FlowForge: A Staged Local Rollout Engine for Flow Field Prediction

Xiaowen Zhang [1 *] **Ziming Zhou** [2 *] **Fengnian Zhao** [1] **David L.S. Hung** [1]

Abstract

Learning-based surrogate simulation for time-dependent physical flows must be accurate, robust to noisy data, and fast enough for repeated use. Existing surrogates often rely on complex models and update the entire field in one pass, which can be slow and fragile to imperfect data. We introduce **FlowForge**, a staged local rollout surrogate for fixed-horizon velocity-field prediction that orchestrates local updates within a single prediction. This design allows FlowForge to simulate flow fields using lightweight models while maintaining strong accuracy. Across incompressible, compressible, and multiphase benchmarks from CFDBench, PDEBench, and BubbleML, FlowForge matches or improves upon strong baseline families, while reducing fixed-horizon prediction latency and showing consistently higher robustness to noise and missing observations.

1. Introduction

Learning-based surrogate simulators for time-dependent physical flows are increasingly used to accelerate forward solves in computational fluid dynamics (CFD) and related PDE-governed systems. Given a current velocity field $U_t = (u_t, v_t) \in \mathbb{R}^{N\times 2}$, engineers and experimentalists repeatedly need a fast surrogate for the velocity field $U_{t+\tau}$ at a fixed frame offset τ to explore small changes in operating conditions, boundary settings, or geometry (Brunton et al., 2020). In this workflow, the surrogate simulator is only useful if it is (i) accurate under nominal conditions, (ii) reliable under realistic input imperfections (noise, missing regions, imperfect boundary conditions (Sciacchitano, 2019)), and (iii) fast enough for interactive iteration (Lu et al., 2021).

However, current surrogate simulators do not provide a reliable off-the-shelf solution. In practice, users must choose among many families (e.g., U-Net (Ronneberger et al., 2015), Fourier Neural Operators (FNO) (Li et al., 2021), and other physics-informed neural networks (PINNs) (Raissi et al., 2019)) whose performance and stability can vary substantially across datasets and noise regimes. Empirical studies in Section 4.2 show that even strong baselines degrade under realistic input corruptions and incur higher fixed-horizon prediction latency.

A common root cause is architectural. Many popular surrogate simulators apply models that rely on deep encoder–decoder pipelines or layers with global receptive fields, such as Fourier spectral operators or global attention, which mix information across the entire spatial grid in a single update. While regularization or spectral truncation can mitigate some instabilities, they do not alter the underlying information-flow graph: every output location is still computed in one global pass and can be influenced by defects anywhere in the input.

This global structure is not incidental but induced by the standard supervised objective, which trains the model to simulate the entire fixed-horizon field in one global update,

$$U_{t+\tau} = f_\tau(U_t).$$

Over short horizons, many flow updates are dominated by local transport, diffusion, and boundary interactions: each spatial site is often well predicted from a bounded neighborhood. Single-pass global predictors instead rewrite the full field at once, so distant defects can influence every site and inference pays for interactions that may be unnecessary in predominantly local regimes.

Our key idea is to execute one fixed-horizon prediction as a **local rollout**. Instead of updating all spatial sites simultaneously, we generate the single output field $U_{t+\tau}$ by progressively rewriting spatial sites in a compiled order. For each target site, a prediction uses (i) the current velocity field U_t, which provides current-frame local context, and (ii) nearby predecessor context from sites in $U_{t+\tau}$ that have already been written in earlier stages. In this way, local dependence is turned into a controllable execution graph and the order specifies how information is exposed during prediction. Figure 1 sketches this compile–execute procedure: the compiler builds a rollout plan from the current

[*]Equal contribution . [1]Shanghai Jiao Tong University, Shanghai, China [2]University of Michigan, Ann Arbor, USA.

Proceedings of the AI4Physics Workshop at the 43rd International Conference on Machine Learning (AI4Physics@ICML 2026), Seoul, South Korea. 2026. Copyright 2026 by the author(s).

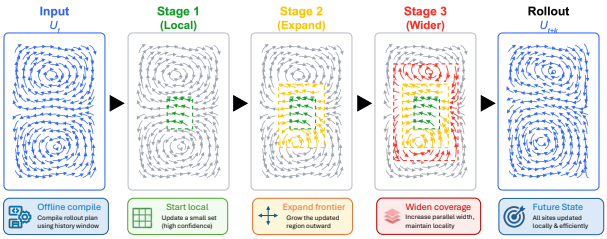

Figure 1. Compile–execute staged local rollout in FLOWFORGE. **Input:** offline compilation of the rollout plan from the current velocity field U_t. **Stages 1–3:** start from a small local core, expand the updated frontier outward, and widen parallel coverage while keeping bounded neighborhoods. **Rollout:** completed fixed-horizon field $U_{t+\tau}$ with all sites updated locally.

field, and the executor rewrites the field in widening local stages until all sites reach $U_{t+\tau}$.

Formally, this corresponds to factorizing the conditional distribution of the fixed-horizon output field into ordered, neighborhood-conditioned site-wise updates:

$$p(U_{t+\tau} \mid U_t) = \prod_{i=1}^{N} p\Big(u_{t+\tau}^{(\sigma(i))} \,\Big|\, U_t,\, u_{t+\tau}^{(\sigma(<i))}\Big), \qquad (1)$$

where σ defines an order over the N spatial sites, and $u_{t+\tau}^{(\sigma(<i))}$ denotes the predecessor context: output-field values generated earlier in that order.

Naively implementing (1) site by site is impractical: it creates a long sequential factorization chain, dramatically increasing the fixed-horizon prediction latency. This raises a central question: *can we preserve locality in the prediction graph while achieving fast, predictable inference at scale?*

FLOWFORGE addresses this with a *compile-execute design*. The *compiler* generates a staged local rollout plan. The plan fixes a rollout order, partitions sites into *stages*, and assigns each stage a plan-defined parallel width together with bounded predecessor contexts for its sites. The *executor* then rewrites the field stage by stage: each stage uses its assigned parallel width for local updates, while stages themselves are executed sequentially. In effect, the compiler fixes the prediction-time information flow, and the executor simply follows this plan during inference.

As a result, information and errors propagate only locally rather than globally, parallelism is exposed in a controlled, stage-wise manner, and each site is updated using a fixed-size neighborhood. Together, these choices make inference both hardware-friendly and latency-predictable, even as problem size grows.

FLOWFORGE achieves best-or-second-best pointwise accuracy on 10/11 datasets across incompressible, compressible, and multiphase flow dynamics, while keeping autoregressive temporal rollouts stable. Under additive noise and block or edge-localized masking, FLOWFORGE exhibits the smallest error increase and maintains a nearly flat per-point latency as resolution grows. We also identify a boundary of this locality-motivated design: in the Lid-driven Cavity case, globally coupled pressure dynamics can still favor global operators.

Contributions. (1) **Formulation:** We formulate fixed-horizon velocity-field prediction as a *staged local rollout*, factorizing $p(U_{t+\tau} \mid U_t)$ into ordered, neighborhood-conditioned spatial updates within one prediction. (2) **System:** We present FLOWFORGE, a compile–execute system that enables efficient and stable training and inference under this formulation. (3) **Evaluation:** We empirically validate FLOWFORGE on CFDBench, PDEBench, and BubbleML, covering accuracy, staged local rollout behavior, robustness, latency, and ablations.

2. Staged Local Rollout

A fixed-horizon velocity-field update in many time-dependent flow regimes has an effective dependency radius: transport is limited by finite propagation speed, and diffusion concentrates influence within a bounded neighborhood over the fixed prediction horizon. This radius motivates making locality an explicit constraint on the prediction graph. To implement (1) efficiently, we use a staged execution factorization: each prediction reads local context from the current frame and predecessor context from already-written sites in the output field, while each stage uses a plan-defined parallel width for local updates. Thus both context sources remain spatially bounded during inference, while the sequential depth and latency are fixed by the compiled plan.

2.1. Staged Formulation of Local Rollout

Rather than predicting all N spatial locations at once, we introduce an ordering σ over locations and partition it into K disjoint stages $S_1, \ldots, S_K$. The stages are executed sequentially, while locations within a stage are processed according to the stage's plan-defined parallel width. During this process, we maintain a working array $Y \approx U_{t+\tau}$ that is progressively filled.

At stage s, the predictor produces values for all locations in S_s using the current field U_t and the values already written in earlier stages:

$$Y_{S_s} \;=\; G_\theta\big(U_t,\, Y_{S_{<s}}\big), \qquad S_{<s} := \{S_1, \ldots, S_{s-1}\}. \qquad (2)$$

The update is masked in the sense that predictions at stage s may read U_t and $Y_{S_{<s}}$, but never entries written in the same or later stages. Executing stages $1{:}K$ yields a complete prediction Y.

This construction reduces the effective sequential depen-

dency depth from N locations to $K \ll N$. The stage schedule then becomes the main execution knob: it determines the sequential depth, the plan-defined parallel width of each stage, and how much predecessor context later locations can access. In practice, the dependency radius guides the local neighborhood in U_t, while the stage schedule determines which nearby output-field sites have already been written when a site is predicted.

2.2. Physical Dependency Structure and Stage Design

We use the finite spatial dependency radius of fixed-horizon updates as a design criterion for choosing context size and stage coverage. For instance, in an advection–diffusion system over a prediction horizon Δt, information is transported along characteristics at speed at most $c_{\max}$ and diffused over a radius proportional to $\sqrt{\nu \Delta t}$. Combining these effects yields a conservative bound on the fixed-horizon dependency radius,

$$r_{\text{req}} \approx c_{\max} \Delta t + r_p,$$

where r_p captures the effective diffusion radius (Proposition F.9; Appendix F.4). Beyond this distance, the influence of the current frame on a local update is negligible.

Bounded locality defines *which* predecessors within r_{req} are informative; the rollout schedule defines *when* they are written. FlowForge stages inference accordingly: context radius R selects neighbors, and $(\sigma, S_{1:K})$ orders their availability—the same separation as in synchronous local time stepping, but without a single global update pass.

On a uniform grid with spacing Δx, this suggests a context radius of approximately

$$R \approx \left\lceil \frac{r_{\text{req}}}{\Delta x} \right\rceil$$

neighboring rings or shells. Plans should supply predecessor coverage within this radius (Lemma F.10); $(\sigma, S_{1:K})$ is then chosen on validation under the latency budget.

2.3. Stability and Latency Implications

With these schedule principles in mind, we relate staging to conditioning quality and end-to-end latency. Later stages read more already-generated structure, so accuracy generally improves as K grows (Table 2).

K trades off parallelism and conditioning. Smaller K leaves more parallelism within each staged local rollout but weaker output-field conditioning across stages; larger K exposes richer predecessor context but increases sequential depth and latency. We choose K on validation subject to a latency budget.

Staging makes the latency side explicit. Stages run sequentially, while sites within a stage use the plan-defined parallel width. If updating one site costs t_{loc} and stage s processes $|S_s|$ sites with width w_s, then

$$T_{\text{total}} = \sum_{s=1}^{K} \left\lceil \frac{|S_s|}{w_s} \right\rceil t_{\text{loc}} \qquad \text{(Proposition F.16)}.$$

The compiler selects a plan $\mathcal{P} = (\sigma, S_{1:K}, H, w_{1:K})$ from candidate plans that satisfy this budget.

3. FlowForge Design

Section 2 motivates predicting $U_{t+\tau}$ via a *staged local rollout*: we maintain a working field and overwrite it in stages, with the rule that predictions in stage s may depend only on values written in stages $< s$. We adopt this formulation because it makes the dependency structure explicit and controllable. In particular, it (i) guarantees that within-stage updates can be parallelized up to the plan-defined width without circular dependencies, (ii) forces each prediction to use a bounded local neighborhood consistent with the required fixed-horizon dependency radius, and (iii) exposes direct accuracy–latency controls through the number of stages and the context size. FlowForge implements this idea as a compile–execute system: an offline compiler fixes an ordering, stage boundaries, and per-site context indices; an online executor follows these tables to produce $\widehat{U}_{t+\tau}$.

3.1. Overview

FlowForge separates *planning* from *execution*. The plan determines what information is visible when predicting each site. Here, a *site* denotes one grid cell or mesh point together with its physical channels, and a *stage* denotes a block of ordered sites with a plan-defined parallel width, not a physical sub-timestep. The plan specifies (i) a traversal order σ over sites, (ii) a partition of the resulting sequence into stages $S_{1:K}$, and (iii) for each site, a fixed list of at most H earlier indices that may be read as context. The executor is then a deterministic staged rewrite of a working buffer: for stage s, it gathers the plan-specified context for all $i \in S_s$, applies a shared local predictor G_θ in batch, and writes the results back. Because the plan only allows reads from earlier stages, the dependency graph is acyclic by construction and no runtime masking checks are required.

This structure positions the remaining subsections. Section 3.2 formalizes the executor given a plan. Section 3.3 describes how the plan and lookup tables are synthesized to balance neighborhood coverage against sequential depth. Section 3.4 aligns training with the same plan-defined inputs to avoid rollout mismatch. Section 3.5 extends the same mechanism to high-resolution settings.

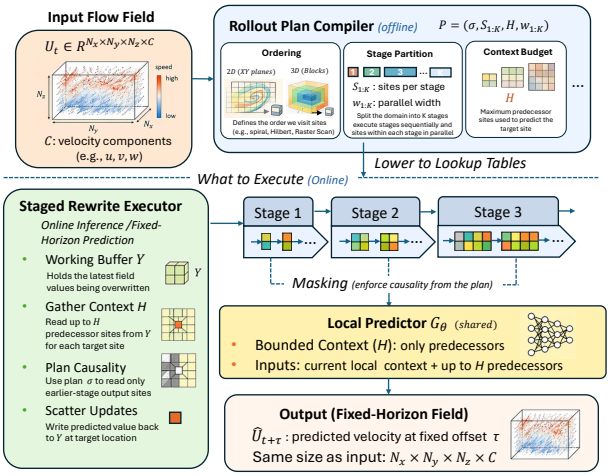

Figure 2. FLOWFORGE workflow. Offline, we compile a staged local rollout plan and lower it into index tables. Online, the executor overwrites a working buffer stage by stage using a shared local predictor G_θ, producing $\widehat{U}_{t+\tau}$.

3.2. Staged Rewrite Executor

At inference time, the executor maintains a working buffer in the compiled rollout order and rewrites it in K stages. Let σ be the compiled ordering of spatial sites, and let $U_t \in \mathbb{R}^{N\times 2}$ be the current two-channel velocity field. The buffer is a sequence

$$Y \;=\; (y_1, \dots, y_N), \qquad y_i \in \mathbb{R}^c,$$

where y_i corresponds to site $\sigma(i)$ and is initialized from the current field,

$$y_i \leftarrow u_t^{(\sigma(i))} \qquad \text{for } i = 1, \dots, N,$$

which is subsequently overwritten into the prediction $\widehat{U}_{t+\tau}$. Only overwritten sites from earlier stages are used as predecessor context; unwritten current-field sites are never treated as valid predecessors.

The staged local rollout plan partitions indices $1{:}N$ into disjoint stages $S_1, \dots, S_K$ executed sequentially. Each stage processes sites in S_s according to its plan-defined parallel width, subject to an acyclic computation-graph constraint: a site in stage s may only read values written in earlier stages. For each $i \in S_s$, the compiler constructs a context set $\mathrm{ctx}(i)$ satisfying

$$\mathrm{ctx}(i) \subseteq \bigcup_{r=1}^{s-1} S_r.$$

The plan is lowered into index tables so the executor can gather predecessor context without runtime decisions. At stage s, the executor evaluates, for all $i \in S_s$,

$$\widehat{y}_i \;=\; G_\theta\Big(\Phi(U_t, \sigma(i)),\; \Psi(Y;\, i),\; \mathrm{pos}(\sigma(i))\Big), \qquad (3)$$

and then commits the results by writing them back to the working buffer:

$$y_i \leftarrow \widehat{y}_i, \qquad i \in S_s. \qquad (4)$$

The input to G_θ is intentionally small and fixed across resolutions. Each reported field is represented as a structured tensor $U_t \in \mathbb{R}^{N\times 2}$ whose channels are the horizontal and vertical velocity components. The target feature $\Phi(U_t, \sigma(i))$ contains the current velocity at the target site; longer histories, when used, are handled by concatenating channels. We append normalized grid coordinates, distance-to-boundary features, and static boundary/geometry masks when a benchmark provides them; otherwise the geometry mask is all-fluid and the boundary feature is derived from the rectangular grid. The predecessor context feature $\Psi(Y; i)$ contains up to H predecessor sites from earlier stages, represented by their working buffer values and relative offsets to $\sigma(i)$. If fewer than H predecessors are valid, the remaining slots are zero-padded and marked by validity bits. Corruption masks used in robustness tests are not passed as extra input channels.

We use a single shared local predictor G_θ across all stages: the update rule is stationary, and stages differ only in how much previously written neighborhood is exposed by $\Psi(\cdot)$. The computation-graph constraint ensures correctness of within-stage parallelism: because every $\Psi(Y; i)$ indexes only $\cup_{r<s} S_r$, all context values are already written and remain fixed while stage s is computed, so no prediction in S_s depends on another prediction in S_s. Consequently, the executor can batch sites in S_s according to the stage width and evaluate (3) without circular dependencies or runtime masking.

Operationally, each stage follows a three-step loop: (1) gather a bounded set of predecessor context vectors from Y using the precomputed indices, (2) apply G_θ to the resulting batch, and (3) scatter the outputs back into Y. This exposes two direct runtime controls: increasing the number of stages K increases sequential depth, while increasing the context budget H increases available predecessor information per update at the cost of additional memory traffic. Because both quantities are fixed by the plan, inference latency is predictable.

3.3. Rollout Plan Compiler

After choosing the executor, the main design question is how to choose a plan that gives each prediction enough local information without making the staged local rollout too sequential. The compiler outputs

$$\mathcal{P} \;=\; (\sigma,\; S_{1:K},\; H,\; w_{1:K})$$

Here σ is the order of sites, $S_{1:K}$ is the stage partition in that order, H is the maximum number of predecessor

context sites used by each prediction, and w_s is the batch width used when executing stage s. Optionally, we can also apply a coarse-to-fine strategy with factor ρ to downscale the grid and upsample back to the native grid (Section 3.5); we treat ρ as an execution option rather than part of $\mathcal{P}$. In the main experiments we search $K \in \{4, 16, 64, 128, 256\}$, $H \in \{8, 16, 32, 64\}$, and $\rho \in \{1, 2, 4\}$ on validation data subject to the latency budget.

We implement the compiler in two steps. First, we choose σ and $S_{1:K}$ to make already-written neighbors available at decode time. Section 2 motivates a required dependency radius r_{req}; on a grid with spacing Δx this corresponds to roughly $R \approx \lceil r_{\text{req}}/\Delta x \rceil$ rings (2D) or shells (3D). Intuitively, we want a site to be decoded after many sites in its R-ring neighborhood have already been written. Second, once σ and $S_{1:K}$ are fixed, we precompute simple index tables so the executor can read context and write outputs without runtime decisions.

In this work we use an Outward Spiral traversal: starting from the integer-rounded grid center, we visit spatially adjacent sites while expanding the visited region by Chebyshev rings (2D) or shells (3D). Outward Spiral is a simple geometry-balanced default: it keeps the visited set compact, expands the frontier approximately uniformly, and increases the fraction of already-written neighbors as the staged local rollout progresses. Other schedules are valid when they expose nearby predecessors under the same predecessor context budget H. We then use a simple, predictable stage template (equal-sized by default), which makes K a direct knob for latency. For each target site, predecessor candidates are restricted to earlier stages, ordered by spatial proximity to the target with rollout index as a deterministic tie-breaker, truncated to H, and zero-padded with explicit validity bits when fewer than H predecessors are available.

The resulting executor runtime can be estimated under the same idealized model as Section 2. Because stages run sequentially, total time is approximately

$$T_{\text{total}}(\mathcal{P}) \;=\; \sum_{s=1}^{K} \left\lceil \frac{|S_s|}{w_s} \right\rceil t_{\text{loc}}(H),$$

where $t_{\text{loc}}(H)$ summarizes the per-batch cost of reading H context vectors and running G_θ.

This analytical model defines the feasible search space of configurations satisfying a latency budget T_{budget}. Within this feasible set, we employ a physics-informed selection strategy to identify the optimal plan $\mathcal{P}^*$. Specifically, we execute candidate configurations on a validation subset and minimize a composite objective:

$$\mathcal{L}_{\text{total}}(\mathcal{P}) = \sum_{m \in \mathcal{M}} \lambda_m \frac{\mathcal{E}_m(\mathcal{P})}{\mu_m}.$$

Algorithm 1 Two-step compiler for FLOWFORGE

Require: Grid sites $\mathcal{V}$ (size N), spacing Δx, required radius r_{req}, number of stages K (or stage size), predecessor context budget H
Ensure: Rollout plan $\mathcal{P}$ and index tables $\mathcal{T}$
$R \leftarrow \lceil r_{\text{req}}/\Delta x \rceil$
Choose traversal σ (Outward Spiral from seed)
Partition $1{:}N$ into stages $S_1, \ldots, S_K$ (e.g., equal-sized)
Build stage tables (`stage_ptr`, `stage_idx`) for iterating stage indices
for $i = 1$ to N **do**
  Find candidate spatial neighbors of site $\sigma(i)$ within R rings/shells
  Let s be the stage containing i (i.e., $i \in S_s$)
  Sort candidates by distance to $\sigma(i)$, breaking ties by rollout index
  Keep up to H neighbors j such that $j < i$ **and** $j \in \cup_{r=1}^{s-1} S_r$ (i.e., j is in an earlier stage)
  Store them in `ctx_idx`$[i, 1{:}H]$; zero-pad and store a validity mask if fewer than H exist
end for
Build per-site metadata tables for $\text{pos}(\cdot)$ and $\Phi(\cdot)$
return $\mathcal{P} = (\sigma, S_{1:K}, H, w_{1:K})$ and $\mathcal{T}$

Here, $\mathcal{M} = \{\text{RMSE}, \text{Vor}, \text{Spec}, \text{Div}\}$ denotes the set of error metrics (see Appendix A.4 for the metric definitions). The term $\mathcal{E}_m(\mathcal{P})$ represents the error computed for plan $\mathcal{P}$ on metric m, and μ_m is the mean error across all candidates used for normalization. We assign weights $\lambda_{\text{RMSE}} = \lambda_{\text{Vor}} = \lambda_{\text{Spec}} = 0.3$ to balance prediction accuracy, physical structure preservation, and spectral fidelity, and $\lambda_{\text{Div}} = 0.1$ as a soft physical conservation constraint. These weights were chosen to prioritize pointwise accuracy and physical structure while ensuring approximate mass conservation. This optimization is performed offline, and the resulting compiled tables are reused across all fixed-horizon predictions.

3.4. Training Without Rollout Mismatch

The staged local rollout enforces a strict information pattern: each site is predicted from current-frame local features plus bounded predecessor context drawn only from earlier stages (Section 3.2). A naive supervised setup can accidentally leak *future-stage* information during training, creating a train–test mismatch that amplifies under staged local rollout. We avoid this by training G_θ with the *same plan-defined inputs* as inference (see Appendix B.4 for details).

Concretely, let $\mathbf{v}^\star_{1:N} = (u^{(\sigma(1))}_{t+\tau}, \ldots, u^{(\sigma(N))}_{t+\tau})$ be the ground-truth fixed-horizon output sequence under order σ. For each index i, we form:

$$x^\star_i \;=\; \Big[\; \Phi(U_t, \sigma(i)),\; \Psi\big(\mathbf{v}^\star;\, i\big),\; \text{pos}(\sigma(i)) \;\Big] \quad \longmapsto \quad \mathbf{v}^\star_i,$$

and minimize mean-squared error over $i = 1{:}N$.

3.5. Scalability and Model Backend

Inference latency scales with the number of sites N, but FLOWFORGE exposes simple knobs that keep latency predictable: bounded per-site context and a fixed number of sequential stages. For very high-resolution grids, we optionally use a coarse-to-fine execution path that shortens the effective sequence length; and once the plan fixes the information flow, a lightweight predictor backend is sufficient. We defer these engineering details to Appendix B.5. In Section 4, we show that this design achieves strong accuracy with low and controllable latency.

4. Results

4.1. Accuracy Across Physical Regimes

We evaluate FLOWFORGE on eleven datasets spanning three physics families: incompressible boundary-driven flows in CFDBench, compressible and reaction–diffusion dynamics in PDEBench, and multiphase or phase-change flows in BubbleML (Appendix A). We compare against DeepONet, U-Net, and FNO, and include CNO/UNO where applicable. All models use the same 48-hour training budget and identical train/validation splits; implementation details and the compact grid searches for settings absent from the original benchmarks appear in Appendix B and Appendix C.

Table 1 reports RMSE together with vorticity, spectral, and divergence errors. Across these regimes, FLOWFORGE achieves the best or second-best RMSE on 10/11 datasets and the lowest divergence error on 10/11 datasets. The gains are most consistent when local transport, interfaces, or boundary-driven vortices dominate: staged conditioning preserves pointwise accuracy while limiting non-local error spread. Appendix D.1 further shows that these gains do not come at the expense of unstable long-horizon autoregressive temporal rollout behavior.

The main failure case is Lid-driven Cavity, where FLOWFORGE is suboptimal in both RMSE and auxiliary metrics. This closed-domain incompressible regime includes an elliptic pressure projection, so pressure constraints couple distant sites and errors recirculate through the walls. Such global coupling is precisely where a local forward-conditioned execution plan is least appropriate, and where global operators or hybrid global corrections can be preferable.

Figure 3 zooms into regions of strong shear and vortex shedding, where small phase or amplitude errors quickly become visually salient. FLOWFORGE closely matches the ground-truth streamlines in both regions: in R1, predicted streamlines are smooth and follow the main flow, while in R2, vortex cores are captured with little distortion. Errors are sparse and remain confined to high-vorticity areas, indicating that the staged local rollout limits their spread. By contrast, FNO preserves the bulk flow but blurs high-frequency structures near interfaces due to global spectral mixing, and U-Net produces overly uniform streamlines with widespread artifacts, likely from amplifying local uncertainty across the domain. Overall, this case study shows that the benefit of FLOWFORGE is not just lower aggregate error but also better error geometry: errors are localized, physically interpretable, and tied to genuine modeling difficulty rather than artifacts of global coupling.

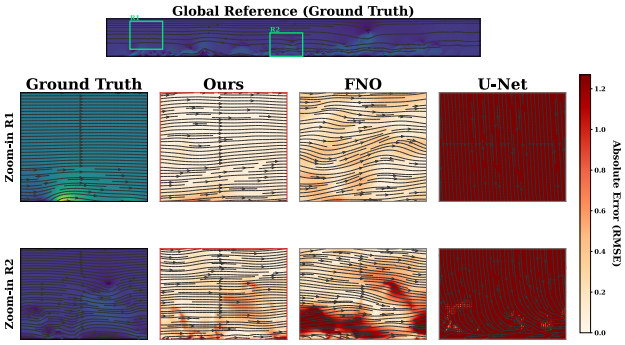

Figure 3. Streamline prediction visualizations for the FB-Gravity dataset.

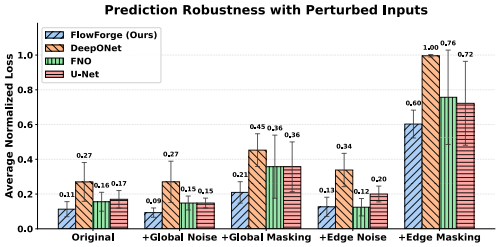

Figure 4. Quantitative evaluation of model robustness under input perturbations. RMSE losses are max-scaled normalized relative to the worst-performing case per dataset. Bar heights indicate the average normalized loss, and error bars represent the standard deviation.

4.2. Robustness to Input Corruptions

We evaluate robustness on CFDBench under Gaussian noise and spatial masking, applied both globally and at edges. We aggregate losses across a range of corruption levels and normalize them for scale (Appendix D.3). Figure 4 shows that FLOWFORGE incurs the smallest error increase across nearly all perturbations. The margin is strongest under masking and near boundaries, where global single-pass predictors amplify local defects most severely. This matches the intended mechanism of the staged local rollout: corrupted observations can only influence bounded neighborhoods rather than immediately contaminating the full field.

To make the aggregate trend concrete, Figure 5 shows a rep-

Table 1. Comparison of different models. Best values are **bolded**, second best are underlined. Entries marked with * do not have official hyperparameters reported in the original benchmarks, and we run a short grid search for 5 epochs over canonical architectural choices.

CFDBench																
Dataset	Tube				Cylinder				Dam				Cavity			
Model	RMSE	Vor	Spec	Div	RMSE	Vor	Spec	Div	RMSE	Vor	Spec	Div	RMSE	Vor	Spec	Div
FNO	0.1156	0.5330	1.2647	0.2396	0.0915	0.1466	0.5345	0.2046	0.0491	**0.1705**	0.5225	0.1070	0.1911	0.0921	0.4822	0.0650
U-Net	0.1461	0.7464	1.5991	0.2402	0.0915	0.1466	0.5289	0.2044	0.0494	0.1742	0.5136	0.1069	**0.1456**	0.0715	0.3608	0.0342
DeepONet	0.3673	0.8676	0.8279	0.0601	0.0709	0.1173	**0.2193**	0.0346	0.0772	0.5063	**0.4794**	0.0347	0.2710	0.0879	**0.3525**	**0.0301**
CNO*	0.1873	0.8997	1.5403	0.2306	0.0924	0.1501	0.5583	0.2056	0.0955	0.3624	0.6372	0.1120	0.2385	0.0928	0.4895	0.0587
UNO*	0.1301	0.7057	1.4948	0.2272	0.1526	0.3376	0.8219	0.2128	0.0691	0.3458	1.1906	0.1109	0.1476	**0.0693**	0.3883	0.0390
FlowForge	**0.1035**	**0.3326**	**0.8208**	**0.0205**	**0.0258**	**0.0806**	0.4325	**0.0215**	**0.0222**	0.2310	0.5303	**0.0173**	0.2949	0.2076	0.6573	0.0919
PDEBench																
Dataset	Diff-React				Rand-M0.1				Rand-M1.0							
Model	RMSE	Vor	Spec	Div	RMSE	Vor	Spec	Div	RMSE	Vor	Spec	Div				
FNO	0.0850	1.0010	16.9993	2.4304	0.1370	**0.3684**	**2.5852**	5.0081	**0.1241**	**0.7569**	**4.3602**	10.5203				
U-Net	0.0871	**0.7544**	**1.5762**	1.8133	0.9253	1.0838	3.9059	14.6274	1.3545	1.2576	6.4127	20.9948				
FlowForge	**0.0837**	1.0634	1.6317	**0.0422**	**0.0451**	1.2696	2.8513	**0.0301**	0.1351	3.4283	5.7483	**0.0938**				
BubbleML																
Dataset	FB-Gravity				PB-Gravity				PB-Subcooled				FB-VelScale			
Model	RMSE	Vor	Spec	Div	RMSE	Vor	Spec	Div	RMSE	Vor	Spec	Div	RMSE	Vor	Spec	Div
FNO	0.4600	1.1274	2.5466	79.4190	0.3304	**1.0191**	2.5540	9.7240	0.4922	**1.0122**	2.5957	8.1798	0.9042	1.3128	**2.0251**	192.8551
U-Net	122.3470*	512.2977*	3.3801*	6.59×10^4 *	**0.2873**	1.0467	1.9896	12.2452	0.5030	1.0462	1.9598	12.3974	9.8684*	6.0919*	3.5103*	1.35×10^3 *
FlowForge	**0.1869**	**0.9843**	**2.2159**	**0.0240**	0.2898	1.0753	**1.4567**	**0.0289**	**0.3798**	1.1878	**1.5041**	**0.0371**	**0.5382**	**0.9678**	2.2434	**0.0421**

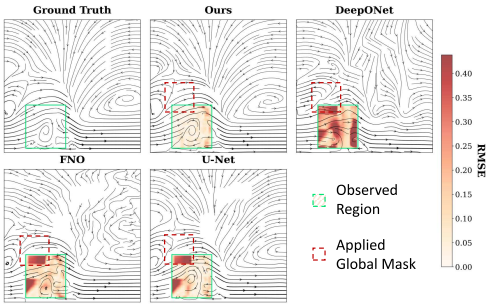

Figure 5. Robustness case study under global block masking corruption.

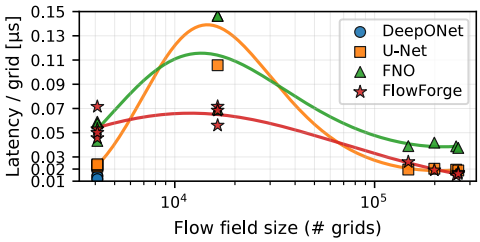

Figure 6. Latency comparison across datasets of increasing resolution. Lower is better.

resentative CFDBench example under strong global block masking. We apply a mask composed of contiguous spatial blocks and compare the reconstructed fields. For FNO, U-Net, and DeepONet, the block pattern is visibly imprinted in the predictions: masked regions reappear as patchy artifacts or over-smoothed blobs, and the corruption bleeds into neighboring clean areas. In contrast, FlowForge produces a reconstruction that remains visually close to the ground truth, with almost no visible trace of the mask and errors confined to a narrow band surrounding the occluded blocks. This qualitative pattern is exactly the error-containment mechanism suggested by the aggregate robustness results.

4.3. Latency Analysis

We report inference latency for 100 held-out fixed-horizon predictions $U_t \mapsto U_{t+\tau}$, normalized by the number of spatial grid points. The normalization captures how efficiently each method processes a single spatial location as the problem size grows. Figure 6 summarizes results on datasets of increasing resolution.

FlowForge achieves competitive per-point latency across problem sizes and is particularly favorable on medium-to-large grids. This latency profile comes from stage-wise parallelism: each stage uses its plan-defined parallel width for local updates, and the fixed number of sequential stages makes the total execution depth predictable. U-Net and FNO exhibit a more noticeable latency increase as resolution grows, consistent with their global single-pass fixed-horizon updates.

4.4. Ablation Study

We ablate core design choices on CFDBench-Cylinder: staged local rollout versus single-pass prediction, rollout granularity K, traversal order σ, and local predictor G_θ. Table 2 reports the selected configuration together with controlled variations.

Is FlowForge merely enforcing locality? One concern is that the observed gains may stem solely from restricting each prediction to a local neighborhood, rather than from the staged local rollout itself. We therefore introduce a *local single-pass* baseline that uses the same current-frame local

Table 2. **Ablation study of key design components on CFDBench-Cylinder.** The base configuration is our best setting on this dataset: K=128, σ=Outward Spiral, and local predictor G_θ=MLP.

Block	Setting	RMSE	Latency (ms)
Base	**FlowForge**	0.0258	161.6
Struct.	single-pass (no staged rollout)	0.3457	104.7
K	4	0.0403	147.9
	16	0.0364	149.3
	64	0.0291	148.8
	256	0.0242	169.7
σ	Raster Scan	0.0262	159.6
	Random	0.0265	166.4
	Hilbert Curve	0.0260	175.7
G_θ	CNN	0.3401	206.2
	FFN	0.8469	219.2

features but predicts all sites in one fixed-horizon pass. Its much higher RMSE (0.3457 vs. 0.0258) shows that current-frame locality alone is insufficient; the gain depends on staged conditioning from earlier output-field values.

Effect of stage granularity (K). Increasing K improves RMSE by exposing more predecessor context (K=256 is best in Table 2), while latency stays nearly flat up to K=64 and rises when K=256 becomes highly sequential. Our default K=128 is the validation choice under the latency budget (Section 2.3).

Effect of traversal order (σ). Outward Spiral, Raster Scan, Hilbert Curve, and Random achieve nearly the same RMSE. Accuracy depends on nearby already-written predecessors, not on a privileged global traversal path; after a short warm-up, even Random attains comparable local coverage (Appendix D.2). Thus σ schedules when that coverage becomes available, while Section 2.2 sets the bounded context radius.

Local predictor backbone (G_θ). With the staged executor fixed, the MLP gives the best accuracy and latency; CNN/FFN alternatives are worse for the sparse predecessor lists produced by the compiler. We therefore use the MLP throughout the paper.

5. Related Work

Learning-based surrogates for PDE-governed dynamics are commonly improved by stabilizing rollout across time steps or by changing the fixed-horizon predictor. FLOWFORGE is orthogonal: it changes how one fixed-horizon prediction is executed through staged local context.

Temporal Rollout Across Steps Versus Staged Local Rollout Prediction. A large body of work studies how a learned predictor composes across physical time steps, emphasizing rollout stability, long-horizon error accumulation, and temporal memory (Hao et al., 2024; Ye et al., 2024; Zhou et al., 2025; Knlgge et al., 2024; Brandstetter et al., 2022; McCabe et al., 2023; Hemmasian & Barati Farimani, 2024; Buitrago et al., 2025; Diab & Al Kobaisi, 2025). These methods modify how a map such as $U_t \mapsto \widehat{U}_{t+\tau}$ is trained, regularized, or repeatedly applied over time. FLOWFORGE instead decomposes one fixed-horizon prediction into staged local updates, making it an execution design for fixed-horizon prediction rather than a multi-step stability method.

Backbone-Centric Surrogate Design Versus Execution-Centric Design. Most learned surrogates improve the applied predictor: classical neural operators and operator regression methods (Li et al., 2021; Lu et al., 2021; Li et al., 2020), geometric or multiscale variants (Li et al., 2023a;b; Raonic et al., 2023; Tran et al., 2023; Rahman et al., 2023), and transformer-style operator models (Hao et al., 2023; Xiao et al., 2024; Liu et al., 2023; Wu et al., 2024; Bryutkin et al., 2024; Herde et al., 2024; Ye et al., 2024; Serrano et al., 2023) expand the function class, receptive field, or global mixing pattern of the fixed-horizon predictor. Despite these differences, they typically treat each prediction as a synchronous field-wide update. FLOWFORGE instead compiles a locality-preserving staged schedule with bounded predecessor context, allowing a lightweight predictor to use explicit local information flow during fixed-horizon prediction.

6. Conclusion

We presented FLOWFORGE, a staged local rollout surrogate simulator for fixed-horizon velocity-field prediction. Its compiled staged execution plan uses bounded local features and predecessor context to improve robustness and keep latency predictable. Across incompressible, compressible, and multiphase benchmarks, FLOWFORGE matches or outperforms global baselines on most datasets and is more resilient to noisy or partial observations. Lid-driven Cavity clarifies the boundary of this locality-motivated design: elliptic pressure coupling and other global constraints can still favor global operators, motivating hybrids with explicit global correction.

Impact Statement

Deploying black-box surrogate models for fluid dynamics often involves brittle performance and tuning sensitivity. FLOWFORGE reframes surrogate design around a compile–execute abstraction with interpretable planning knobs, such as rollout granularity and locality, that control accuracy–latency tradeoffs at inference time.

References

Brandstetter, J., Worrall, D. E., and Welling, M. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vSix3HPYKSU.

Brunton, S. L., Noack, B. R., and Koumoutsakos, P. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020. doi: 10.1146/annurev-fluid-010719-060214. URL https://www.annualreviews.org/doi/10.1146/annurev-fluid-010719-060214.

Bryutkin, A., Huang, J., Deng, Z., Yang, G., Schönlieb, C.-B., and Aviles-Rivero, A. Hamlet: graph transformer neural operator for partial differential equations. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024. URL https://dl.acm.org/doi/10.5555/3692070.3692256.

Buitrago, R., Marwah, T., Gu, A., and Risteski, A. On the benefits of memory for modeling time-dependent PDEs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=o9kqa5K3tB.

Diab, W. and Al Kobaisi, M. Temporal neural operator for modeling time-dependent physical phenomena. *Scientific Reports*, 15(1):32791, 2025. doi: 10.1038/s41598-025-16922-5. URL https://doi.org/10.1038/s41598-025-16922-5.

Hao, Z., Wang, Z., Su, H., Ying, C., Dong, Y., Liu, S., Cheng, Z., Song, J., and Zhu, J. Gnot: a general neural operator transformer for operator learning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, 2023. URL https://dl.acm.org/doi/10.5555/3618408.3618917.

Hao, Z., Su, C., Liu, S., Berner, J., Ying, C., Su, H., Anandkumar, A., Song, J., and Zhu, J. Dpot: auto-regressive denoising operator transformer for large-scale pde pre-training. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024. doi: 10.5555/3692070.3692773. URL https://dl.acm.org/doi/10.5555/3692070.3692773.

Hassan, S. M. S., Feeney, A., Dhruv, A., Kim, J., Suh, Y., Ryu, J., Won, Y., and Chandramowlishwaran, A. BubbleML: A multi-physics dataset and benchmarks for machine learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=0Wmglu8zak.

Hemmasian, A. and Barati Farimani, A. Multi-scale time-stepping of partial differential equations with transformers. *Computer Methods in Applied Mechanics and Engineering*, 426:116983, 2024. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2024.116983. URL https://www.sciencedirect.com/science/article/pii/S0045782524002391.

Herde, M., Raonic, B., Rohner, T., Käppeli, R., Molinaro, R., de Bezenac, E., and Mishra, S. Poseidon: Efficient foundation models for PDEs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=JC1VKK3UXk.

Knlgge, D. M., Wessels, D. R., Valperga, R., Papa, S., Sonke, J.-J., Gavves, E., and Bekkers, E. J. Space-time continuous pde forecasting using equivariant neural fields. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, 2024. ISBN 9798331314385. doi: 10.5555/3737916.3740354. URL https://dl.acm.org/doi/10.5555/3737916.3740354.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Multipole graph neural operator for parametric partial differential equations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, 2020. URL https://dl.acm.org/doi/10.5555/3495724.3496291.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Li, Z., Huang, D. Z., Liu, B., and Anandkumar, A. Fourier neural operator with learned deformations for pdes on general geometries. *J. Mach. Learn. Res.*, January 2023a. URL https://dl.acm.org/doi/10.5555/3648699.3649087.

Li, Z., Kovachki, N., Choy, C., Li, B., Kossaifi, J., Otta, S., Nabian, M. A., Stadler, M., Hundt, C., Azizzadenesheli, K., and Anandkumar, A. Geometry-informed neural operator for large-scale 3d pdes. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 35836–35854. Curran Associates, Inc., 2023b.

Liu, X., Xu, B., and Zhang, L. HT-net: Hierarchical transformer based operator learning model for multiscale

PDEs, 2023. URL https://openreview.net/forum?id=UY5zS0OsK2e.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3:218–229, 2021. doi: 10.1038/s42256-021-00302-5. URL https://www.nature.com/articles/s42256-021-00302-5.

Luo, Y., Chen, Y., and Zhang, Z. Cfdbench: A large-scale benchmark for machine learning methods in fluid dynamics, 2023. URL https://arxiv.org/abs/2310.05963. arXiv:2310.05963.

McCabe, M., Harrington, P., Subramanian, S., and Brown, J. Towards stability of autoregressive neural operators. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=RFfUUtKYOG.

Rahman, M. A., Ross, Z. E., and Azizzadenesheli, K. U-NO: U-shaped neural operators. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=j3oQF9coJd.

Raissi, M., Perdikaris, P., and Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Raonic, B., Molinaro, R., Ryck, T. D., Rohner, T., Bartolucci, F., Alaifari, R., Mishra, S., and de Bezenac, E. Convolutional neural operators for robust and accurate learning of PDEs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=MtekhXRP4h.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241, Cham, 2015. Springer International Publishing. URL https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28.

Sciacchitano, A. Uncertainty quantification in particle image velocimetry. *Measurement Science and Technology*, 30(9):092001, 2019. doi: 10.1088/1361-6501/ab1db8. URL https://iopscience.iop.org/article/10.1088/1361-6501/ab1db8.

Serrano, L., Boudec, L. L., Koupaï, A. K., Wang, T. X., Yin, Y., Vittaut, J.-N., and patrick gallinari. Operator learning with neural fields: Tackling PDEs on general geometries. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=4jEjq5nhg1.

Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. PDEBench Datasets. DaRUS, 2022. URL https://doi.org/10.18419/darus-2986.

Tran, A., Mathews, A., Xie, L., and Ong, C. S. Factorized fourier neural operators. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=tmIiMPl4IPa.

Wu, H., Luo, H., Wang, H., Wang, J., and Long, M. Transolver: a fast transformer solver for pdes on general geometries. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024. URL https://dl.acm.org/doi/10.5555/3692070.3694270.

Xiao, Z., Hao, Z., Lin, B., Deng, Z., and Su, H. Improved operator learning by orthogonal attention, 2024. URL https://openreview.net/forum?id=mt5NPvTp5a.

Ye, Z., Huang, X., Chen, L., Liu, H., Wang, Z., and Dong, B. PDEformer: Towards a foundation model for one-dimensional partial differential equations. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024. URL https://openreview.net/forum?id=GLDMCwdhTK.

Zhou, H., Ma, Y., Wu, H., Wang, H., and Long, M. Unisolver: PDE-conditional transformers are universal PDE solvers, 2025. URL https://openreview.net/forum?id=f3xXPDCh8Q.

A. Dataset and Evaluation Metrics

To ensure reproducibility and facilitate fair comparison, we provide a comprehensive breakdown of the datasets used in our experiments, spanning three benchmark suites: CFDBench, PDEBench, and BubbleML. Our main evaluation uses eleven datasets (Turb-M0.1 is excluded from the main results). In this section, we detail the governing equations and physical parameters for each dataset.

A.1. CFDBench

We utilize four classical 2D fluid dynamics scenarios from CFDBench (Luo et al., 2023). These flows are governed by the Navier-Stokes equations for an incompressible Newtonian fluid.

Governing Equations. Let $\mathbf{u} = (u, v)^\top$ denote the velocity field, p the pressure, ρ the fluid density, and μ the dynamic viscosity. The dynamics follow:

$$\begin{cases} \nabla \cdot \mathbf{u} = 0 \\ \dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\dfrac{1}{\rho}\nabla p + \dfrac{\mu}{\rho}\nabla^2 \mathbf{u} \end{cases}$$

The boundary conditions are defined such that $\mathbf{u}$ is constant on the domain boundaries $\partial\mathcal{D}$. The four specific configurations are:

- `Cavity` (Lid-driven Cavity): Flow in a square container driven by a moving top lid. This dataset poses a challenge due to the mathematical singularities at the corners where the moving lid meets stationary walls, requiring the model to resolve sharp gradients and recirculating vortices.
- `Tube` (Circular Tube Flow): Represents the development of a laminar boundary layer. Physically, viscous resistance near the wall creates a parabolic velocity profile. This task tests the model's ability to capture boundary layer theory and liquid-gas interfaces in a confined geometry.
- `Dam` (Dam Break): Models the transient release of fluid over an obstacle using the Volume of Fluid (VOF) method. The flow regime transitions dynamically from viscosity-dominated (creeping flow) to inertia-dominated (jet formation), testing generalization across different physical force balances.
- `Cylinder` (Flow Around a Cylinder): A classic aerodynamic problem where boundary layer separation creates a Karman vortex street. This is the primary benchmark for modeling periodic oscillatory behaviors induced by rigid obstacles.

A.2. PDEBench

The selected datasets from PDEBench (Takamoto et al., 2022) introduce compressibility and non-linear chemical dynamics, significantly expanding the physical scope beyond standard CFD tasks.

Compressible Navier-Stokes (`Rand-M0.1`, `Rand-M1.0`, `Turb-M0.1`). These datasets evolve density ρ, velocity $\mathbf{v}$, and pressure p coupled with internal energy, governed by the full conservation laws:

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \quad \rho(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \nabla \cdot \sigma'$$

To ensure precise reproducibility, we specify the exact subsets used from the PDEBench repository. The datasets differ in both compressibility (Mach number M) and viscosity parameters (shear η and bulk ζ):

- `Rand-M0.1`: Corresponds to file `2D_Rand_-M0.1_Eta0.01_Zeta0.01`. This represents a subsonic regime ($M < 0.1$) with low viscosity ($\eta = \zeta = 10^{-2}$), serving as a baseline bridging incompressible and compressible physics.
- `Rand-M1.0`: Corresponds to file `2D_Rand_-M1.0_Eta0.1_Zeta0.1`. This represents a transonic regime ($M \leq 1.0$) with shock waves. Note that this dataset uses higher viscosity ($\eta = \zeta = 10^{-1}$) than M0.1. The model must capture sharp discontinuities where gradients become theoretically infinite.

2D Diffusion-Reaction (`Diff-React`). This dataset models biological pattern formation (activator u, inhibitor v) via the FitzHugh-Nagumo equations:

$$\partial_t u = D_u \Delta u + R_u(u, v), \quad \partial_t v = D_v \Delta v + R_v(u, v)$$

where non-linear reaction terms R and differential diffusion rates ($D_v > D_u$) lead to complex Turing patterns, testing the model's capacity to maintain spectral fidelity over long rollouts.

A.3. BubbleML

BubbleML (Hassan et al., 2023) represents the most physically complex benchmark in our evaluation, involving multiphase flows with heat transfer and phase change. The simulations use FC-72 (perfluorohexane) as the working fluid.

Governing Equations. The system solves the non-dimensionalized incompressible Navier-Stokes equations coupled with the energy equation. The liquid (l) and vapor (v) phases are tracked using a level-set function ϕ, where the zero level set $\phi = 0$ represents the interface Γ. The

dynamics evolve according to:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho'} \nabla P + \nabla \cdot \left[\frac{\mu'}{\rho' \mathrm{Re}} \nabla \mathbf{u} \right] + \frac{\mathbf{g}}{\mathrm{Fr}^2} + \mathbf{S}_{\mathbf{u}}^{\Gamma} + \mathbf{S}_P^{\Gamma}$$
$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T = \nabla \cdot \left[\frac{\alpha'}{\mathrm{RePr}} \nabla T \right] + S_T^{\Gamma}$$

Here, $\mathbf{u}$ is velocity, P is pressure, and T is temperature. The system dynamics are controlled by key dimensionless numbers: Reynolds number (Re), Prandtl number (Pr), and Froude number (Fr). The terms ρ', μ', α' denote density, viscosity, and thermal diffusivity scaled by their liquid-phase values.

Crucially, the terms $\mathbf{S}^{\Gamma}$ represent singular source terms at the interface Γ:

- $\mathbf{S}_{\mathbf{u}}^{\Gamma}$ accounts for surface tension forces, modeled via the Weber number (We).
- $\mathbf{S}_P^{\Gamma}$ accounts for the pressure jump induced by phase change.
- S_T^{Γ} represents the latent heat source/sink during evaporation or condensation.

Dataset Configurations. We utilize four datasets representing two boiling regimes:

- **Pool Boiling (`PB-Gravity`, `PB-Subcooled`):** Simulates fluid confined in a tank with a bottom heater (resembling nuclear waste cooling). The `PB-Subcooled` variant introduces a temperature deficit in the bulk liquid to suppress bubble growth.
- **Flow Boiling (`FB-Gravity`, `FB-VelScale`):** Models fluid pumped through a channel containing a heater (resembling data center GPU liquid cooling). These datasets feature extreme aspect ratios (e.g., 1600×160) and complex interactions between bulk momentum and buoyancy-driven bubbles.

A.4. Evaluation Metrics

To provide a comprehensive assessment of physical fidelity, we employ four distinct metrics targeting prediction accuracy, physical structure preservation, frequency consistency, and physical conservation. Let $U \in \mathbb{R}^{N \times C \times H \times W}$ denote the ground truth flow field batch and $\hat{U}$ denote the prediction.

Root Mean Squared Error (RMSE). We measure the absolute pixel-level deviation aggregated globally over the entire dataset. This formulation ensures that the metric is insensitive to batch size variations and penalizes large outliers effectively:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N \cdot C \cdot H \cdot W} \sum_{i,c,x,y} \left(U - \hat{U} \right)^2}$$

Relative Vorticity Error (Vor). To evaluate the capture of rotational dynamics and small-scale structures, we compute the relative Frobenius norm error of the vorticity field $\omega = \nabla \times U$. Using relative error accounts for varying flow magnitudes across different samples:

$$\mathrm{VorError} = \frac{\|\omega - \hat{\omega}\|_F}{\|\omega\|_F} = \frac{\sqrt{\sum (\omega - \hat{\omega})^2}}{\sqrt{\sum \omega^2}}$$

where $\|\cdot\|_F$ denotes the Frobenius norm over the spatial dimensions and batch.

Spectral Error (Spec). Standard metrics often ignore high-frequency details due to the spectral bias of neural networks. We compute the error in the frequency domain using the Log Spectral Distance to balance low- and high-frequency contributions:

$$\mathrm{SpecError} = \frac{1}{N} \sum_{i=1}^{N} \Big\| \log(|\mathcal{F}(U_i)| + \epsilon) - \log\Big(|\mathcal{F}(\hat{U}_i)| + \epsilon \Big) \Big\|_2$$

where $\mathcal{F}$ denotes the 2D Fast Fourier Transform (FFT) applied spatially, and $\epsilon = 10^{-8}$ ensures numerical stability.

Divergence Error (Div). For incompressible flows, mass conservation dictates that the divergence $\nabla \cdot U$ should be zero. We measure the root mean squared deviation of the predicted divergence:

$$\mathrm{DivError} = \sqrt{\frac{1}{N \cdot H \cdot W} \sum_{i,x,y} \Big((\nabla \cdot U) - (\nabla \cdot \hat{U}) \Big)^2}$$

Here, $\nabla \cdot U$ is computed using second-order central finite differences for interior points.

Table 3. Detailed specifications of the datasets. **Size** denotes the total storage size of the dataset in Gigabytes (GB). The "Physics" column references the specific governing equations defined in the text. Note: Turb-M0.1 is excluded from the main evaluation results.

Suite	Dataset	Resolution	Physics / PDE	Boundary Conditions	Size (GB)
CFDBench	`Cavity`	64×64	Incompressible NS (Eq. A.1)	Wall	**0.8**
	`Tube`	64×64	Incompressible NS (Eq. A.1)	Inflow/Outflow	**0.3**
	`Dam`	64×64	Multiphase (VOF)	Wall	**1.3**
	`Cylinder`	64×64	Incompressible NS (Eq. A.1)	Inflow/Outflow	**12.5**
PDEBench	`Diff-React`	128×128	Reaction-Diffusion	Periodic	**12.3**
	`Rand-M0.1`	128×128	Compressible NS ($M < 0.1$)	Periodic	**51.2**
	`Rand-M1.0`	128×128	Compressible NS ($M \leq 1.0$)	Periodic	**51.2**
	`Turb-M0.1`	512×512	Compressible Turbulence	Periodic	**82.0**
BubbleML	`FB-Gravity`	1600×160	Flow Boiling	Wall/Outflow	**16.1**
	`PB-Gravity`	512×512	Pool Boiling	Wall	**24.2**
	`PB-Subcooled`	384×384	Pool Boiling	Wall	**15.4**
	`FB-VelScale`	1244×160	Flow Boiling	Wall/Outflow	**15.7**

B. Implementation Details

B.1. Codebase

FlowForge is implemented in Python with approximately 10k lines of code. We will release the full code upon publication.

B.2. Architectures and Backend

FlowForge Predictor and Backend Model. Once the plan fixes what information is visible (current-frame local features, bounded predecessor context from earlier stages, and static site features), we instantiate G_θ as a compact, fixed predictor across all benchmarks: a 4-layer MLP with $\approx 6 \times 10^4$ parameters. Given local features, the model computes

$$\big[\Phi(U_t, \sigma(i)),\ \Psi(Y; i),\ \mathrm{pos}(\sigma(i))\big] \ \mapsto\ \widehat{y}_i \in \mathbb{R}^c.$$

This emphasizes the core design message: reliability and controllability come from the staged information flow enforced by the plan and executor, rather than from global attention or deep encoder–decoder stacks.

Baselines. For U-Net, FNO, and DeepONet, we use the reference implementations and default architectures provided by the corresponding benchmarks/datasets.

B.3. Hardware and Training Protocol

All experiments were run on a single NVIDIA RTX 3090 GPU (24 GB). Unless otherwise specified, we train all models with Adam (learning rate 10^{-3}), MSE loss, and a StepLR schedule for up to 150 epochs, with a wall-clock cap of 48 hours. Batch size is chosen per method/dataset to maximize GPU utilization without exceeding VRAM.

B.4. Plan-Aligned Training Inputs

During inference, the executor forms each input from the current-frame features and a bounded neighborhood gathered from the partially rewritten buffer, with the guarantee that all gathered entries come from earlier stages (Section 3.2). To avoid training–inference mismatch, we reuse the same plan-defined neighborhood index lists when constructing training inputs. When those indices would read from the inference-time buffer, we instead read from the ground-truth fixed-horizon output sequence $\mathbf{v}^\star_{1:N}$, but only at indices that are valid under the stage order. This preserves the executor's information constraints while avoiding the instability of feeding intermediate model predictions during training.

B.5. Inference Efficiency and Scalability

CUDA Graphs. We execute FlowForge with CUDA Graphs. We capture the full inference workflow into a static CUDA graph and replay it for each input, reducing CPU launch overhead to a single replay call and mitigating GPU idle time caused by frequent kernel launches. For fair comparison, all baselines are also optimized with CUDA Graphs using the same approach.

Coarse-to-fine execution. For very high-resolution grids, we optionally reduce the effective sequence length by running FlowForge on a downscaled field and then upsampling the prediction back to the native grid. Let $\mathcal{D}_\rho$ downscale by factor ρ per axis and $\mathcal{R}_\rho$ upsample back. We compute:

$$\widehat{U}^{\mathrm{coarse}}_{t+\tau} = \textsc{FlowForge}\big(\mathcal{D}_\rho(U_t)\big),$$
$$\widehat{U}_{t+\tau} = \mathcal{R}_\rho\big(\widehat{U}^{\mathrm{coarse}}_{t+\tau}\big).$$

B.6. Ordering Schemes for Tokenization

Let Ω denote the set of grid coordinates (2D/3D) with $N = |\Omega|$. An ordering is a bijection $\sigma : \{1, \ldots, N\} \to \Omega$.

Outward Spiral. Let c be the integer-rounded grid center. Define $\|p-c\|_\infty = \max_j |p_j - c_j|$. Outward Spiral ordering enumerates sites by increasing Chebyshev radius:

$$\sigma \;=\; \bigcup_{r=0}^{r_{\max}} \{\, p \in \Omega \mid \|p-c\|_\infty = r \,\},$$

where ties within the same radius are broken by a fixed deterministic scan over the bounding box of the shell (and duplicates are skipped).

Raster Scan. Lexicographic (row-major) order: 2D: (x, y) with y outer and x inner. 3D: (x, y, z) with z outer, then y, then x.

Random. A uniform random permutation of Ω, generated once with a fixed RNG seed and reused.

Hilbert Curve. Embed the grid into the smallest power-of-two cube/box with side $M = 2^{\lceil \log_2(\max(X,Y(,Z)))\rceil}$. Assign each $p \in \Omega$ a Hilbert Curve index $h(p) \in [0, M^d)$ and sort by $h(p)$; embedded out-of-bounds points are ignored.

Orders that preserve spatial locality (e.g., Outward Spiral and Raster Scan) tend to behave similarly, whereas orders that fail to expose nearby predecessors early under a fixed predecessor context budget can degrade.

C. Baseline Hyperparameter Search

For baseline settings that are not reported in the original benchmark suites, we run a compact architecture sweep around the canonical public implementations while keeping the optimizer, scheduler, and overall 48-hour training budget unchanged. For CNO and UNO on CFDBench, each candidate first receives a 5-epoch proxy run, and we promote the setting with the best validation RMSE to the full training run. For U-Net on BubbleML, we instead run every candidate width for 25 epochs on the target BubbleML case and select the width with the lowest validation RMSE.

We retain the implementation names `kernel_size` for CNO and `uno_kernel_size` for UNO; both denote the local operator kernel width used inside the corresponding architecture.

C.1. CNO on CFDBench

Parameters. We search the three CNO knobs that most directly shape capacity and local aggregation. `hidden_dim` sets the feature width after the lifting layer and therefore the channel capacity carried by every operator block, making it the main width parameter. `depth` is the number of stacked operator blocks, so it determines how much iterative feature refinement the model can perform under a fixed training budget. `kernel_size` controls the local receptive field of the depthwise operator in each block and is therefore the key parameter for how much near-field spatial context each update can absorb. We do not tune padding separately, because the implementation deterministically sets `cno_padding` = (`kernel_size` - 1) / 2 to preserve spatial resolution.

Search Spaces and Results. We sweep:

$$\texttt{hidden_dim} \in \{64, 96, 128\},$$
$$\texttt{depth} \in \{4, 6, 8\},$$
$$\texttt{kernel_size} \in \{3, 5\}.$$

For each CFDBench dataset, we choose the setting with the lowest validation RMSE among the 5-epoch proxy runs. The final selected configurations are:

Dataset	depth	hidden_dim	kernel_size
Tube	4	64	3
Cylinder	6	96	5
Dam	8	128	5
Cavity	8	128	5

C.2. UNO on CFDBench

Parameters. We search the UNO parameters that most strongly shape its multiscale hierarchy. `uno_base_dim` sets the channel width at the finest scale and therefore the width of the full encoder–decoder stack. `uno_levels` controls how many resolutions the model traverses, so it changes both receptive-field growth and total multiscale capacity. `uno_kernel_size` determines the local receptive field inside each operator block and therefore how aggressively each stage aggregates nearby context. We keep `uno_depth_per_level`=2 and `uno_bottleneck_depth`=4 fixed. These parameters regulate how many blocks are repeated within each scale and at the bottleneck, so holding them constant keeps the sweep focused on width, hierarchy depth, and local context size instead of exploding the search space.

Search Spaces and Results. We sweep:

$$\texttt{uno_base_dim} \in \{48, 64, 96\},$$
$$\texttt{uno_levels} \in \{3, 4\},$$
$$\texttt{uno_kernel_size} \in \{3, 5\},$$

while keeping the following structural parameters fixed:

$$\texttt{uno_depth_per_level} = 2,$$
$$\texttt{uno_bottleneck_depth} = 4.$$

For each CFDBench dataset, we again select the setting with the lowest validation RMSE from the 5-epoch proxy runs. The resulting chosen configurations are:

Dataset	base_dim	levels	kernel_size
Tube	48	3	3
Cylinder	64	4	5
Dam	96	4	5
Cavity	96	4	5

C.3. U-Net on BubbleML

Parameters. For BubbleML we keep the U-Net topology used in our experiments fixed: four encoder blocks separated by 2×2 max-pooling layers, a bottleneck block, and four mirrored decoder blocks. Each decoder stage first upsamples with a 2×2 transposed convolution, concatenates the corresponding encoder skip feature, and then applies the decoder block, followed by a final 1×1 output projection. Under this implementation, `init_features` is the only exposed width knob; it sets the channel count of the first encoder block. It is therefore the main architecture parameter for trading off model capacity against memory footprint, while the overall encoder–decoder topology remains fixed.

Search Spaces and Results. We search only:

$$\texttt{init_features} \in \{32, 64, 128\}.$$

For each BubbleML case requiring a U-Net width choice, we train all three candidate widths for 25 epochs on that target case and choose the width with the lowest validation RMSE. Thus the BubbleML U-Net width is selected from target-dataset validation performance, not from a `Diff-React` proxy or transferred setting. This procedure selects `init_features`=64 for the BubbleML cases reported in the main comparison.

D. Additional Results and Analysis

D.1. Multi-Step Autoregressive Rollout Analysis

We evaluate long-horizon autoregressive temporal rollouts by iteratively feeding model predictions back as inputs and tracking error metrics over time (Figure 7) on PDEBench Diff-React. While all methods exhibit growing RMSE with rollout length, FlowForge remains comparable to FNO and more stable than U-Net, indicating that the staged local rollout does not introduce additional numerical instability when used inside an autoregressive temporal rollout. The most pronounced difference appears in physical consistency: FlowForge maintains near-zero divergence error throughout the temporal rollout, whereas FNO and U-Net exhibit steadily increasing divergence, reflecting accumulation of global inconsistency. FlowForge also preserves vorticity and spectral content at levels comparable to or better than baselines, despite relying on strictly local updates. Overall, these results show that the staged local rollout mechanism yields stable long-horizon behavior and effectively limits constraint drift under autoregressive use.

D.2. Analysis of Nearby Predecessor Availability

In Section 4.4, FLOWFORGE shows similar performance across several traversal orders, including Outward Spiral, Raster Scan, Hilbert Curve, and Random. This result is consistent with the locality motivation in Section 2.2 and Appendix F.4: the predictor is primarily sensitive to nearby predecessor availability, not to the global geometry of the traversal path. We quantify this effect using the distance from each target site to its nearest already-written predecessor.

Let $\mathcal{P}_q = \{x_{\sigma(1)}, \ldots, x_{\sigma(q-1)}\}$ denote sites written before rollout index q. We define the **nearest preceding neighbor distance**

$$d_q = \min_{j<q} \|x_{\sigma(q)} - x_{\sigma(j)}\|_2.$$

Smaller d_q means a closer already-written predecessor. We simulate traversal on a 16×16 grid and record d_q; in FLOWFORGE each target uses an irregular list of up to H predecessors within R rings, not a dense patch.

Figure 8 plots a moving average of d_q for each order. Two phases appear:

1. **Warm-up ($q < 30$):** Structured orders expose adjacent predecessors immediately, whereas Random initially selects scattered sites and has larger nearest-predecessor distances.

2. **Dense-predecessor ($q \geq 30$):** Once a modest number of sites has been written, the predecessor set becomes dense over the grid. The nearest already-written site to a new target is then typically within one grid spacing, even under Random ordering. The predecessor context gathered by the predictor becomes similar across traversal orders.

The small gap between Random and structured orders therefore separates local dependency radius and coverage from global traversal geometry. Under the fixed predecessor context budget, many schedules provide adequate nearby predecessor coverage after a short warm-up. FLOWFORGE benefits from nearby output-field predecessor conditioning; traversal geometry is a secondary design choice once such coverage is available.

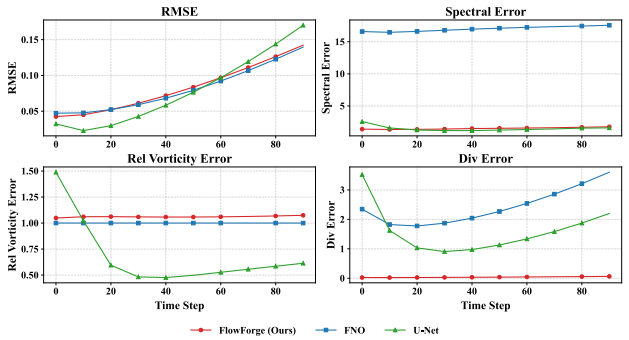

Figure 7. **Multi-step autoregressive rollout on PDEBench Diff-React.** We report RMSE, spectral error, relative vorticity error, and divergence error over rollout time steps for FlowForge (ours), FNO, and U-Net.

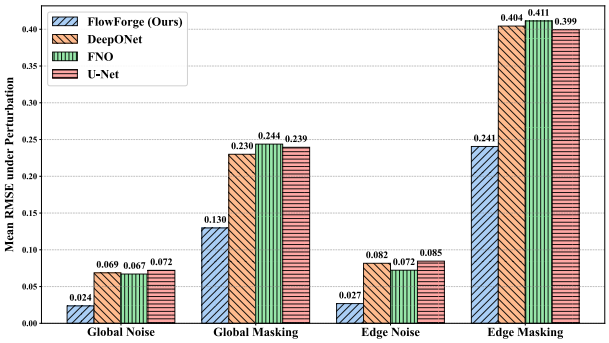

Figure 9. **Robustness on CFDBench-Cylinder.** Average RMSE across the perturbation settings in Section 4.2.

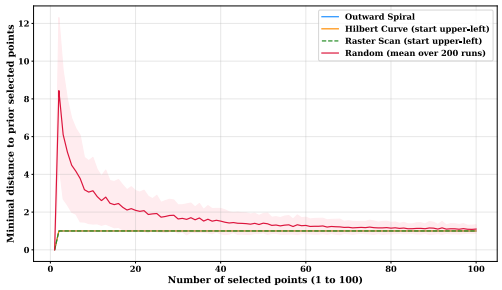

Figure 8. **Nearest preceding-neighbor distance d_q on a 16×16 grid.** Structured orders start with d_q=1; Random is larger early but converges to the same local-neighbor regime once the written set is dense.

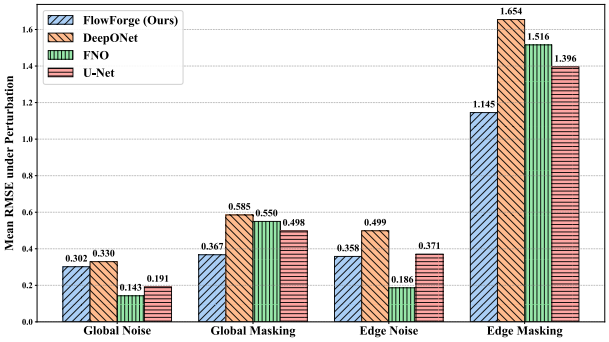

Figure 10. **Robustness on CFDBench-Cavity.** Average RMSE across the perturbation settings in Section 4.2.

D.3. Complete Robustness Test Results

This section presents comprehensive robustness evaluations across all four CFDBench datasets, complementing the findings in Section 4.2. We report the average loss computed over distinct severity levels: noise standard deviation $\sigma \in \{0.01, 0.05, 0.1\}$, edge width $w \in \{1, 5, 10\}$, and masking rate $k \in \{1\%, 5\%, 10\%\}$. As shown in Figures 9, 10, 11, and 12, FlowForge achieves the lowest or second-lowest mean RMSE in almost all perturbation settings across the four CFDBench datasets. These comprehensive results reinforce the analysis in Section 4.2 and indicate that FlowForge is more resilient to input perturbations overall than the baselines.

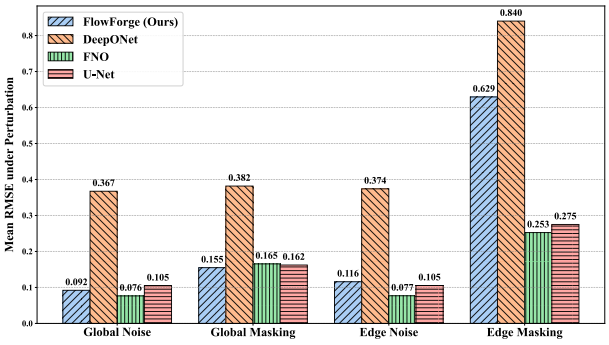

Figure 11. **Robustness on CFDBench-Tube.** Average RMSE across the perturbation settings in Section 4.2.

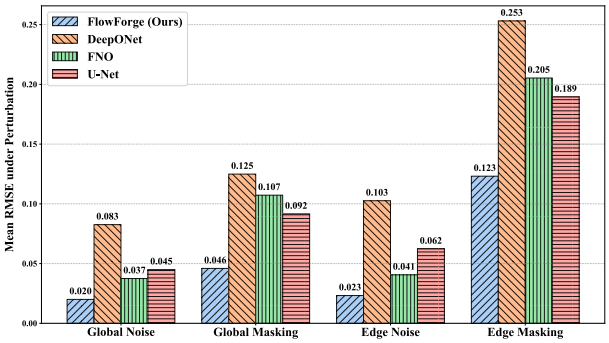

Figure 12. **Robustness on CFDBench-Dam.** Average RMSE across the perturbation settings in Section 4.2.

E. Extended Case Studies and Qualitative Analysis

In Section 4.1, we quantitatively analyzed the flow dynamics on the `FB-Gravity` dataset. To demonstrate the versatility of FLOWFORGE across diverse flow regimes and verify its stability under corruption, we provide additional qualitative comparisons here. We focus on two complementary aspects: (1) prediction fidelity in complex multiphase flows involving sharp interfaces and phase changes, and (2) visual verification of error containment mechanisms under severe input degradation.

E.1. Fidelity on Multiphase Benchmarks

We examine two challenging scenarios from the CFDBench and BubbleML suites, characterized by rapid topological changes and multi-scale interfacial dynamics.

Dam Break (Multiphase VOF). Figure 13 visualizes the Volume-of-Fluid (VOF) simulation of a collapsing fluid column. This regime poses a significant challenge due to the presence of sharp gas-liquid interfaces and the generation of fine-scale vorticity during impact.

To scrutinize performance in physically volatile regions, the visualization combines macroscopic velocity streamlines with a microscopic error analysis. We define a region of intense vorticity gradients (marked by the green bounding box) and overlay the absolute vorticity error heatmap ($|\omega_{\text{pred}} - \omega_{\text{gt}}|$) to reveal localized discrepancies.

As observed in Figure 13, the performance gap manifests differently across model families. U-Net and DeepONet exhibit substantial deviations, marked by intense red hotspots in the error heatmaps, indicating a struggle to capture the complex multi-fluid mixing dynamics. The comparison between FLOWFORGE and FNO is more nuanced yet physically critical. While FNO achieves a low global error magnitude comparable to FLOWFORGE (indicated by similarly faint heatmaps), a closer inspection of the streamlines reveals a key divergence: FNO suffers from *over-smoothing* at the vortex core, failing to resolve the tight rotational structures. In contrast, FLOWFORGE precisely preserves these fine-grained dynamics. This confirms that the staged local rollout does not merely lower pixel-wise error but, more importantly, prevents the loss of high-frequency physical information that plagues monolithic global predictors.

Flow Boiling (FB-VelScale). Figure 14 depicts the void fraction field in a heated channel flow. This regime involves small-scale bubble nucleation and advection, governed by the interplay between buoyancy and inlet velocity perturbations.

We highlight two distinct regions to illustrate scale-dependent performance.

- **Inlet Region (Box 1):** Located near the channel entrance, flow dynamics are relatively smooth and dominated by inlet boundary conditions. FLOWFORGE successfully captures this laminar behavior. In contrast, U-Net and FNO introduce spurious oscillations, leading to significant local errors.
- **Nucleation Region (Box 2):** Located near the heated wall, this area features vigorous bubble formation and detachment, characterized by high-frequency spatial variations in void fraction. Here, FLOWFORGE effectively preserves individual bubble structures. Baselines, particularly FNO and U-Net, exhibit a washout effect—blurring adjacent bubbles into a unified field due to the spectral bias of global operators—which is detrimental for accurate heat transfer estimation.

E.2. Visualizing Robustness Mechanisms

To complement the aggregate robustness scores in Section 4.2, we visualize the flow fields under two distinct corruption protocols to empirically validate the error containment hypothesis.

Global Block Masking. We apply a random binary block mask to the input frame of the `Cavity` dataset, zeroing out approximately 10% of the spatial domain (Figure 15).

For global architectures like U-Net and FNO, the missing blocks introduce high-frequency discontinuities that propagate globally via convolution or Fourier mixing. This results in spectral poisoning: artifacts teleport from the masked regions to the unmasked center and distort the primary vortex structure. The rectangular regions of high error visible in the baseline heatmaps (Figure 15) directly correspond to this non-local error propagation.

In contrast, due to bounded local information flow in the staged predecessor graph, FLOWFORGE limits how quickly errors from masked sites can spread across subsequent

stages. Consequently, the majority of the flow field remains structurally intact, demonstrating that FLOWFORGE localizes many defects instead of letting them immediately destroy global physical consistency.

Global Gaussian Noise Injection. We inject additive Gaussian noise ($\mathcal{N}(0, 0.1^2)$) into some random blocks of the input velocity field for the `Cylinder` dataset (Figure 16).

Global models tend to amplify high-frequency noise, interpreting it as physical turbulence. This results in disordered streamlines around the cylinder and a distorted wake structure, as shown in the baseline predictions. FLOWFORGE, however, leverages its local aggregation mechanism as an implicit spatial filter. By conditioning the prediction on a bounded neighborhood of previous estimates, FLOWFORGE effectively averages out the incoherent zero-mean noise, recovering a clean Karman vortex street that closely matches the ground truth.

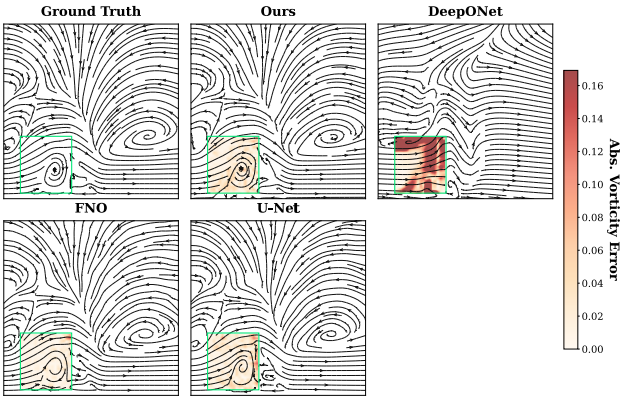

Figure 13. **Qualitative results on CFDBench-Dam.** Visualization of the fluid phase fraction and vorticity error. While FNO tends to over-smooth tight vortex cores, FLOWFORGE accurately captures the sharp interface and rotational dynamics of the water column as it impacts the obstacle.

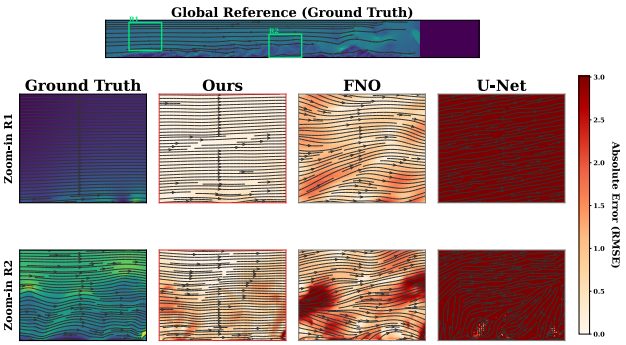

Figure 14. **Qualitative results on BubbleML FB-VelScale.** Visualization of gas phase distribution (void fraction). FLOWFORGE preserves the integrity of small-scale bubbles (Box 2) that are prone to washout in global baselines, while maintaining stability in smooth inlet regions (Box 1).

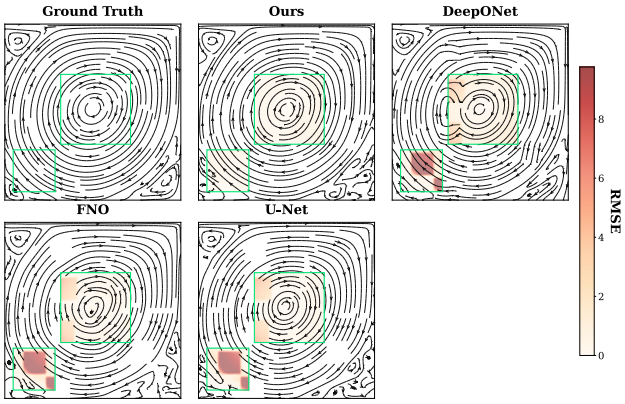

Figure 15. **Robustness to Global Block Masking (Cavity, ≈10% masked).** The input contains zeroed-out blocks. Baselines exhibit non-local error propagation (artifacts appearing far from masks), whereas FLOWFORGE keeps errors localized near the corrupted regions, preserving the central vortex details.

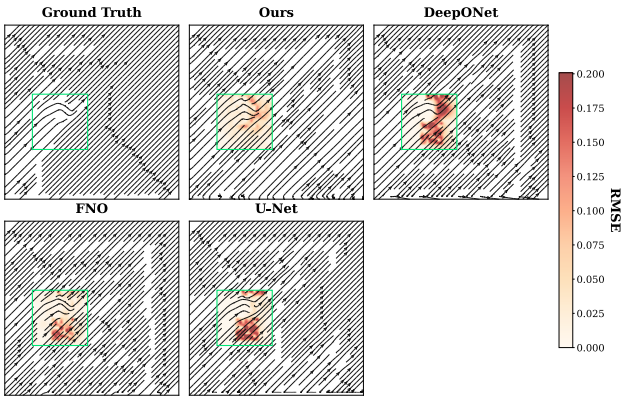

Figure 16. **Robustness to Localized Gaussian Noise (Cylinder, $\sigma = 0.1$).** Comparison of streamlines under noisy input. Global models amplify noise into spurious turbulence. FLOWFORGE acts as a conditional filter, recovering a smooth and physically consistent wake.

F. Additional Theoretical Details and Proofs

F.1. Notation and Standing Assumptions

- **Domain and norms.** The spatial domain is either the d-torus $\Omega = \mathbb{T}^d$ (periodic) or a bounded Lipschitz domain with homogeneous Neumann (no-flux) boundary conditions. We write $\langle f, g\rangle = \int_\Omega f \cdot g$ and $\|f\|_2^2 = \langle f, f\rangle$.

- **Discrete field.** The fixed-horizon output field is represented on a uniform grid as $U_{t+\tau} \in \mathbb{R}^{N\times c}$ (or equivalently vectorized in $\mathbb{R}^{Nc}$), with grid spacing Δx per axis.

- **Ordering and stages.** An *ordering scheme* induces an order σ of the spatial indices $\{1, \dots, N\}$ and a stage partition $S_1, \dots, S_K$ (disjoint union).

- **Stage-wise rollout map.** At inference, stage s outputs

$$Y_{S_s} \;=\; G_\theta\big(U_t,\; Y_{S_{<s}}\big), \qquad S_{<s} := \{S_1, \dots, S_{s-1}\}.$$

  The predictor G_θ is Fréchet differentiable in its second argument and is *masked*: it does not access Y_{S_s} nor any Y_{S_r} with $r > s$.

- **Norms for block vectors.** For block vectors $z = (z_1, \dots, z_m)$ we use the Euclidean norm $\|\cdot\|$ and the induced operator norm for Jacobians.

- **Helmholtz projection (incompressible case).** Let

$$\mathcal{V} = \left\{ \begin{array}{l} v \in L^2(\Omega; \mathbb{R}^d) \;: \nabla \cdot v = 0, \\ \qquad\qquad\qquad v \cdot n|_{\partial\Omega} = 0 \text{ (if } \partial\Omega \neq \emptyset) \end{array} \right\},$$

  and let $P : L^2(\Omega; \mathbb{R}^d) \to \mathcal{V}$ denote the L^2-orthogonal projection (Helmholtz–Hodge projection): $Pv = v - \nabla\phi$ where ϕ solves $\Delta\phi = \nabla \cdot v$ with compatible BCs (and zero-mean in the periodic case).

F.2. Causality from Masking: Triangular Jacobian

F.2.1. Block Lower–Triangular Context Jacobian

Theorem F.1 (Block lower–triangular context Jacobian)**.** *Define $F : \mathbb{R}^{Nc} \to \mathbb{R}^{Nc}$ blockwise by*

$$\begin{aligned} F_{S_1}(Y) &= G_\theta(U_t, \emptyset), \\ F_{S_2}(Y) &= G_\theta(U_t, Y_{S_1}), \\ &\;\;\vdots \\ F_{S_s}(Y) &= G_\theta(U_t, Y_{S_{<s}}). \end{aligned}$$

Let $J^{\text{ctx}} := \frac{\partial F}{\partial Y}$ in the stage-block layout. Then

$$J^{\text{ctx}}_{s,r} \;=\; \frac{\partial F_{S_s}}{\partial Y_{S_r}} \;=\; 0 \qquad \textit{for all } r \geq s,$$

i.e., J^{ctx} is strictly block lower–triangular. In particular, $(J^{\text{ctx}})^K = 0$.

Proof. By construction, F_{S_s} depends only on $(Y_{S_1}, \dots, Y_{S_{s-1}})$. Hence for any $r \geq s$, F_{S_s} is independent of Y_{S_r} and the Fréchet partial derivative with respect to Y_{S_r} is the zero operator, giving $J^{\text{ctx}}_{s,r} = 0$. Strict lower–triangular block matrices are nilpotent with index at most K. □

Corollary F.2 (One-way error flow)**.** *A perturbation injected into Y_{S_r} cannot affect any Y_{S_q} with $q \leq r$. Error can only propagate to later stages.*

Proof. Immediate from $J^{\text{ctx}}_{q,r} = 0$ for $q \leq r$. □

F.3. Stage Sensitivity and Error Propagation

F.3.1. Lipschitz Product Bound

Assumption F.3 (Stage Lipschitzness)**.** For each stage s there exists $L_s \geq 0$ such that for all context vectors a, b,

$$\big\|G_\theta(U_t, a) - G_\theta(U_t, b)\big\| \;\leq\; L_s \, \|a - b\|.$$

Proposition F.4 (Geometric damping of upstream perturbations)**.** *Fix $r < s$. Consider the composite mapping from Y_{S_r} to Y_{S_s} induced by executing stages $r+1, \dots, s$ with all other inputs fixed. Then*

$$\left\| \frac{\partial Y_{S_s}}{\partial Y_{S_r}} \right\| \;\leq\; \prod_{q=r+1}^{s} L_q.$$

Equivalently, any small perturbation δY_{S_r} obeys

$$\|\delta Y_{S_s}\| \;\leq\; \Big(\prod_{q=r+1}^{s} L_q \Big) \, \|\delta Y_{S_r}\|.$$

Proof. Define $H_{r+1}(z) = G_{r+1}(U_t, (Y_{S_{<r}}, z))$, which is L_{r+1}-Lipschitz by Assumption F.3. Similarly define $H_{r+2}, \dots, H_s$ for successive stages. The dependence from Y_{S_r} to Y_{S_s} is the composition $Y_{S_s} = H_s \circ \cdots \circ H_{r+1}(Y_{S_r})$. Lipschitz constants multiply under composition, yielding the bound. □

Corollary F.5 (Monotone decay under easier later stages)**.** *If $L_{q+1} \leq L_q$ and often $L_q < 1$, then $\prod_{q=r+1}^{s} L_q$ decays geometrically in $s - r$. A widening schedule (smaller K) further reduces worst-case propagation depth.*

F.3.2. Stage-Wise Attenuation (Sufficient-Condition Analysis)

Theorem F.6 (Stage-wise error attenuation under contraction)**.** *Let $y^\star_{S_s}$ be the predictions obtained when earlier contexts equal their ground-truth values, and let $\hat{y}_{S_s}$ be the actual predictions obtained when earlier stages may carry errors. Suppose Assumption F.3 holds and $\max_s L_s \leq \lambda < 1$. Then for every s,*

$$\big\|\hat{y}_{S_s} - y^\star_{S_s}\big\| \;\leq\; \sum_{r=1}^{s-1} \Big(\prod_{q=r+1}^{s} L_q \Big) \, \big\|\hat{y}_{S_r} - y^\star_{S_r}\big\|.$$

In particular,

$$\max_{j \leq s} \big\|\hat{y}_{S_j} - y^\star_{S_j}\big\| \;\leq\; \frac{\lambda}{1-\lambda} \max_{j<s} \big\|\hat{y}_{S_j} - y^\star_{S_j}\big\|.$$

Thus newly generated tokens attenuate prior errors, and total error cannot blow up across stages.

Proof. By definition, $\hat{y}_{S_s} - y^\star_{S_s} = G_\theta(U_t, \hat{y}_{S_{<s}}) - G_\theta(U_t, y^\star_{S_{<s}})$. Assumption F.3 and the triangle inequality

imply $\|\hat{y}_{S_s} - y^\star_{S_s}\| \le L_s \|\hat{y}_{S_{<s}} - y^\star_{S_{<s}}\|$. Expand block-by-block and apply Proposition F.4; then bound the geometric series by $\lambda/(1-\lambda)$. □

F.4. Short-Horizon Locality and Coverage Bounds

F.4.1. Kernel Form for Advection–Diffusion

Lemma F.7 (Advection–diffusion fundamental solution)**.** *Consider the scalar advection–diffusion PDE*

$$\partial_t u + a\cdot\nabla u \;=\; \nu\,\Delta u, \qquad a \in \mathbb{R}^d,\ \nu > 0,$$

with sufficiently regular $u(\cdot,t)$ and periodic or no-flux BCs. Then

$$u(x, t+\Delta t) \;=\; \int_{\mathbb{R}^d} G_{\Delta t}(y)\; u\big(x - a\,\Delta t - y,\ t\big)\, dy,$$

$$G_{\Delta t}(y) \;=\; \frac{1}{(4\pi\nu\Delta t)^{d/2}}\, e^{-\frac{\|y\|^2}{4\nu\Delta t}}.$$

Proof. Let $w(x,t) = u(x+at,t)$. Then $\partial_t w = a\cdot\nabla u + \partial_t u = \nu\,\Delta u = \nu\,\Delta w$. Hence $w_t = \nu\Delta w$ and $w(\cdot, t+\Delta t) = G_{\Delta t} * w(\cdot,t)$. Rewriting in terms of u yields the claim. □

F.4.2. Tail Mass Inside a Radius

Lemma F.8 (Mass inside a radius)**.** *For $Y \sim \mathcal{N}(0, 2\nu\Delta t\, I_d)$,*

$$\mathbb{P}\big(\|Y\| \le r\big) \;=\; F_{\chi^2_d}\!\left(\frac{r^2}{2\nu\Delta t}\right),$$

where $F_{\chi^2_d}$ is the CDF of the χ^2 distribution with d degrees of freedom.

Proof. $\|Y\|^2/(2\nu\Delta t) \sim \chi^2_d$ by isotropy. □

F.4.3. Tail-Mass Error Bound for Truncated Dependence

Proposition F.9 (CFL-style coverage ⇒ tail-mass error bound)**.** *Fix $p \in (0,1)$ and let*

$$r_p \;:=\; \sqrt{2\nu\Delta t\; F^{-1}_{\chi^2_d}(p)}$$

so that $\int_{\|y\|\le r_p} G_{\Delta t}(y)\, dy = p$. Define the truncated predictor

$$v(x) \;:=\; \int_{\|y\|\le r} G_{\Delta t}(y)\; u\big(x - a\Delta t - y, t\big)\, dy.$$

Then for any $r \ge r_p$:

$$\text{(i)}\ \|u(\cdot,t+\Delta t) - v\|_\infty \;\le\; (1-p)\,\|u(\cdot,t)\|_\infty,$$
$$\text{(ii)}\ \|u(\cdot,t+\Delta t) - v\|_2 \;\le\; (1-p)\,\|u(\cdot,t)\|_2.$$

Proof. By Lemma F.7, the error equals the kernel tail $e(x) = \int_{\|y\|>r} G_{\Delta t}(y)\, u(x - a\Delta t - y, t)\, dy$. Bounding by $\|u(\cdot,t)\|_\infty$ gives (i) since $\int_{\|y\|>r} G_{\Delta t} = 1 - p$ for $r \ge r_p$. For (ii), Young's inequality yields $\|e\|_2 \le \|G_{\Delta t}\mathbf{1}_{\{\|y\|>r\}}\|_1 \|u(\cdot,t)\|_2 \le (1-p)\|u(\cdot,t)\|_2$. □

F.4.4. Coverage Under Outward Spiral Traversal (Geometry)

Let the Outward Spiral include all grid points within Chebyshev radius R (i.e., ℓ_∞-ball $B_\infty(R\Delta x)$). The Euclidean and Chebyshev balls satisfy

$$B_2(r) \;\subseteq\; B_\infty(r) \;\subseteq\; B_2(\sqrt{d}\, r). \tag{5}$$

Lemma F.10 (Ring coverage vs. Euclidean radius)**.** *If the Outward Spiral has generated R rings, then $B_2(r) \subseteq$ (available context) for all $r \le R\Delta x$. Conversely, if $B_2(r)$ is required, it suffices to take $R \ge \lceil r/\Delta x \rceil$ rings.*

Proof. The available context equals $B_\infty(R\Delta x)$; apply the left inclusion in (5). □

Definition F.11 (Coverage coefficient on a grid)**.** Let the Outward Spiral have generated R rings prior to decoding the token at x. Define

$$\Gamma_p(x) \;:=\; \frac{R\,\Delta x}{\|a\|\,\Delta t + \sqrt{2\nu\Delta t\, F^{-1}_{\chi^2_d}(p)}}.$$

By Lemma F.10, $\Gamma_p(x) \ge 1$ implies $B_2(r_p) \subseteq$ (available context).

Remark F.12 (Discrete coverage with norm equivalence). If the implementation reasons in the ℓ_∞ geometry, one may also use the inclusion $B_2(r) \subseteq B_\infty(r)$; thus $\Gamma_p(x) \ge 1$ (Euclidean) is conservative for Chebyshev coverage. Combining Proposition F.9 with Lemma F.13 yields a practical bound

$$\|u(\cdot,t+\Delta t) - v_{\text{grid}}\|_{\infty,2} \;\le\; (1-p)\,\|u(\cdot,t)\|_{\infty,2} + C\,\Delta x.$$

F.4.5. Discrete Quadrature Error for Truncated Gaussian

Lemma F.13 (Discrete quadrature error for truncated Gaussian)**.** *Assume $u(\cdot,t) \in C^1(\Omega)$ and use a Riemann sum over available grid points in $B_2(r)$ to approximate $v(x)$. Then, uniformly in x,*

$$\big|\, v(x) - v_{\text{grid}}(x) \,\big| \;\le\; C\,\Delta x,$$

where C depends on $\|u(\cdot,t)\|_{C^1}$, ν, Δt, and d (Gaussian smoothness).

Proof. The truncated Gaussian is smooth and rapidly decaying; a standard first-order Riemann-sum estimate for Lipschitz integrands gives $O(\Delta x)$ uniform error. □

F.4.6. Hyperbolic and Multiphysics Extensions

For a linear symmetric hyperbolic system $\partial_t w + \sum_{j=1}^{d} A_j \partial_{x_j} w = 0$ with wave speed bound $c_{\max}$, the domain of dependence over Δt is contained in $B_2(c_{\max}\Delta t)$. For advection–diffusion–heat (with thermal diffusivity κ), a conservative requirement is

$$r_{\text{req}} \;\approx\; c_{\max}\Delta t \;+\; \sqrt{2\nu\Delta t\, F^{-1}_{\chi^2_d}(p)} \;+\; \sqrt{2\kappa\Delta t\, F^{-1}_{\chi^2_d}(p)}.$$

This can be used in Definition F.11 by replacing $\|a\|$ with $c_{\max}$ and augmenting the diffusive radii.

F.5. Non-Expansiveness and Persistence of Coverage Bound

We now make precise that the coverage bound is preserved by the pressure projection.

Lemma F.14 (Helmholtz projection is non-expansive)**.** *$P : L^2(\Omega;\mathbb{R}^d) \to L^2(\Omega;\mathbb{R}^d)$ is an L^2-orthogonal projection onto $\mathcal{V}$. Hence $\|Pv - Pw\|_2 \le \|v - w\|_2$ and, in particular, $\|Pv\|_2 \le \|v\|_2$. Moreover, there exists a domain-dependent constant C_Ω such that*

$$\|(I-P)v\|_2 = \|\nabla\phi\|_2 \;\le\; C_\Omega\, \|\nabla\cdot v\|_2,$$

where $\Delta\phi = \nabla\cdot v$ (elliptic regularity).

Proof. Orthogonality is standard: decompose $v = w + \nabla\phi$ with $w \in \mathcal{V}$ and ϕ solving the Poisson problem; P is the orthogonal projector $v \mapsto w$ so it is non-expansive in L^2. The estimate $\|\nabla\phi\|_2 \le C_\Omega\|\Delta\phi\|_{H^{-1}} \le C_\Omega\|\nabla\cdot v\|_2$ follows from elliptic regularity on Ω with the stated BCs. □

Proposition F.15 (Coverage bound persists under projection)**.** *Let $u^*(\cdot, t+\Delta t)$ denote the advective–diffusive update and let the true incompressible update be $u(\cdot, t+\Delta t) = Pu^*(\cdot, t+\Delta t)$. Let v be the truncated predictor from Proposition F.9 and $\hat{u} := Pv$. Then*

$$\begin{aligned}\|u(\cdot,t+\Delta t) - \hat{u}\|_2 &= \|Pu^* - Pv\|_2\\ &\le \|u^* - v\|_2\\ &\le (1-p)\,\|u(\cdot,t)\|_2.\end{aligned}$$

In particular, the projection step cannot worsen *the coverage error bound; it can only improve it.*

Proof. Apply Lemma F.14 to v and u^*, then Proposition F.9(ii). □

F.6. Runtime Model: Idealized Makespan

Proposition F.16 (Latency as a tunable knob)**.** *Assume each token evaluation takes constant time t_{tok}, tokens within a stage are independent, and up to w_s tokens of stage s can be processed in parallel. Then the stage makespan satisfies*

$$T_s \;=\; \left\lceil \frac{|S_s|}{w_s} \right\rceil t_{\text{tok}},$$

and the end-to-end latency is

$$T_{\text{total}} \;=\; \sum_{s=1}^{K} T_s \;=\; \sum_{s=1}^{K} \left\lceil \frac{|S_s|}{w_s} \right\rceil t_{\text{tok}}.$$

Hence, with fixed t_{tok}, latency is directly controlled by the stage widths $\{w_s\}$.

Proof. Within stage s, $|S_s|$ identical jobs on w_s identical parallel servers have optimal makespan $\lceil |S_s|/w_s \rceil t_{\text{tok}}$ (tight list-scheduling bound for identical tasks). Stages are executed sequentially, so the total time is the sum. □