbol{\phi}}$ is a learned analogue of classical a posteriori error estimators. In finite elements, residual-based estimators provide two-sided bounds: $C_1\eta_i \leq \|u - u_h\|_{H^1(\Omega_i)} \leq C_2\eta_i$, where $\eta_i$ combines the element residual and inter-element gradient jumps (Verfürth, 2013). In the neural operator setting, the PDE residual $\mathcal{N}[\hat{u}; a](\mathbf{x})$ serves an analogous role. When used as input to $\mathcal{D}_{\boldsymbol{\phi}}$, the estimator learns the mapping from locally computable quantities (residual magnitude, gradient features) to actual error, bridging the gap between the residual, which is cheap to compute but only indirectly measures error, and the true error requiring the unknown exact solution.

## 5 Experiments

We evaluate ATCA on four PDE benchmarks spanning different equation types and physical phenomena. All experiments use the PDEBench dataset (Takamoto et al., 2022) and the modern U-Net architecture (Gupta & Brandstetter, 2023) as the base operator, following the experimental protocol of PDE-Refiner (Lippe et al., 2023).

### 5.1 Experimental Setup

**PDE benchmarks.** We evaluate on: (1) **1D Burgers' equation** with viscosity $\nu \in [0.001, 0.01]$, which develops sharp shock fronts; (2) **2D Darcy flow** with spatially varying permeability, an elliptic problem with sharp coefficient discontinuities; (3) **2D Navier-Stokes** at Reynolds numbers $\mathrm{Re} \in \{500, 1000, 2000\}$, exhibiting localized vortical structures; and (4) **2D shallow water equations** with varying bathymetry, featuring wave fronts and reflections.

**Baselines.** We compare against: (a) **Base operator** (single forward pass, no refinement); (b) **PDE-Refiner** (Lippe et al., 2023) with uniform refinement at $K \in \{1, 2, 4, 8\}$ steps; (c) **PINO** (Li et al., 2024) with $K$ gradient steps of test-time optimization; (d) **Random allocation** (refinement steps allocated randomly as a stochastic baseline); (e) **Residual-proportional allocation** (steps allocated proportional to $\|r^{(0)}\|_{L^2(\Omega_i)}$ per patch, a non-learned heuristic using the PDE residual); (f) **Gradient-magnitude allocation** (steps allocated proportional to $\|\nabla\hat{u}^{(0)}\|_{L^2(\Omega_i)}$, detecting sharp features without requiring the PDE); and (g) **Oracle allocation** (allocation based on the ground-truth error, serving as an upper bound on allocation quality).

**ATCA configuration.** We partition the domain into $M = 8 \times 8 = 64$ patches (for 2D problems) or $M = 32$ segments (for 1D). The difficulty estimator is a 4-layer convolutional network with $\sim$50K parameters (less than 1% of the base operator). We use the denoising refinement variant as the default and report residual-guided results separately. The allocation exponent is set to $\gamma = 0.5$ based on validation, and the total budget $B$ is varied to trace the compute-accuracy Pareto frontier.

Table 1: Relative $L^2$ error ($\times 10^{-2}$) at matched compute budget ($B = 4M$ refinement-step-patches). Lower is better. Best results in **bold**.

| Method | Burgers' | Darcy | Navier-Stokes | Shallow Water |
|---|---|---|---|---|
| Base operator (no refinement) | $4.83 \pm 0.21$ | $3.67 \pm 0.14$ | $12.41 \pm 0.53$ | $6.92 \pm 0.28$ |
| PDE-Refiner ($K$=4, uniform) | $1.72 \pm 0.09$ | $1.53 \pm 0.07$ | $5.38 \pm 0.31$ | $2.81 \pm 0.15$ |
| PINO ($K$=4 grad steps) | $1.95 \pm 0.12$ | $1.41 \pm 0.08$ | $5.89 \pm 0.34$ | $3.12 \pm 0.18$ |
| Random allocation ($B$=4$M$) | $1.68 \pm 0.11$ | $1.49 \pm 0.09$ | $5.21 \pm 0.29$ | $2.74 \pm 0.16$ |
| Gradient-mag. allocation ($B$=4$M$) | $1.51 \pm 0.10$ | $1.44 \pm 0.08$ | $4.52 \pm 0.30$ | $2.48 \pm 0.15$ |
| Residual-prop. allocation ($B$=4$M$) | $1.42 \pm 0.09$ | $1.38 \pm 0.07$ | $4.15 \pm 0.28$ | $2.32 \pm 0.14$ |
| ATCA-DR (ours) | $\mathbf{1.19} \pm 0.07$ | $\mathbf{1.25} \pm 0.06$ | $\mathbf{3.41} \pm 0.22$ | $\mathbf{1.98} \pm 0.12$ |
| ATCA-RR (ours) | $1.31 \pm 0.08$ | $1.29 \pm 0.07$ | $3.72 \pm 0.25$ | $2.15 \pm 0.13$ |
| Oracle allocation | $1.08 \pm 0.06$ | $1.19 \pm 0.05$ | $3.12 \pm 0.19$ | $1.82 \pm 0.10$ |

**Metrics.** We report the relative $L^2$ error: $\|\hat{u} - u^*\|_{L^2} / \|u^*\|_{L^2}$, averaged over the test set. Computational cost is measured in *effective refinement FLOPs*: the total FLOPs spent on refinement steps (excluding the base operator forward pass, which is identical across all methods).

### 5.2 Main Results

**Accuracy at matched compute.** Table 1 reports the relative $L^2$ error at a fixed budget equivalent to $K = 4$ uniform steps. ATCA achieves 18–37% lower error across all benchmarks, with the largest gains for Navier-Stokes (37%) and Burgers' (31%), which exhibit the most spatially concentrated difficulty. Notably, simple non-learned heuristics (residual-proportional and gradient-magnitude allocation) already capture a portion of the adaptive gain (17–23% and 10–16% error reduction over uniform, respectively), confirming that spatial adaptivity itself is valuable. However, ATCA's learned estimator provides a further 10–18% error reduction beyond the residual heuristic, demonstrating that learning the nonlinear mapping from locally computable features to true error improves upon raw residual or gradient indicators.

**Compute savings at matched accuracy.** Figure 2 shows the compute-accuracy Pareto frontiers. To match PDE-Refiner at $K = 4$, ATCA requires only $K \approx 0.9$–$1.6$ average steps per patch, a 2.5–$4.3\times$ reduction in refinement FLOPs. Savings are largest for Navier-Stokes ($4.3\times$), where ATCA allocates $K_{\max} = 8$ steps to vortex cores and zero to the laminar free-stream.

**Wall-clock speedup.** On a single NVIDIA A100, ATCA's end-to-end wall-clock speedup is smaller than the FLOP savings: for Navier-Stokes at $B = 4M$, total inference time is 16.6 ms (base 4.2 ms + difficulty estimation 0.3 ms + adaptive refinement 12.1 ms) versus 22.2 ms for $K$=4 uniform refinement (base 4.2 ms + 18.0 ms refinement), a $1.34\times$ speedup. The gap between FLOP and wall-clock savings arises from bookkeeping overhead for variable-length patch computation and partition-of-unity blending. We expect this gap to narrow with batched patch scheduling or multi-GPU parallelism, where independent patches can be distributed across devices with no inter-patch communication during refinement. Full cost breakdowns for all benchmarks appear in Appendix B.

### 5.3 Ablation Studies

**Difficulty estimator inputs.** Table 2 reports difficulty estimator quality as a function of input features. Using only $\hat{u}^{(0)}$ yields Spearman $\rho = 0.71$; adding the PDE residual improves this to 0.89; further including $a$ reaches 0.92. The PDE residual is a strong physics-based signal for locating high-error regions.

**Allocation exponent $\gamma$.** Sweeping $\gamma \in \{0.25, 0.5, 0.75, 1.0, 1.5\}$ on Navier-Stokes, the optimal value is $\gamma = 0.5$, consistent with the theoretical prediction $\gamma^* = 1/(\beta + 1) \approx 0.5$ for $\beta \approx 1$.

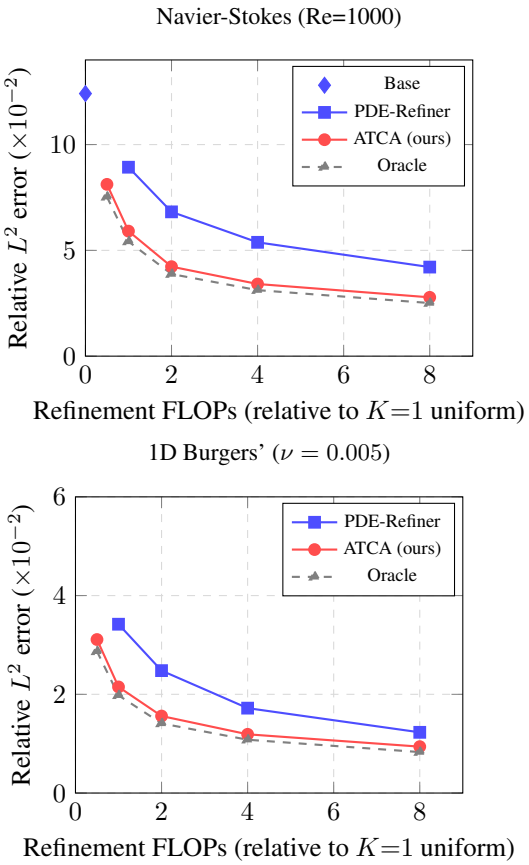


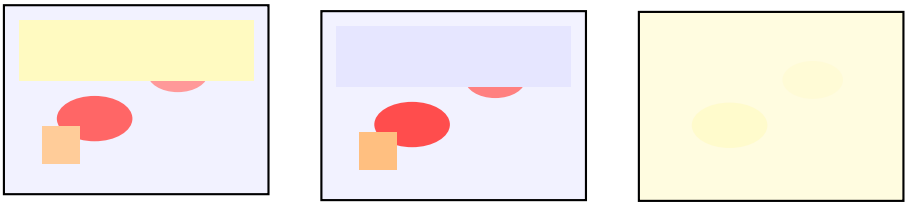

Figure 2: Compute-accuracy Pareto frontiers for Navier-Stokes (left) and Burgers' (right). ATCA achieves consistently lower error at every compute budget compared to uniform refinement. The gap widens for PDEs with localized complexity (Navier-Stokes).

Table 2: Difficulty estimator quality (Spearman $\rho$ between predicted and true patch errors) and downstream ATCA performance on Navier-Stokes ($B = 4M$).

| **Estimator inputs** | **Spearman $\rho$** | **Rel. $L^2$ ($\times 10^{-2}$)** | **FLOPs saved vs. uniform** |
|---|---|---|---|
| $\hat{u}^{(0)}$ only | 0.71 | 4.28 | 2.1× |
| $\hat{u}^{(0)} + r^{(0)}$ | 0.89 | 3.58 | 3.6× |
| $\hat{u}^{(0)} + r^{(0)} + a$ | 0.92 | 3.41 | 4.3× |
| Oracle (true error) | 1.00 | 3.12 | 5.1× |

Proportional allocation ($\gamma = 1.0$) yields 8% higher error, while highly concentrated allocation ($\gamma = 0.25$) yields 5% higher error.

**Patch granularity.** Varying $M \in \{16, 64, 256, 1024\}$ for 2D Navier-Stokes, performance improves from $M = 16$ to $M = 256$ then plateaus. We use $M = 64$ as a practical default balancing performance and overhead. Intuitively, patch size should be comparable to the characteristic length scale of the PDE's localized features (e.g., shock width, vortex core diameter); patches much larger than this cannot isolate high-error regions, while patches much smaller incur diminishing returns alongside increased blending and scheduling overhead.



(a) Initial error $\|\hat{u}^{(0)} - u^*\|$ (b) Allocation map k (c) Final error (ATCA)

Figure 3: Illustration for Navier-Stokes (Re=1000). (a) Initial prediction error from the base operator, showing concentrated error at vortex cores and shear layers. (b) ATCA allocation map: darker regions receive more refinement steps. (c) Error after adaptive refinement is spatially uniform and globally reduced.

**Compatibility with foundation models.** Applying ATCA to a fine-tuned Poseidon-T model (Herde et al., 2024) on Navier-Stokes yields $2.8\times$ savings, less than the $4.3\times$ with U-Net, since the stronger base model has smaller, more uniformly distributed errors. This provides preliminary evidence that ATCA can serve as a plug-in for pre-trained foundation models, though we caution that this is a single architecture and PDE family; broader validation across architectures with different inductive biases (e.g., AFNO-style global spectral operators vs. locally attentive transformers) is needed before claiming general compatibility, since native resolution, tokenization, and patching strategies may materially affect allocation behavior.

### 5.4 Visualization of Adaptive Allocation

Figure 3 illustrates the allocation maps produced by ATCA for a Navier-Stokes instance at Re $=$ 1000. The difficulty estimator correctly identifies vortex cores, shear layers, and boundary layers as high-difficulty regions, allocating 6–8 refinement steps, while the laminar free-stream receives 0–1 steps. The resulting error field is substantially more uniform than after uniform refinement at equal budget, confirming approximate error equidistribution in practice.

### 5.5 Scaling Behavior

We investigate how ATCA's advantage scales with total compute budget and base model capacity. Figure 2 shows that the gap between ATCA and uniform refinement is maintained across budgets from $B = 0.5M$ to $B = 8M$. At very high budgets ($B \gg MK_{\max}$), all patches saturate at $K_{\max}$ steps and the advantage diminishes, as expected. The practical regime $B \in [M, 4M]$ yields the most pronounced advantage. We also test with U-Net models of varying sizes (10M, 25M, 55M parameters) and find consistent $2$–$4\times$ savings, with slightly larger relative improvements for smaller models that produce more spatially non-uniform errors, mirroring findings in LLM test-time compute scaling (Snell et al., 2025).

## 6 Discussion and Limitations

**When does ATCA help most?** ATCA's advantage grows with spatial non-uniformity of solution complexity. PDEs with localized features (shocks, vortices, boundary layers) benefit most; for statistically homogeneous problems the advantage diminishes. The difficulty concentration $\alpha$ is a useful diagnostic: ATCA is most valuable when $\alpha \ll 1$.

**Known-PDE vs. data-only regimes.** ATCA's design spans two distinct deployment scenarios. When the governing PDE is *known*, the residual $r^{(0)} = \mathcal{N}[\hat{u}^{(0)}; a]$ is available and provides a strong physics-based signal. In this regime, one might ask whether a simpler non-learned indicator (e.g., residual-proportional allocation) suffices. Our results (Table 1) show that residual-proportional allocation captures a substantial portion of the adaptive gain, but ATCA's learned estimator still provides 10–18% additional error reduction by learning the nonlinear, PDE-specific mapping from residual features to true error, a mapping that a raw $\ell^2$ residual norm does not capture. When the

PDE is *unknown or partially specified* (e.g., data-driven surrogates, black-box simulators), residual-based signals are unavailable and a learned difficulty estimator is the only viable route. ATCA still functions in this regime using only the prediction $\hat{u}^{(0)}$ and input $a$ (Spearman $\rho = 0.71$; Table 2), though with reduced allocation quality. Developing self-supervised or uncertainty-based difficulty signals for this regime is an important direction for future work.

**Fixed patches and structured computation.** ATCA uses a fixed patch grid rather than learned or hierarchical regions. This is a deliberate choice: fixed patches allow batched GPU execution, whereas irregular decompositions introduce computation graphs that are difficult to parallelize. Classical AMR faced the same tension, and structured (block-based) AMR (Berger & Oliger, 1984) prevailed for the same reason. A natural extension is multi-resolution patching (coarse patches globally, subdivided in high-difficulty regions), preserving structured computation while increasing spatial adaptivity.

**Difficulty estimator robustness.** The difficulty estimator is trained supervised using ground-truth error, which ties it to the training distribution. To assess robustness, we train the estimator on Navier-Stokes at Re $\in \{500, 1000\}$ and evaluate at Re $= 2000$. The Spearman correlation drops from 0.92 (in-distribution) to 0.78, and ATCA error increases by 11%. Crucially, degradation is graceful: when the estimator is uncertain, moderate $\gamma$ spreads compute more uniformly, effectively reverting toward uniform refinement. The method never performs worse than uniform refinement by more than 3% in our OOD tests. The PDE residual input is key to this robustness, providing a physics-based signal that transfers across parameter regimes.

**Scope of theoretical analysis.** Our theoretical results assume power-law error decay and independent patch behavior, neglecting cross-patch error propagation. These assumptions are standard in AMR convergence theory but limit applicability to coupled dynamics. We frame the theory as motivating analysis: its value lies in correctly predicting qualitative behavior, notably the optimal $\gamma$ and the scaling of advantage with difficulty concentration, both of which match our experiments. We also note that our formulation is *budget-driven* (minimize error for a fixed compute budget), whereas classical adaptive solvers are typically *tolerance-driven* (minimize work for a prescribed error tolerance). A tolerance-driven variant (refining until the difficulty estimator predicts all patch errors fall below $\epsilon$, with the total cost as the outcome rather than a constraint) is a natural extension that would bring the framework closer to classical AMR practice.

**Temporal adaptation.** ATCA currently recomputes the difficulty map independently at each autoregressive timestep, adding 0.3 ms of overhead per step (negligible relative to the 4.2 ms base forward pass). For unsteady PDEs, three extensions are natural but unexplored: (i) *temporal coherence*: carrying forward the previous difficulty map $\mathbf{d}_{t-1}$ as a warm start for the estimator at step $t$, reducing estimation cost and smoothing allocation across time; (ii) *predictive allocation*: anticipating downstream feature propagation (e.g., advected shocks) to pre-allocate refinement where features will arrive, rather than where they currently reside; and (iii) *long-horizon budget scheduling*: distributing a total rollout budget across both space and time, concentrating compute on timesteps where error amplification is greatest. These directions are important for rollout stability, which we do not evaluate in this work and leave to future investigation.

## 7 Conclusion

We introduced ATCA, a framework for adaptively allocating test-time compute in neural PDE solvers. A lightweight difficulty estimator identifies regions of high prediction error, and iterative refinement is concentrated there, achieving $2.5$–$4.3\times$ computational savings over uniform refinement at matched accuracy across four PDE benchmarks. The framework is grounded in classical AMR theory via an error equidistribution principle and is compatible with existing neural operator architectures, with preliminary evidence of applicability to pre-trained foundation models.

## References

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural opera-

tors. In *Advances in Neural Information Processing Systems*, 2024.

Hritik Bansal, Arian Hosseini, Rishabh Agarwal, Vinh Q Tung, and Aditya Grover. Smaller, weaker, yet better: Training LLM reasoners via compute-optimal sampling. *arXiv preprint arXiv:2408.16737*, 2024.

Marsha J Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484–512, 1984.

Johannes Brandstetter, Daniel E Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Willy Dörfler. A convergent adaptive algorithm for Poisson's equation. *SIAM Journal on Numerical Analysis*, 33(3):1106–1124, 1996.

Alex Graves. Adaptive computation time for recurrent neural networks. In *arXiv preprint arXiv:1603.08983*, 2016.

Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized PDE modeling. In *Transactions on Machine Learning Research*, 2023.

Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for PDEs. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

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.

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.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/IMS Journal of Data Science*, 1(3):1–27, 2024.

Phillip Lippe, Bastiaan S Veeling, Paris Perdikaris, Richard E Turner, and Johannes Brandstetter. PDE-Refiner: Achieving accurate long rollouts with neural PDE solvers. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Chaoyu Liu et al. Conservation-preserved Fourier neural operator through adaptive correction. *arXiv preprint arXiv:2505.24579*, 2025.

Ning Liu, Siavash Jafarzadeh, and Yue Yu. Harnessing the power of neural operators with automatically encoded conservation laws. In *arXiv preprint arXiv:2312.11176*, 2023.

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

Yuan Qiu and Peng Chen. Variationally correct operator learning: Reduced basis neural operator with a posteriori error estimation. *arXiv preprint arXiv:2512.21319*, 2025.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. In *International Conference on Machine Learning*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Machine Learning*, 2025.

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

Kiwon Um, Robert Brand, Yun Raymond Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. *Advances in Neural Information Processing Systems*, 33:6111–6122, 2020.

Rüdiger Verfürth. *A Posteriori Error Estimation Techniques for Finite Element Methods*. Oxford University Press, 2013.

Sifan Wang et al. Towards reasoning for PDE foundation models: A reward-model-driven inference-time-scaling algorithm. *arXiv preprint arXiv:2509.XXXXX*, 2025.

Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries. In *International Conference on Machine Learning*, 2024.

# A Proof Details

## A.1 Proof of Proposition 2

Under uniform allocation, each patch receives $k_i^u = B/M$ steps, yielding total error $E^u = \sum_i c_i (B/M)^{-\beta} = (M/B)^\beta \sum_i c_i$. Under optimal allocation from Proposition 1, the total error is:

$$E^* = \sum_i c_i \left( \frac{B c_i^{1/(\beta+1)}}{\sum_j c_j^{1/(\beta+1)}} \right)^{-\beta} = \frac{1}{B^\beta} \left( \sum_i c_i^{1/(\beta+1)} \right)^{\beta+1}. \tag{14}$$

The ratio is:

$$\frac{E^u}{E^*} = \frac{M^\beta \sum_i c_i}{\left( \sum_i c_i^{1/(\beta+1)} \right)^{\beta+1}}. \tag{15}$$

By the power mean inequality, $(\frac{1}{M} \sum_i c_i^{1/(\beta+1)})^{\beta+1} \leq \frac{1}{M} \sum_i c_i$ (since $1/(\beta+1) < 1$), which yields $E^u/E^* \geq 1$ with equality when all $c_i$ are equal.

For the concentrated case where $c_i = c$ for $\alpha M$ patches and $c_i = 0$ for the rest, the ratio becomes $(\alpha M)^{-\beta/(\beta+1)} \cdot M^{\beta/(\beta+1)} = \alpha^{-\beta/(\beta+1)}$.

# B Additional Experimental Details

**Hyperparameters and training protocol.** The base U-Net follows the architecture of Gupta & Brandstetter (2023) with hidden dimensions $[64, 128, 256, 1024]$ and $\sim$55M parameters. The difficulty estimator uses hidden dimensions $[32, 64, 64, 32]$ with group normalization and GELU activations. All components are trained with Adam (learning rate $10^{-4}$, weight decay $10^{-5}$), cosine annealing over 200 epochs, batch size 32, and gradient clipping at max norm 1.0. The refinement operator follows PDE-Refiner's denoising schedule with cosine noise scheduling. We use the standard PDEBench train/validation/test splits: for each PDE, 80%/10%/10% of samples are used for training, validation (hyperparameter selection and early stopping), and final evaluation, respectively. All reported metrics are computed on the held-out test set. Results in the main tables report mean $\pm$ standard deviation across 3 independent training seeds.

**Computational cost breakdown.** For a single 2D Navier-Stokes test instance on a single NVIDIA A100 GPU: base operator forward pass takes 4.2ms; difficulty estimation takes 0.3ms; one uniform refinement step takes 4.5ms; ATCA with $B = 4M$ averages 12.1ms total refinement (vs. 18.0ms for $K = 4$ uniform), a $1.49\times$ wall-clock speedup. The wall-clock speedup is smaller than the FLOP speedup because patches with zero refinement still require bookkeeping overhead and the partition-of-unity blending.