KaleidoFlow: an autonomous data engine for learning flows on diverse geometries

Anonymous authors
Paper under double-blind review

Abstract

Neural operators are emerging as surrogates that predict a flow much faster than a numerical solver. However, a neural operator generalizes only within the *geometry families* it was trained on. This limitation comes from the data. Every existing flow dataset depends heavily on human experts to design every geometry and to verify its mesh, so it is expensive to scale. Thus, each dataset can cover only a single geometry or similar geometries from the same family. We present KaleidoFlow, an autonomous data engine that scales a flow dataset to diverse geometries in three stages, all without human intervention. First, a language model agent continuously proposes new geometry families, together with the geometric parameters and the Reynolds numbers that vary them. Second, the agent designs the mesh of every family and repairs it iteratively until a convergence test verifies its discretization error. Third, OpenFOAM solves every combination of the geometric parameters and the Reynolds number, and each solution is one *trajectory*. The engine has produced 3,948 distinct geometries from 459 families, with 16,512 trajectories. Our results show that (1) the engine **proposes diverse geometries** automatically, covering 10 application categories from cyclones to biomedical vessels, and (2) it **successfully repairs the mesh strategies that fail**, which most of the families require. With this dataset, we build the first benchmark that measures how far a neural operator generalizes across geometry families. On the benchmark, we find that (1) current neural operators **generalize poorly to a new geometry family**, where the error of all nine architectures we evaluate increases by 3× to 6× from G2 to G3, and (2) the error on new families **decreases with the number of training families approximately as a power law**, for a convolutional, a transformer, and a spectral architecture. We will release the engine and the benchmark for future development of foundation neural operators that generalize across diverse geometries.

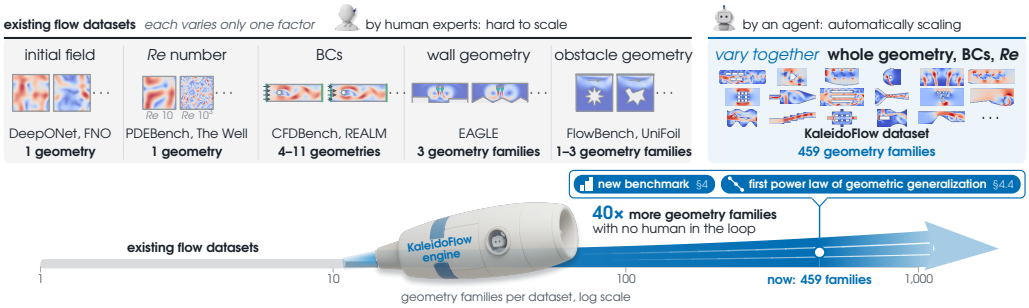

Figure 1: **Comparison of the dataset generated by KaleidoFlow with existing flow datasets.** Each existing dataset varies only one factor on geometries designed by human experts. KaleidoFlow uses a language model agent to vary the geometry, the boundary conditions (BCs), and the Reynolds number together. The axis counts the geometry families of each dataset on a logarithmic scale. The current release contains 459 geometry families, 40× the largest existing count, and yields the benchmark (Sec. 4) and the power law (Sec. 4.4).

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

1 Introduction

Foundation models in language and vision are trained on diverse data and generalize to new tasks without task-specific training (Brown et al., 2020; Kirillov et al., 2023). Inspired by these models, much work in physics simulation is pursuing foundation neural operators (McCabe et al., 2024; Chen et al., 2025; McCabe et al., 2026). Such an operator is expected to learn universal physical features from simulated data (Lu et al., 2021; Li et al., 2021; Kovachki et al., 2023) and to predict new solutions across diverse scenarios much faster than the numerical solver. However, existing flow datasets vary one factor at a time, the Reynolds number (Takamoto et al., 2022; Ohana et al., 2024), the boundary conditions (Luo et al., 2023; Mao et al., 2025), the walls of one hall (Janny et al., 2023), or the obstacle inside one box (Tali et al., 2025; Kanchi et al., 2025). They are insufficient (Fig. 1) because (1) they depend on a human expert to design every geometry and mesh strategy, so scaling the dataset is expensive, and (2) they contain only a single geometry or similar geometries from the same family, so they do not support learning and generalization across diverse geometries. Therefore, how to efficiently scale a flow dataset to diverse geometries is a central problem for the development of foundation neural operators.

This work aims to scale the flow dataset with a language model agent. There are three requirements. First, the agent must continuously propose new geometry families. A family specifies a closed outer boundary with a boundary condition on every segment and the obstacles inside it. It also specifies the geometric parameters and the Reynolds numbers that the family varies. The families must differ in shape and in the number of inlets and obstacles, not only in the geometric parameters of one family. Second, the agent must design a mesh that resolves the regions of high gradient of every geometry at an acceptable cost. The discretization error of the mesh is verified by a convergence test at several resolutions (Roache, 1994; Celik et al., 2008), and a mesh that fails the test must be redesigned. Third, all of this should run without human intervention, because a dataset that depends on an expert is expensive to scale.

We present KaleidoFlow[1], an autonomous data engine that proposes diverse geometries and designs their mesh strategies as a human expert does. The engine produces a geometry family and its trajectories in three stages (Fig. 2). In the first stage, the agent proposes a new geometry family as a short script on predefined geometry APIs (Sec. 2.1). In the second stage, the agent designs the mesh strategy as a script on predefined mesh APIs (Sec. 2.2), and the engine verifies the strategy by the mesh convergence test. When the test fails, the agent receives the failure report and rewrites the strategy until one passes. In the third stage, the engine simulates every trajectory of the family with the verified strategy in OpenFOAM (Weller et al., 1998) and exports the fields on a uniform grid (Sec. 2.3). The three stages run independently as a pipeline, so the engine efficiently produces a large dataset.

We evaluate the engine on the dataset it generates. The dataset currently contains 3,948 distinct two-dimensional geometries from 459 families and 16,512 trajectories, which is 40 times the geometry families of any previous flow dataset (Fig. 1). Our results show that (1) the engine **proposes diverse geometries** automatically, covering 10 application categories such as cyclones, hydraulic structures, and biomedical vessels (Sec. 3.1), and (2) it **successfully repairs the mesh strategies that fail** (Sec. 3.2). Of the 459 families, only 35 pass the convergence test on the default mesh strategy, and the other 424 pass after the agent repairs the strategy. Because the engine runs without human intervention, the dataset scales automatically, and the numbers above are those of the current release. To our knowledge, the KaleidoFlow dataset is the first flow dataset of diverse geometries.

To fully demonstrate the practical value of the engine, we further build the first benchmark for generalization across diverse geometries on this dataset. We design three generalization levels of increasing difficulty, G1, G2, and G3. G1 tests a new Reynolds number of a trained geometry, G2 tests a new geometry of a trained family, and G3 tests a new family, drawn as a whole cluster far from every training family in shape space (Sec. 4.1). We train nine architectures of the convolutional, spectral, and transformer families under the

[1]The name refers to the kaleidoscope and its ever-changing patterns. The engine likewise generates a large variety of new geometries and the flows they produce.

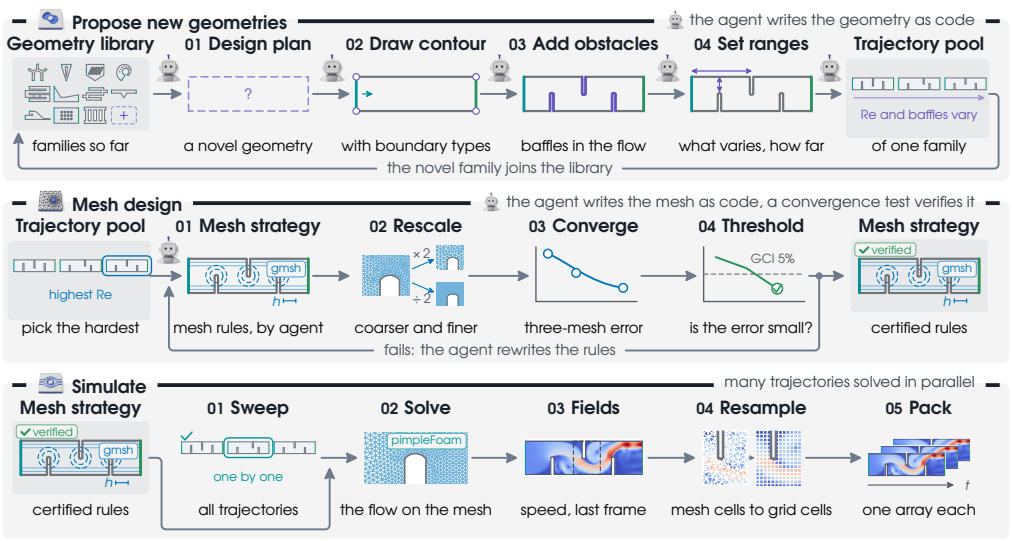

Figure 2: **The KaleidoFlow autonomous data engine.** Top, the agent proposes a new geometry family and its geometries (Sec. 2.1). Middle, the agent generates the mesh strategy and verifies it on the hardest trajectory (Sec. 2.2). Bottom, the simulation runs each trajectory on the verified mesh strategy (Sec. 2.3). All three rows show the same geometry family, a channel with three baffles. Code 1 is the definition of this family, and Code 2 is its mesh strategy.

same protocol (Sec. 4.2). We report two findings on the benchmark. First, current neural operators generalize poorly to a new family. For every architecture, the **error increases by 3× to 6×** from G2 to G3 but remains almost unchanged from G1 to G2, independent of the architecture family and the model size. Second, the error on new families decreases with the number of training families **approximately as a power law**. A convolutional, a transformer, and a spectral architecture follow the same law. To our knowledge, it is *the first power law observed for the generalization of neural operators across geometries.*

Our main contributions are the following. **(a)** We build the first autonomous data engine for verified flow simulation, based on language model agents and the grid convergence index test. **(b)** We produce the first flow dataset of diverse geometries, with a verified mesh for every geometry family, and the dataset scales automatically. **(c)** We build the first benchmark that trains and evaluates neural operators jointly across diverse geometries. **(d)** We observe the first approximate power law for the generalization of neural operators across geometries. The error on new families decreases with the number of training families, for a convolutional, a transformer, and a spectral architecture. We will release the engine and the benchmark for the future development of foundation neural operators that generalize across diverse geometries.

2 The KaleidoFlow data engine

We introduce KaleidoFlow, an autonomous data engine that produces diverse geometry families and their trajectories without human intervention. A *geometry family* is a parametric flow geometry. It consists of an outer boundary with a boundary condition on every segment, the obstacles inside it, and the *geometric parameters* and *Reynolds numbers* that vary them. One setting of the geometric parameters induces one *geometry*, and one geometry solved at one Reynolds number is one *trajectory* (Fig. 3). Each geometry family has its *mesh strategy*, a rule that produces a mesh for every geometry of the family. The geometry family is written as a short script on predefined APIs.

The engine generates each geometry family in three stages (Fig. 2). Stage 1 (Sec. 2.1) generates a new *geometry family* and its *geometries*. Stage 2 (Sec. 2.2) generates the *mesh strategy*. Stage 3 (Sec. 2.3) generates the *trajectories*. Due to the page limit, we provide the details of every stage in Appendix A.

2.1 Proposing new geometries

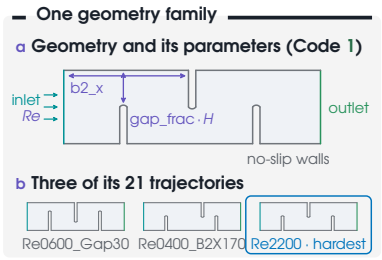

Figure 3: **A geometry family.**

In Stage 1, the agent continuously generates new geometry families to be simulated, in the four steps of the top row of Fig. 2. First, the agent reviews the existing families and proposes a design plan that differs from all of them. The plan describes an actual engineering flow with physically justified dimensions (Appendix A.1 introduces its four rules). Second, the agent constructs the outer boundary of the geometry as a closed loop of segments and assigns one boundary condition to every segment. Third, the agent adds the obstacles in the flow and sets the Reynolds number with a reference velocity and length. Finally, the agent specifies the geometric parameters that the family varies, such as a gap width or the position of a baffle, together with the Reynolds numbers. Each combination of values corresponds to one trajectory. The new family and its trajectories are then added to the existing families, and the next proposal is based on them. Therefore, the agent continuously proposes families that differ from the earlier ones.

Although the overall framework is simple, it is effective, and we find that the engine produces diverse geometry families (Sec. 3.1). In the practical implementation, we provide predefined geometry APIs for the geometric primitives, the boundary conditions, and the physics settings, so the agent performs every step by writing a short script (Code 1).

2.2 Mesh design and convergence verification

In Stage 2, the agent generates a mesh strategy for each geometry family from Stage 1, in the four steps of the middle row of Fig. 2. First, the agent generates the mesh strategy based on Gmsh (Geuzaine & Remacle, 2009), which states how the mesh of the whole geometry is generated, where the cells should be fine, and where they can be coarse (Appendix A.2 shows one example, Code 2). Second, the engine selects the hardest trajectory of the family, usually the trajectory with the highest Reynolds number. It builds three meshes of this trajectory with the strategy at the scale factors $\times 2$, $\times 1$, and $\times 1/2$, so the coarse, middle, and fine mesh have the same design. Third, the engine solves the three meshes and estimates the discretization error of the middle mesh by the grid convergence index (Roache, 1994; Celik et al., 2008) (Algorithm 1). Finally, the strategy passes when the middle mesh has at most 400,000 cells and its grid convergence index is below 5%. The index is computed for the velocity field, the pressure field, and four integral quantities, the mean speed, the peak speed, the kinetic energy, and the pressure drop (Table 1 in Appendix B.1). If the strategy passes, it will be used to solve every trajectory of the family.

If the test fails, the agent iteratively rewrites the strategy from a failure report (Prompt 1).

$$\text{mesh-strategy}_{j+1} = \text{agent}\big(\text{mesh-strategy}_j,\ \text{failure-report}_j\big), \qquad j = 0, 1, \ldots. \tag{1}$$

The failure report lists the failed quantity and every earlier attempt with its sizes, cell counts, and result, so the agent can locate the region that requires cells. Appendix A.2 shows one report example and one reply of the agent (Response 1). The repair ends when a strategy passes the test.

We find that the agent can successfully repair the failing strategies, and Fig. 5b shows three repairs round by round. Sec. 3.2 reports the repair statistics collected while the dataset was built, and Appendices C.3 and C.4 present the full statistics and more repair examples.

2.3 Simulation and export

In Stage 3, the engine simulates every trajectory of the family with the verified mesh strategy. For each trajectory, it builds the mesh, solves incompressible flow with OpenFOAM (Weller et al., 1998), and records the velocity and pressure fields at the output times of the family (Appendix A.3). The fields are then resampled to a uniform grid over the bounding box of the geometry by conservative area-weighted averaging, which also produces the fluid mask (Appendix A.4). Each trajectory is exported with its fields, the fluid mask, the boundary-type map, and the prescribed boundary values. The trajectories are independent, so the engine computes many families in parallel (Appendix A.5), and every trajectory is checked for mesh quality, solver convergence, and resampling loss (Appendix B.2).

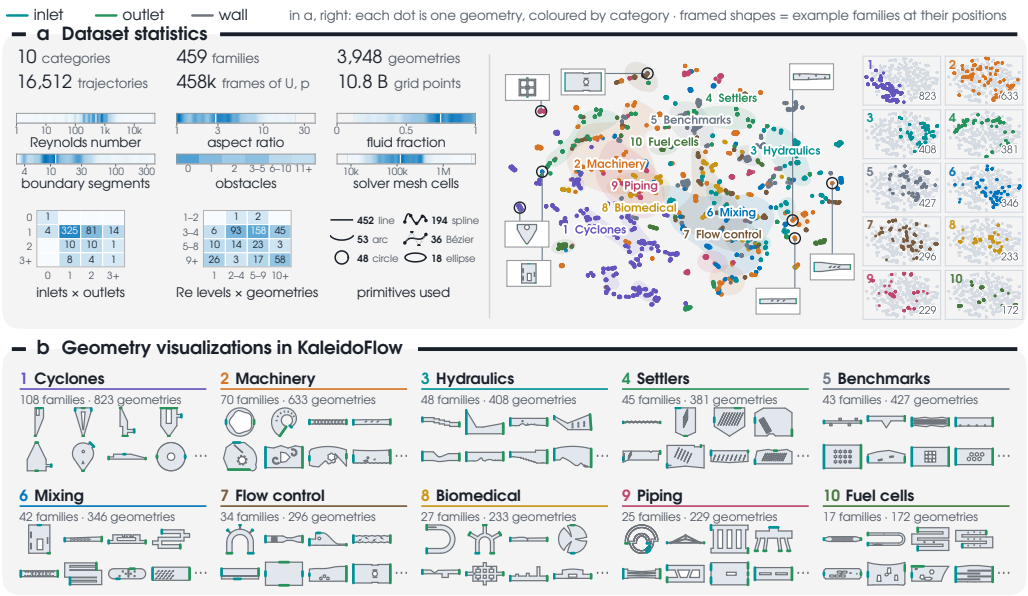

Figure 4: **The diversity of the dataset generated by KaleidoFlow.** **a**, Left, the statistics of the dataset. Right, the shape space of the 3,948 geometries, embedded from their rasterized fluid regions by a principal component analysis. Each dot is one geometry, coloured by the category of its family. **b**, Geometry examples of each category.

3 Evaluation of the data engine

To evaluate the engine, we deploy it on a CPU cluster and run it without human intervention. This produced the current release of the KaleidoFlow dataset at a cost of approximately 54K CPU hours, 2K for the convergence tests and 52K for the trajectories.

We analyze this dataset from two aspects. First, its geometries are diverse, covering 10 application categories and differing in shape within each category (Sec. 3.1). Second, the failed mesh strategies are successfully repaired (Sec. 3.2). To our knowledge, the KaleidoFlow dataset is the first flow dataset of diverse geometries.

3.1 The engine generates diverse geometries

The current release contains 459 verified geometry families, 3,948 distinct geometries, and 16,512 trajectories, with 458,001 snapshots and 10.8 billion grid points in total. Fig. 4 reports its statistics, and Appendix C.1 presents more details. The dataset is diverse in three aspects. The geometries cover many applications and shapes, the families differ widely in shape, and the trajectories span a range of flow conditions.

First, the geometries are diverse in application and in shape statistics (Fig. 4). (a) The families cover 10 application categories, although the engine is not explicitly required to cover any of them (Fig. 4b). We examine the produced families and assign each to one category. The largest category is cyclones with 108 families, and the smallest is fuel cells with 17. Appendix C.2 describes all 10 categories. The categories range from industrial separators and rotating machines to blood vessels and heart valves. (b) The shape statistics of the geometries span wide ranges (Fig. 4a, left). The outer boundary of a geometry has 2 to 1,746 segments, the number of obstacles ranges from 0 to 74, and the aspect ratio spans 1 to 23. This demonstrates the diversity of the dataset in application and in shape.

Second, the families differ widely in shape. To quantify the differences between the families, we place every geometry in a shape space by a principal component analysis (PCA) of its rasterized fluid region, as visualized in Fig. 4a, right (Appendix C.1). In this space, the differences between families exceed the differences within a family in three measures. (a) 92% of the shape variance lies between families and 8% within them, so a new family changes the shape much more than a new geometry of an existing family. (b) The PCA of the dataset needs 70 components to reach 90% of the variance, and the PCA of a single

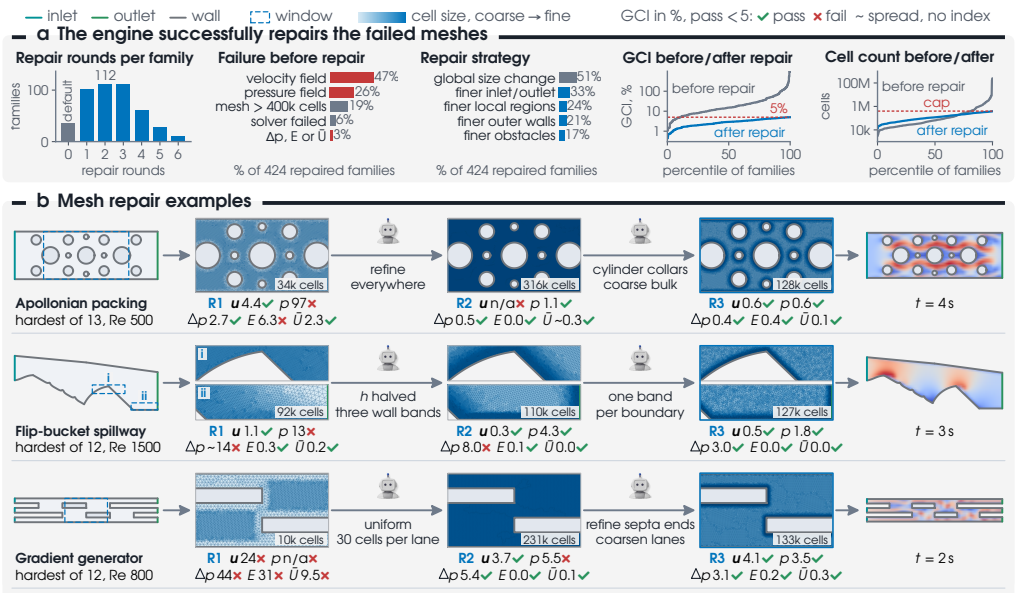

Figure 5: **The engine successfully repairs the failed meshes. a**, Mesh repair statistics of the dataset. **b**, Three mesh repair examples, one family per row. The framed mesh passed. Fig. 9 shows all five examples.

family needs 4, so the dataset spans many more shape directions than any one family. (c) The distance from a family to its nearest family is 3.8 times the spread of its own geometries at the median, so the families do not overlap. Therefore, the shape diversity of the dataset grows with the number of families.

Third, the trajectories are diverse in flow conditions. The Reynolds number spans 100 to 40,000, and most of the trajectories are below 2,500. Every geometry is solved at several Reynolds numbers. The boundary conditions include velocity inlets, pressure outlets, no-slip walls, moving or rotating walls, and symmetry planes.

In summary, the dataset is diverse in application, in shape, and in flow conditions. The engine varies the geometry within a family and the family across the dataset. It is also worth noting that the diversity can continue to increase while the engine runs.

3.2 The engine successfully repairs the failed meshes

Solving the trajectories of diverse geometries efficiently requires a bounded cell count for every geometry. The engine therefore limits the mesh of a family to 400,000 cells. Within this limit, a mesh strategy specifies how the mesh of every geometry of the family is generated, and the convergence test verifies the mesh strategy on the hardest trajectory (Sec. 2.2). The mesh strategy passes when its grid convergence index is below 5% for the velocity field, the pressure field, and four integral quantities, and Appendix B.1 presents the details. Every family starts from a default mesh strategy, and the agent iteratively repairs the strategy until the test passes.

Our results show that the engine can successfully repair the failed mesh strategies (Fig. 5). Of the 459 families in the dataset, only 35 pass the test on the default strategy, and the other 424 pass after repair. The default strategy fails on the velocity field in 47% of these families and on the pressure field in 26%, and in 19% its mesh exceeds the cell limit. The agent repairs a strategy from the failure report in 2.6 rounds on average and at most 6, and it generated 1,107 strategies in total. The strategy that passes changes the cell size globally in 51% of the repaired families and refines the mesh locally in the others, at the inlets and outlets, in specified regions, at the outer walls, or at the obstacle walls (Appendix C.3). Fig. 5b shows three repairs round by round. After repair, the median grid convergence index falls from 15% on the default mesh to 2.9% on the released mesh (Fig. 5a). In 68% of the families the released mesh has fewer cells than a mesh that failed the test, because the repair places cells where the flow requires them.

In summary, the engine repairs and verifies the mesh strategies without human intervention. In addition, it checks every trajectory for mesh quality, solver convergence, and resampling loss (Appendix B.2). The engine therefore supplies both properties the dataset requires, the diversity of the geometries and the verification of the meshes.

4 Benchmarking neural operators across diverse geometries

To fully demonstrate the practical value of the engine, we further build the first benchmark for generalization across diverse geometries on the dataset it produces. We report two findings. First, every architecture we evaluate generalizes poorly to a new geometry family. Second, the error on new families decreases with the number of training families approximately as a power law. This suggests that the error can potentially be reduced further by producing more families with the engine.

4.1 Benchmark splits

The benchmark consists of one training split, one validation split, and four test splits (Fig. 6). Each test split tests generalization along one factor. G1-1 tests generalization to a new Reynolds number of a trained geometry, an interior value between a lower and a higher value in training. G1-2 tests generalization to the highest Reynolds number of that geometry, above every value in training. G2 tests generalization to a new geometry of a trained family and consists of 15 percent of its geometries. G3 tests generalization to a new family and consists of whole clusters of families in shape space. The four test splits contain 127, 51, 157, and 194 trajectories, respectively. The validation split is the only split that influences training. Appendix D.1 presents the construction of the splits in detail.

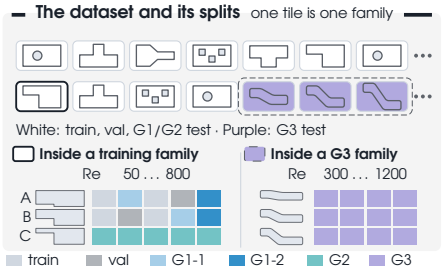

Figure 6: **The benchmark splits.** Top, the dataset with one tile per family. A cluster of families is extracted as the G3 test split. Bottom, two families expanded, with geometries as rows, Reynolds numbers as columns, and one trajectory per cell.

4.2 Training and evaluation of neural operators

The benchmark evaluates neural operators that receive the geometry as an input. Classical operator learning fixes a geometry Ω and learns $G_\Omega : (u_0, g) \mapsto u$ from the initial and boundary data. The benchmark instead evaluates how well one learned $\hat{G}$ approximates $G : (\Omega, u_0, g) \mapsto u$ over a space of admissible geometries.

Every architecture is trained and evaluated on the same uniform grid under the same protocol. Each architecture receives the same input on the uniform grid of the dataset, the velocity field, the fluid mask, the boundary-type map, the prescribed boundary values, and the coordinates of the cells, and predicts the velocity field at the next output time. The normalization, the residual step, and the masked loss are identical for all nine architectures of the convolutional, spectral, and transformer families. Therefore, a difference in the results is attributable to the architecture. Every metric is computed on the autoregressive rollout of a test trajectory. Fig. 7 reports the *relative L2 error*, the *correlation time*, the *divergence excess*, the *kinetic-energy deviation*, and the *vorticity relative L2*, and Appendix E reports all metrics of all 13 architectures, including the *high-band spectral deviation*. Appendices D.2 and D.3 introduce the interface, the protocol, and the metrics in full.

4.3 Neural operators generalize poorly to a new family

The error of every architecture jumps at a new family (Fig. 7). Relative L2 remains within a factor of 1.4 of its G1-1 value at G1-2 and at G2, and it rises by a factor of 3 to 6 from G2 to G3, in every architecture family and at every model size (Fig. 7a). The correlation time, the divergence excess, and the kinetic-energy deviation show the same jump, and at G3 every predicted field departs from the reference (Fig. 7b). The best architecture also changes at G3. UNet-c has the lowest error at G1-1, G1-2, and G2, and HNO has the lowest error at G3. In summary, current neural operators generalize well to a new Reynolds number and to a new geometry of a trained family, but every one of the nine architectures still generalizes poorly to a new geometry family.

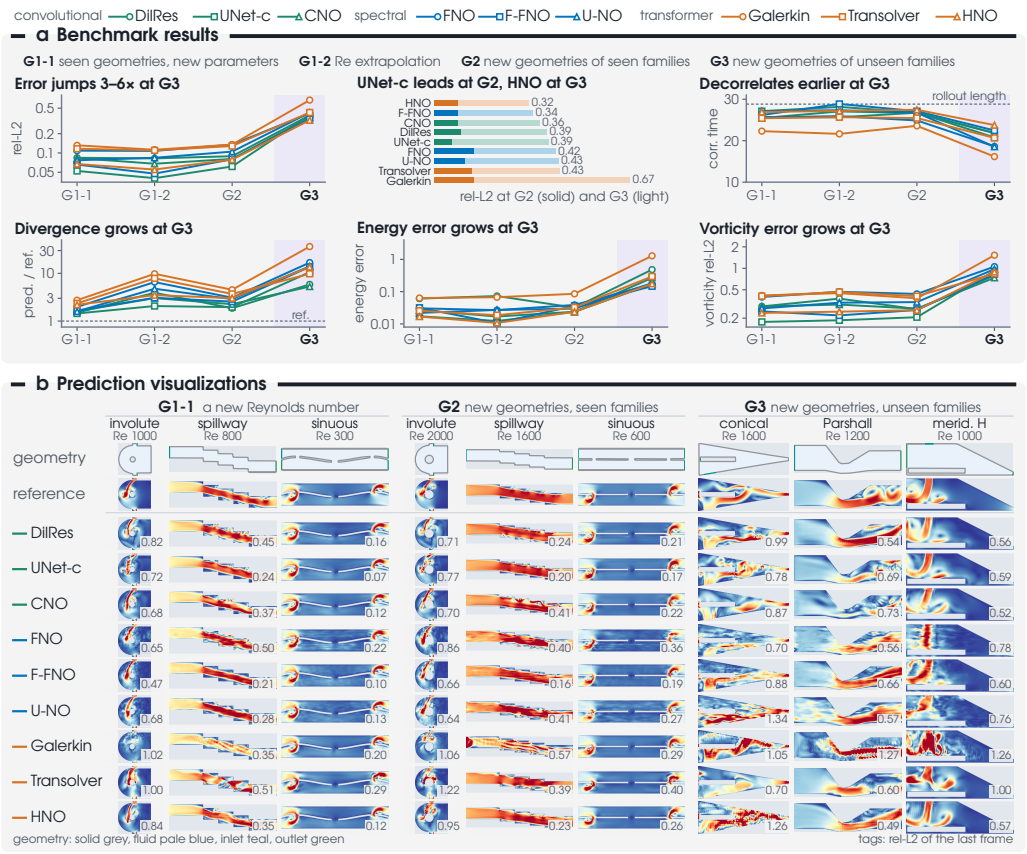

Figure 7: **Neural operators on the generalization ladder. a**, Rollout metrics of nine architectures on the four test splits. **b**, Speed at the last output time on nine test trajectories.

4.4 The error on new families follows a power law

To measure the effect of the number of training geometry families, we train a convolutional, a transformer, and a spectral architecture (UNet-c, HNO, and F-FNO) on K of 32, 64, and 128 training families and evaluate them on the same 194 G3 trajectories. Every run receives 200,000 training steps, so the compute is identical for every K. The reported error is the next-step prediction error, the relative L2 of the predicted next snapshot, as a geometric mean over families. It plays the same role as the next-token loss in the scaling laws of language models.

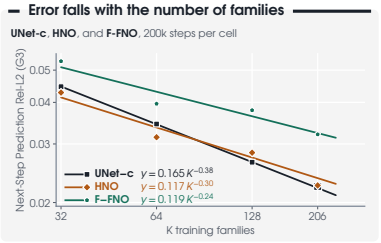

Figure 8: **Error on new families versus the number of training families.**

The error on new families decreases with the number of training families approximately as a power law, and the three architectures follow it (Fig. 8). From 32 to 128 families the error falls by 41 percent for UNet-c, 34 percent for HNO, and 29 percent for F-FNO, and a least-squares line in log-log coordinates gives exponents of 0.38, 0.30, and 0.24. We further compare the runs on all 206 training families of Sec. 4.3, which are not part of the fit, with the fitted lines, and the errors are within 5 percent. In addition, the number of trajectories per family does not change the error on new families (Fig. 10 in Appendix E.3).

In summary, the error on new families follows an approximate power law in the number of training families, and a convolutional, a transformer, and a spectral architecture share the law. With exponents of 0.24 to 0.38, each doubling of the number of families reduces the error on new families by 16 to 23 percent, and additional trajectories of the same families do not. This suggests that geometric diversity is a promising axis for scaling neural operators, and the engine can drive this scaling.

5 Related work

KaleidoFlow differs from four lines of prior work, which Appendix F discusses in full. **(a)** Existing flow datasets fix the geometry (Takamoto et al., 2022; Ohana et al., 2024), provide several fixed geometries (Luo et al., 2023; Janny et al., 2023; Tali et al., 2025), or vary the obstacle within one family (Bonnet et al., 2022; Elrefaie et al., 2024). KaleidoFlow changes the geometry from family to family, verifies every family by a mesh convergence test, and tests on new families (Appendix F.1). **(b)** Neural operators receive the geometry as fields on a grid (Li et al., 2021; Raonic et al., 2023), as a mesh or point cloud (Pfaff et al., 2021; Wu et al., 2024), as coordinates (Lu et al., 2021), or through a map to a reference geometry (Li et al., 2023c;d), and each design is reported on one geometry or one family. The benchmark evaluates nine architectures through one interface at three distances from the training geometries, so the ladder locates the level at which each fails (Appendix F.2). **(c)** Scaling laws for neural operators relate the error to the number of samples, in theory (Lanthaler et al., 2022; Liu et al., 2024a) and on one geometry (Raonic et al., 2023), or to the number of PDE systems (Subramanian et al., 2023; McCabe et al., 2024; Herde et al., 2024), whose samples share one geometry. The benchmark separates the number of geometry families from the number of trajectories per family and finds a power law in the former (Appendix F.3). **(d)** Language model agents configure and run a simulation of a geometry that the user specifies (Chen et al., 2024; Pandey et al., 2025; Yue et al., 2025a). In KaleidoFlow the agent proposes the geometry, generates and successfully repairs the mesh strategy, and every family passes the mesh convergence test of computational fluid dynamics (Roache, 1994; Celik et al., 2008) before it is added to the dataset (Appendices F.4 and F.5).

In summary, KaleidoFlow enables the generation of a dataset of diverse geometry families, and thus enables the whole geometry family as an input of neural operator learning. Prior work treats the geometry as a fixed input, and existing datasets vary the geometry within a limited range, so neural operators can only be trained and evaluated within the same geometry family. In KaleidoFlow the agent proposes new families, the mesh convergence test verifies them, and the dataset scales without human intervention.

6 Limitations

Although they do not affect the core conclusions of this paper, this work inevitably has the following limitations. First, the current dataset is two dimensional. Extending it to three dimensions is valuable, and the engine framework is compatible with that. The main obstacle is that each trajectory requires more computational cost. Second, although beyond the scope of this work, it is worth exploring better neural operator architectures for learning across diverse geometries. This work focuses on the data engine and the benchmark infrastructure, so we leave this for future work. Third, the engine currently generates families independently of the model training. Directing the generation toward the families where the error is highest would close the loop between the data generation and the model training, and this is worth exploring in the future.

7 Conclusion

We present KaleidoFlow, an autonomous data engine that proposes diverse geometry families, designs and verifies their mesh strategies, and simulates their trajectories without human intervention. The engine has produced a flow dataset of 459 verified families in 10 application categories. On this dataset we build a benchmark for generalization across geometry families and report two findings. Current neural operators generalize poorly to a new family, and their error on new families decreases with the number of training families approximately as a power law.

The two findings suggest two directions for the future. First, we should explore better architectures for learning the geometry, because the poor generalization to new families appears in most existing architectures. Second, we should build datasets of larger and more diverse geometry scenarios, because the power law suggests that more data may further improve the zero-shot generalization to new families.

Scaling neural operators to diverse geometries is still at an early stage, and this work takes only the first step. The engine enables a faster scaling of the dataset, and the benchmark provides a first reference. We will release the engine, the dataset, and the benchmark to promote the future development of neural operator learning across diverse geometries.

Reproducibility statement

To support the reproducibility of our results, we report the details of the engine, the dataset, and the benchmark in the appendix. Appendix A reports the three stages of the engine, with examples of the script of a geometry family, a mesh strategy, and a repair round. Appendix B reports the algorithm and the acceptance rules of the mesh convergence test, and the checks applied to every trajectory. Appendix C reports the statistics of the dataset, its application categories, and the mesh repair statistics. Appendix D reports the split rules, the configuration and training protocol of every architecture, and the definition of every metric. Appendix E reports every metric of all 13 architectures on the four test splits and every run of the power-law experiment. The code of the engine and the benchmark will be released upon publication.

AI use statement

In this work, we used generative AI tools for generating the synthetic dataset and for implementing the methods. A language model agent is a component of the data engine (Sec. 2). It proposes the geometry families and generates the mesh strategies, and OpenFOAM computes every trajectory. Outside the data engine, we used generative AI tools to assist the writing and the coding. We have not used generative AI tools for proposing or refining hypotheses, for designing the methodology or the experiments, or for interpreting the results. We have reviewed all AI-assisted work. AI-generated code was reviewed and tested by the authors, and every AI-edited passage of the text was reviewed by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

References

Corey Adams, Rishikesh Ranade, Ram Cherukuri, and Sanjay Choudhry. GeoTransolver: Learning physics on irregular domains using multi-scale geometry aware physics attention transformer. *arXiv preprint arXiv:2512.20399*, 2025.

Frédéric Alauzet and Adrien Loseille. A decade of progress on anisotropic mesh adaptation for computational fluid dynamics. *Computer-Aided Design*, 72:13–39, 2016.

Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Benedikt Alkin, Maurits Bleeker, Richard Kurle, Tobias Kronlachner, Reinhard Sonnleitner, Matthias Dorfer, and Johannes Brandstetter. AB-UPT: Scaling neural CFD surrogates for high-fidelity automotive aerodynamics simulations via anchored-branched universal physics transformers. *Transactions on Machine Learning Research*, 2025.

Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. Generating CAD code with vision-language models for 3D designs. In *International Conference on Learning Representations*, 2025.

Neil Ashton, Jordan B. Angel, Aditya S. Ghate, Gaetan K. W. Kenway, Man Long Wong, Cetin Kiris, Astrid Walle, Danielle C. Maddix, and Gary Page. WindsorML: High-fidelity computational fluid dynamics dataset for automotive aerodynamics. *arXiv preprint arXiv:2407.19320*, 2024a.

Neil Ashton, Danielle C. Maddix, Samuel Gundry, and Parisa M. Shabestari. AhmedML: High-fidelity computational fluid dynamics dataset for incompressible, low-speed bluff body aerodynamics. *arXiv preprint arXiv:2407.20801*, 2024b.

Neil Ashton, Charles Mockett, Marian Fuchs, Louis Fliessbach, Hendrik Hetmann, Thilo Knacke, Norbert Schonwald, Vangelis Skaperdas, Grigoris Fotiadis, Astrid Walle, Burkhard Hupertz, and Danielle Maddix. DrivAerML: High-fidelity computational fluid dynamics dataset for road-car external aerodynamics. *arXiv preprint arXiv:2408.11969*, 2024c.

Noah J Bagazinski and Faez Ahmed. Ship-D: Ship hull dataset for design optimization using machine learning. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 87301, pp. V03AT03A028. American Society of Mechanical Engineers, 2023.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27): e2311878121, 2024.

Jinshuai Bai, Haolin Li, Zahra Sharif Khodaei, M. H. Aliabadi, YuanTong Gu, and Xi-Qiao Feng. Discrete solution operator learning for geometry-dependent PDEs. *arXiv preprint arXiv:2601.09143*, 2026.

Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

Cristian Bodnar, Wessel P. Bruinsma, Ana Lucic, Megan Stanley, Anna Allen, Johannes Brandstetter, Patrick Garvan, Maik Riechert, Jonathan A. Weyn, Haiyu Dong, Jayesh K. Gupta, Kit Thambiratnam, Alexander T. Archibald, Chun-Chieh Wu, Elizabeth Heider, Max Welling, Richard E. Turner, and Paris Perdikaris. A foundation model for the Earth system. *Nature*, 641:1180–1187, 2025.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

Florent Bonnet, Jocelyn Mazari, Paola Cinnella, and Patrick Gallinari. AirfRANS: High fidelity computational fluid dynamics dataset for approximating Reynolds-averaged Navier–Stokes solutions. *Advances in Neural Information Processing Systems*, 35:23463–23478, 2022.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6:525–535, 2024.

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

Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K. Gupta. Clifford neural layers for PDE modeling. In *International Conference on Learning Representations*, 2023.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Jake Bruce, Michael D. Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments. In *International Conference on Machine Learning*, pp. 4603–4623, 2024.

Alexander Bukharin, Shiyang Li, Zhengyang Wang, Jingfeng Yang, Bing Yin, Xian Li, Chao Zhang, Tuo Zhao, and Haoming Jiang. Data diversity matters for robust instruction tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 3411–3425, 2024.

Shuhao Cao. Choose a transformer: Fourier or Galerkin. In *Advances in Neural Information Processing Systems*, volume 34, pp. 24924–24940, 2021.

Ishmail B Celik, Urmila Ghia, Patrick J Roache, Christopher J Freitas, Hugh W Coleman, and Peter E Raad. Procedure for estimation and reporting of uncertainty due to discretization in CFD applications. *Journal of Fluids Engineering*, 130(7):078001, 2008.

Junfeng Chen, Elie Hachem, and Jonathan Viquerat. Graph neural networks for laminar flow prediction around random two-dimensional shapes. *Physics of Fluids*, 33(12), 2021.

Tianyu Chen, Haoyi Zhou, Ying Li, Hao Wang, Chonghan Gao, Rongye Shi, Shanghang Zhang, and Jianxin Li. OmniArch: Building foundation model for scientific computing. In *International Conference on Machine Learning*, 2025.

Yuxuan Chen, Xu Zhu, Hua Zhou, and Zhuyin Ren. MetaOpenFOAM: An LLM-based multi-agent framework for CFD. *arXiv preprint arXiv:2407.21320*, 2024.

Ze Cheng, Zhongkai Hao, Xiaoqiang Wang, Jianing Huang, Youjia Wu, Xudan Liu, Yiru Zhao, Songming Liu, and Hang Su. Reference neural operators: Learning the smooth dependence of solutions of PDEs on geometric deformations. In *International Conference on Machine Learning*, 2024.

Wai Tong Chung, Bassem Akoush, Pushan Sharma, Alex Tamkin, Ki Sung Jung, Jacqueline H. Chen, Jack Guo, Davy Brouzet, Mohsen Talei, Bruno Savard, Alexei Y. Poludnenko, and Matthias Ihme. Turbulence in focus: Benchmarking scaling behavior of 3D volumetric super-resolution with BLASTNet 2.0 data. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Boris Diskin, James Thomas, Christopher L Rumsey, and Axel Schwöppe. Grid convergence for turbulent flows. In *53rd AIAA Aerospace Sciences Meeting*, pp. 1746, 2015.

Zhehao Dong, Zhen Lu, and Yue Yang. Fine-tuning a large language model for automating computational fluid dynamics simulations. *Theoretical and Applied Mechanics Letters*, 15:100594, 2025.

Luis Eça and Martin Hoekstra. A procedure for the estimation of the numerical uncertainty of CFD calculations based on grid refinement studies. *Journal of Computational Physics*, 262:104–130, 2014.

Mohamed Elrefaie, Florin Morar, Angela Dai, and Faez Ahmed. DrivAerNet++: A large-scale multimodal car dataset with computational fluid dynamics simulations and deep learning benchmarks. *Advances in Neural Information Processing Systems*, 37:499–536, 2024.

Mohamed Elrefaie, Angela Dai, and Faez Ahmed. DrivAerNet: A parametric car dataset for data-driven aerodynamic design and prediction. *Journal of Mechanical Design*, 147 (4):041712, 2025a.

Mohamed Elrefaie, Dule Shu, Matt Klenk, and Faez Ahmed. CarBench: A comprehensive benchmark for neural surrogates on high-fidelity 3D car aerodynamics. *arXiv preprint arXiv:2512.07847*, 2025b.

Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. OMNI-EPIC: Open-endedness via models of human notions of interestingness with environments programmed in code. In *International Conference on Learning Representations*, 2025.

Fangzhi Fei, Jiaxin Hu, Qiaofeng Li, and Zhenyu Liu. GFocal: A global-focal neural operator for solving PDEs on arbitrary geometries. *arXiv preprint arXiv:2508.04463*, 2025.

Krzysztof J. Fidkowski and Guodong Chen. Metric-based, goal-oriented mesh adaptation using machine learning. *Journal of Computational Physics*, 426:109957, 2021.

Niklas Freymuth, Tobias Würth, Nicolas Schreiber, Balázs Gyenes, Andreas Boltres, Johannes Mitsch, Aleksandar Taranovic, Tai Hoang, Philipp Dahlinger, Philipp Becker, Luise Kärger, and Gerhard Neumann. AMBER: Adaptive mesh generation by iterative mesh resolution prediction. *Advances in Neural Information Processing Systems*, 38: 23797–23835, 2025.

Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

Somdatta Goswami, Katiana Kontolati, Michael D. Shields, and George Em Karniadakis. Deep transfer operator learning for partial differential equations under conditional shift. *Nature Machine Intelligence*, 4(12):1155–1164, 2022.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, et al. Accelerating scientific discovery with Co-Scientist. *Nature*, 655:487–496, 2026.

Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. CAD-Coder: Text-to-CAD generation with chain-of-thought and geometric reward. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 481–490, 2016.

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

Mingyu Han, Daniel Zhengyu Huang, Yuhan Wang, Yanshu Zhang, and Jiayi Zhou. Geometric generalization of neural operators from a kernel integral perspective. *arXiv preprint arXiv:2602.01498*, 2026.

Zhen Hang, Yushan Yashengjiang, Junhui Li, Huanshuo Dong, Yang Wei, Zhezheng Hao, Jiangtao Ma, Songlin Bai, Haozhong Kai, Xihang Yue, Gangzong Si, Dongming Jiang, Chao Yao, Zhanhua Hu, Jiangqing Zhang, Pengwei Liu, Yaomin Shen, Xingyu Ren, Lei Liu, Zikang Xu, Han Li, Qingsong Yao, Hande Dong, and Hong Wang. PDEAgent-Bench: A multi-metric, multi-library benchmark for PDE solver generation. *arXiv preprint arXiv:2605.09636*, 2026.

Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. GNOT: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, 2023.

Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. DPOT: Auto-regressive denoising operator transformer for large-scale PDE pre-training. In *International Conference on Machine Learning*, 2024.

Sheikh Md Shakeel Hassan, Arthur Feeney, Akash Dhruv, Jihoon Kim, Youngjoon Suh, Jaiyoung Ryu, Yoonjin Won, and Aparna Chandramowlishwaran. BubbleML: A multiphysics dataset and benchmarks for machine learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel De Bezenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for PDEs. *Advances in Neural Information Processing Systems*, 37:72525–72624, 2024.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Matthew Ho, Brian Liu, Jixuan Chen, Audrey Wang, and Lianhui Qin. Auto-configuring scientific simulators with lightweight coding-agent adapters. *arXiv preprint arXiv:2606.09774*, 2026.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030, 2022.

Benjamin Holzschuh, Qiang Liu, Georg Kohl, and Nils Thuerey. PDE-Transformer: Efficient and versatile transformers for physics simulations. In *International Conference on Machine Learning*, 2025.

Pu Hua, Minghuan Liu, Annabella Macaluso, Yunfeng Lin, Weinan Zhang, Huazhe Xu, and Lirui Wang. GenSim2: Scaling robot data generation with multi-modal and reasoning LLMs. In *Conference on Robot Learning*, 2024.

Jianing Huang, Kaixuan Zhang, Youjia Wu, and Ze Cheng. Operator learning with domain decomposition for geometry generalization in PDE solving. In *International Conference on Learning Representations*, 2026.

Steeven Janny, Aurélien Bénéteau, Madiha Nadri, Julie Digne, Nicolas Thome, and Christian Wolf. EAGLE: Large-scale learning of turbulent fluid dynamics with mesh transformers. In *International Conference on Learning Representations*, 2023.

Rohit Kanchi, Benjamin Melanson, Nithin Somasekharan, Shaowu Pan, and Sicheng He. UniFoil: A universal dataset of airfoils in transitional and turbulent regimes for subsonic and transonic flows. *Advances in Neural Information Processing Systems*, 38, 2025.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Ali Kashefi, Davis Rempe, and Leonidas J. Guibas. A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries. *Physics of Fluids*, 33(2):027104, 2021.

Shubham Kavane, Lukas Schröder, Kajol Kulkarni, Fernando Gonzalez, and Harald Koestler. ChannelFlow-Tools: A configuration-driven pipeline for generating machine-learning-ready datasets of 3D obstructed channel flows. *arXiv preprint arXiv:2509.15236*, 2025.

Mohammad S Khan, Sankalp Sinha, Talha U Sheikh, Didier Stricker, Sk A Ali, and Muhammad Z Afzal. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. *Advances in Neural Information Processing Systems*, 37:7552–7579, 2024.

Sungwon Kim, Juho Song, Seungmin Shin, Guimok Cho, Sangkook Kim, and Chanyoung Park. EqGINO: Equivariant geometry-informed Fourier neural operators for 3D PDEs. In *International Conference on Machine Learning*, 2026.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. In *IEEE/CVF International Conference on Computer Vision*, 2023.

Felix Koehler, Simon Niedermayr, Rüdiger Westermann, and Nils Thuerey. APEBench: A benchmark for autoregressive neural emulators of PDEs. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Minsu Koh, Beom-Chul Park, Heejo Kong, and Seong-Whan Lee. Integrating locality-aware attention with transformers for general geometry PDEs. In *International Joint Conference on Neural Networks*, 2025.

Georg Kohl, Li-Wei Chen, and Nils Thuerey. Benchmarking autoregressive conditional diffusion models for turbulent flow simulation. *Neural Networks*, 199:108641, 2026.

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

Samuel Lanthaler, Siddhartha Mishra, and George Em Karniadakis. Error estimates for DeepONets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1):tnac001, 2022.

Hojin Lee, Andreas Lintermann, Sangseung Lee, and Mario Rüttgers. UrbanFlow-3K: A dataset of 3,000 lattice-Boltzmann simulations of random building layouts. *arXiv preprint arXiv:2603.16554*, 2026.

Shanda Li, Tanya Marwah, Junhong Shen, Weiwei Sun, Andrej Risteski, Yiming Yang, and Ameet Talwalkar. CodePDE: An inference framework for LLM-driven PDE solver generation. *Transactions on Machine Learning Research*, 2026.

Xingang Li, Yuewan Sun, and Zhenghui Sha. LLM4CAD: Multimodal large language models for three-dimensional computer-aided design generation. *Journal of Computing and Information Science in Engineering*, 25(2):021005, 2025.

Yi Li, Eric Perlman, Minping Wan, Yunke Yang, Charles Meneveau, Randal Burns, Shiyi Chen, Alexander Szalay, and Gregory Eyink. A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, 9:N31, 2008.

Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *Transactions on Machine Learning Research*, 2023a.

Zijie Li, Dule Shu, and Amir Barati Farimani. Scalable transformer for PDE surrogate modeling. In *Advances in Neural Information Processing Systems*, volume 36, 2023b.

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

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

Zongyi Li, Nikola Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, et al. Geometry-informed neural operator for large-scale 3D PDEs. *Advances in Neural Information Processing Systems*, 36:35836–35854, 2023d.

William Liang, Sam Wang, Hungju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Eurekaverse: Environment curriculum generation via large language models. In *Conference on Robot Learning*, 2024.

Fanqi Lin, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. Data scaling laws in imitation learning for robotic manipulation. In *International Conference on Learning Representations*, 2025.

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

Hao Liu, Zecheng Zhang, Wenjing Liao, and Hayden Schaeffer. Neural scaling laws of deep ReLU and deep operator network: A theoretical study. *arXiv preprint arXiv:2410.00357*, 2024a.

Jian Liu, Jianyu Wu, Hairun Xie, Guoqing Zhang, Jing Wang, Wei Liu, Wanli Ouyang, Junjun Jiang, Xianming Liu, Shixiang Tang, and Miao Zhang. AFBench: A large-scale benchmark for airfoil design. In *Advances in Neural Information Processing Systems*, volume 37, 2024b.

Menghan Liu, Jianhuan Cen, Ziyang Zhou, Haolong Fan, Hongji Li, Ping Wei, Guohang Peng, Changye He, Yuzhe Qin, Yutong Lu, and Qingsong Zou. CFDONEval: A comprehensive evaluation of operator-learning neural network models for computational fluid dynamics. In *International Joint Conference on Artificial Intelligence*, pp. 5752–5760, 2025a.

Qibang Liu, Weiheng Zhong, Hadi Meidani, Diab Abueidda, Seid Koric, and Philippe Geubelle. Geometry-informed neural operator transformer. *arXiv preprint arXiv:2504.19452*, 2025b.

Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the CoordConv solution. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Songming Liu, Zhongkai Hao, Chengyang Ying, Hang Su, Ze Cheng, and Jun Zhu. NUNO: A general framework for learning parametric PDEs with non-uniform data. In *International Conference on Machine Learning*, 2023.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

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

Lisa Lüddecke, Michael Hohmann, Sebastian Eilermann, Jan Tillmann-Mumm, Pezhman Pourabdollah, Mario Oertel, and Oliver Niggemann. WeirNet: A large-scale 3D CFD benchmark for geometric surrogate modeling of piano key weirs. *arXiv preprint arXiv:2602.20714*, 2026.

Huakun Luo, Haixu Wu, Hang Zhou, Lanxiang Xing, Yichen Di, Jianmin Wang, and Mingsheng Long. Transolver++: An accurate neural solver for PDEs on million-scale geometries. In *International Conference on Machine Learning*, 2025.

Yining Luo, Yingfa Chen, and Zhen Zhang. CFDBench: A large-scale benchmark for machine learning methods in fluid dynamics. *arXiv preprint arXiv:2310.05963*, 2023.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024.

Runze Mao, Rui Zhang, Xuan Bai, Tianhao Wu, Teng Zhang, Zhenyi Chen, Minqi Lin, Bocheng Zeng, Yangchen Xu, Yingxuan Xiang, et al. Benchmarking neural surrogates on realistic spatiotemporal multiphysics flows. *arXiv preprint arXiv:2512.18595*, 2025.

Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, Francois Lanusse, Mariel Pettee, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. Multiple physics pretraining for spatiotemporal surrogate models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 119301–119335, 2024.

Michael McCabe, Payel Mukhopadhyay, Tanya Marwah, Bruno Régaldo-Saint Blancard, Francois Rozet, Cristiana Diaconu, Lucas Meyer, Kaze W. K. Wong, Hadi Sotoudeh, Alberto Bietti, Irina Espejo, Rio Fear, Siavash Golkar, Tom Hehir, Keiya Hirashima, Geraud Krawezik, Francois Lanusse, Rudy Morel, Ruben Ohana, Liam Parker, Mariel Pettee, Jeff

Shen, Kyunghyun Cho, Miles Cranmer, and Shirley Ho. Walrus: A cross-domain foundation model for continuum dynamics. In *International Conference on Machine Learning*, 2026.

Théodore Michel, Antoine Campos, Alban Dujardin, Henry Areiza, Philippe Meliga, and Elie Hachem. ONE CYLinder: A benchmark for graph-based surrogate modeling of unsteady bluff-body flows. *arXiv preprint arXiv:2609.08947*, 2026.

Sepehr Mousavi, Shizheng Wen, Levi Lingsch, Maximilian Herde, Bogdan Raonić, and Siddhartha Mishra. RIGNO: A graph-based framework for robust and accurate operator learning for PDEs on arbitrary domains. *arXiv preprint arXiv:2501.19205*, 2025.

Yao Mu, Tianxing Chen, Zanxin Chen, Shijia Peng, Zhiqian Lan, Zeyu Gao, Zhixuan Liang, Qiaojun Yu, Yude Zou, Mingkun Xu, Lunkai Lin, Zhiqiang Xie, Mingyu Ding, and Ping Luo. RoboTwin: Dual-arm robot benchmark with generative digital twins. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Nayantara Mudur, Hao Cui, Subhashini Venugopalan, Paul Raccuglia, Michael P. Brenner, and Peter Norgaard. FEABench: Evaluating language models on multiphysics reasoning ability. *arXiv preprint arXiv:2504.06260*, 2025.

Mohammad Amin Nabian, Chang Liu, Rishikesh Ranade, and Sanjay Choudhry. X-MeshGraphNet: Scalable multi-scale graph neural networks for physics simulation. *arXiv preprint arXiv:2411.17164*, 2024.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. *arXiv preprint arXiv:2603.04356*, 2026.

Bo Ni and Markus J. Buehler. MechAgents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge. *Extreme Mechanics Letters*, 67:102131, 2024.

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

William L Oberkampf and Timothy G Trucano. Verification and validation in computational fluid dynamics. *Progress in Aerospace Sciences*, 38(3):209–272, 2002.

Ruben Ohana, Michael McCabe, Lucas Meyer, Rudy Morel, Fruzsina J Agocs, Miguel Beneitez, Marsha Berger, Blakesley Burkhart, Stuart B Dalziel, Drummond B Fielding, et al. The well: a large-scale collection of diverse physics simulations for machine learning. *Advances in Neural Information Processing Systems*, 37:44989–45037, 2024.

Sandeep Pandey, Ran Xu, Wenkang Wang, and Xu Chu. OpenFOAMGPT: A retrieval-augmented large language model agent for OpenFOAM-based computational fluid dynamics. *Physics of Fluids*, 37(3), 2025.

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 2332–2341. IEEE, 2019.

Jieyuan Pei, Zhuoxuan Li, Wei Li, Haobo Zhang, Jiawei Jiang, and Jianwei Zheng. Hyperbolic neural operator. In *International Conference on Machine Learning*, 2026.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

Tyrone S. Phillips and Christopher J. Roy. Richardson extrapolation-based discretization uncertainty estimation for computational fluid dynamics. *Journal of Fluids Engineering*, 136(12):121401, 2014.

Yupeng Qi, Ran Xu, and Xu Chu. FeaGPT: An end-to-end agentic-AI for finite element analysis. *arXiv preprint arXiv:2510.21993*, 2025.

Jiyan Qiu, Lyulin Kuang, Guan Wang, Yichen Xu, Leiyao Cui, Shaotong Fu, Yixin Zhu, and Ruihua Zhang. DrivAerStar: An industrial-grade CFD dataset for vehicle aerodynamic optimization. *arXiv preprint arXiv:2510.16857*, 2025.

Ali Rabeh, Ethan Herron, Aditya Balu, Soumik Sarkar, Chinmay Hegde, Adarsh Krishnamurthy, and Baskar Ganapathysubramanian. Benchmarking scientific machine-learning approaches for flow prediction around complex geometries. *Communications Engineering*, 4:182, 2025.

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

Rishikesh Ranade, Mohammad Amin Nabian, Kaustubh Tangsali, Alexey Kamenev, Oliver Hennigh, Ram Cherukuri, and Sanjay Choudhry. DoMINO: A decomposable multi-scale iterative neural operator for modeling large scale engineering simulations. *arXiv preprint arXiv:2501.13350*, 2025.

Nadun Ranawaka, Josiah Wong, Wei-Lin Pai, Wei-Teng Chu, Tianyuan Dai, Masoud Moghani, Hang Yin, Yunfan Jiang, Wesley Durbano, Brandon Huynh, Yu Fang, Danfei Xu, Ruohan Zhang, Li Fei-Fei, Linxi Fan, Bowen Wen, Ajay Mandlekar, and Yuke Zhu. SimFoundry: Modular and automated scene generation for policy learning and evaluation. *arXiv preprint arXiv:2606.28276*, 2026.

Bogdan Raonic, Roberto Molinaro, Tim De Ryck, Tobias Rohner, Francesca Bartolucci, Rima Alaifari, Siddhartha Mishra, and Emmanuel De Bézenac. Convolutional neural operators for robust and accurate learning of PDEs. *Advances in Neural Information Processing Systems*, 36:77187–77200, 2023.

Mateus Dias Ribeiro, Abdul Rehman, Sheraz Ahmed, and Andreas Dengel. DeepCFD: Efficient steady-state laminar flow approximation with deep convolutional neural networks. *arXiv preprint arXiv:2004.08826*, 2020.

Lewis F Richardson and J Arthur Gaunt. The deferred approach to the limit. Part I. Single lattice. Part II. Interpenetrating lattices. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 226: 299–361, 1927.

Lewis Fry Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 210:307–357, 1911.

Patrick J Roache. Perspective: A method for uniform reporting of grid refinement studies. *Journal of Fluids Engineering*, 116(3):405–413, 1994.

Patrick J Roache. Quantification of uncertainty in computational fluid dynamics. *Annual Review of Fluid Mechanics*, 29:123–160, 1997.

Patrick J Roache. *Verification and validation in computational science and engineering*. Hermosa Publishers, Albuquerque, NM, 1998.

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

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241. Springer, 2015.

Christopher J. Roy. Review of code and solution verification procedures for computational simulation. *Journal of Computational Physics*, 205(1):131–156, 2005.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Recode: Reverse engineering CAD code from point clouds. In *IEEE/CVF International Conference on Computer Vision*, 2025.

Nadim Saad, Gaurav Gupta, Shima Alizadeh, and Danielle C. Maddix. Guiding continuous operator learning through physics-based boundary constraints. In *International Conference on Learning Representations*, 2023.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468, 2020.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using LLM agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.

Louis Serrano, Lise Le Boudec, Armand Kassaï Koupaï, Thomas X. Wang, Yuan Yin, Jean-Noël Vittaut, and Patrick Gallinari. Operator learning with neural fields: Tackling PDEs on general geometries. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Louis Serrano, Thomas X. Wang, Etienne Le Naour, Jean-Noël Vittaut, and Patrick Gallinari. AROMA: Preserving spatial structure for latent PDE modeling with local neural fields. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Paul Setinek, Gianluca Galletti, and Johannes Brandstetter. Towards multi-fidelity scaling laws of neural surrogates in CFD. *arXiv preprint arXiv:2511.01830*, 2025.

Benjamin D. Shaffer, Shawn Koohy, Brooks Kinch, M. Ani Hsieh, and Nathaniel Trask. Structure-preserving learning improves geometry generalization in neural PDEs. *arXiv preprint arXiv:2602.02788*, 2026.

Omer Shafiq, Amin Rahmat, Alessio Alexiadis, and Bahman Ghiassi. Evaluating the performance of large language models for geometry and simulation file generation in physics-based simulations. *Applied Sciences*, 15(22):12114, 2025.

Junhong Shen, Tanya Marwah, and Ameet Talwalkar. UPS: Efficiently building foundation models for PDE solving via cross-modal adaptation. *Transactions on Machine Learning Research*, 2024.

Modi Shi, Li Chen, Jin Chen, Yuxiang Lu, Chiming Liu, Guanghui Ren, Ping Luo, Di Huang, Maoqing Yao, and Hongyang Li. Is diversity all you need for scalable robotic manipulation? *arXiv preprint arXiv:2507.06219*, 2025.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. LLM-SR: Scientific equation discovery via programming with large language models. In *International Conference on Learning Representations*, 2025.

Nithin Somasekharan, Ling Yue, Yadi Cao, Weichao Li, Patrick Emami, Pochinapeddi Sai Bhargav, Anurag Acharya, Xingyu Xie, and Shaowu Pan. CFDLLMBench: A benchmark suite for evaluating large language models in computational fluid dynamics. *arXiv preprint arXiv:2509.20374*, 2025.

Guijin Son, Jehyun Park, Seyeon Park, Sunghee Ahn, and Youngjae Yu. Self-improving CAD generation agents with finite element analysis as feedback. *arXiv preprint arXiv:2605.17448*, 2026.

Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari S. Morcos. Beyond neural scaling laws: Beating power law scaling via data pruning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 19523–19536, 2022.

Kimberly Stachenfeld, Drummond B. Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned coarse models for efficient turbulence simulation. In *International Conference on Learning Representations*, 2022.

Fred Stern, Robert V Wilson, Hugh W Coleman, and Eric G Paterson. Comprehensive approach to verification and validation of CFD simulations—Part 1: Methodology and procedures. *Journal of Fluids Engineering*, 123(4):793–802, 2001.

Shashank Subramanian, Peter Harrington, Kurt Keutzer, Wahid Bhimji, Dmitriy Morozov, Michael W. Mahoney, and Amir Gholami. Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Guoze Sun, Tianya Miao, Haoyang Huang, Huaguan Chen, Han Wan, Rui Zhang, and Hao Sun. Geometry-aware neural optimizer for shape optimization and inversion. *arXiv preprint arXiv:2605.04474*, 2026.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

Makoto Takamoto, Francesco Alesiani, and Mathias Niepert. Learning neural PDE solvers with parameter-guided channel attention. In *International Conference on Machine Learning*, pp. 33448–33467. PMLR, 2023.

Ronak Tali, Ali Rabeh, Cheng-Hau Yang, Mehdi Shadkhah, Samundra Karki, Abhisek Upadhyaya, Suriya Dhakshinamoorthy, Marjan Saadati, Soumik Sarkar, Adarsh Krishnamurthy, Chinmay Hegde, Aditya Balu, and Baskar Ganapathysubramanian. FlowBench: A large scale benchmark for flow simulation over complex geometries. *Journal of Data-centric Machine Learning Research*, 2025.

Hao Tang, Jiongyu Zhu, Zimeng Feng, Hao Li, and Chao Li. Geometric Laplace neural operator. *arXiv preprint arXiv:2512.16409*, 2025.

Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows. *AIAA Journal*, 58(1): 25–36, 2020.

Artur P. Toshev, Gianluca Galletti, Fabian Fritz, Stefan Adami, and Nikolaus A. Adams. LagrangeBench: A Lagrangian fluid mechanics benchmarking suite. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized Fourier neural operators. In *International Conference on Learning Representations*, 2023.

Nobuyuki Umetani and Bernd Bickel. Learning three-dimensional flow for interactive aerodynamic design. *ACM Transactions on Graphics (TOG)*, 37(4):1–10, 2018.

Jonathan Viquerat and Elie Hachem. A supervised neural network for drag prediction of arbitrary 2D shapes in laminar flows at low Reynolds number. *Computers & Fluids*, 210: 104645, 2020.

Haixin Wang, Ruoyan Li, Fred Xu, Fang Sun, Kaiqiao Han, Zijie Huang, Ching Chang, Xiao Luo, Wei Wang, and Yizhou Sun. FD-Bench: A modular and fair benchmark for data-driven fluid simulation. *arXiv preprint arXiv:2505.20349*, 2025a.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. GenSim: Generating robotic simulation tasks via large language models. In *International Conference on Learning Representations*, 2024a.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

Sifan Wang, Jacob H. Seidman, Shyam Sankaran, Hanwen Wang, George J. Pappas, and Paris Perdikaris. CViT: Continuous vision transformer for operator learning. In *International Conference on Learning Representations*, 2025b.

Tian Wang and Chuang Wang. Latent neural operator for solving forward and inverse PDE problems. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning language models with self-generated instructions. In *Annual Meeting of the Association for Computational Linguistics*, pp. 13484–13508, 2023.

Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. RoboGen: Towards unleashing infinite data for automated robot learning via generative simulation. In *International Conference on Machine Learning*, 2024b.

Jiaqi Wei, Yuejin Yang, Xiang Zhang, Yuhan Chen, Xiang Zhuang, Zhangyang Gao, et al. From AI for science to agentic science: A survey on autonomous scientific discovery. *arXiv preprint arXiv:2508.14111*, 2025.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-Instruct. In *International Conference on Machine Learning*, 2024.

Henry G Weller, Gavin Tabor, Hrvoje Jasak, and Christer Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in Physics*, 12(6):620–631, 1998.

Shizheng Wen, Arsh Kumbhat, Levi Lingsch, Sepehr Mousavi, Yizhou Zhao, Praveen Chandrashekar, and Siddhartha Mishra. Geometry aware operator transformer as an efficient and accurate neural surrogate for PDEs on arbitrary domains. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Haixu Wu, Tengge Hu, Huakun Luo, Jianmin Wang, and Mingsheng Long. Solving high-dimensional PDEs with latent spectral models. In *International Conference on Machine Learning*, pp. 37417–37438, 2023.

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

Haoyang Wu, Xinxin Zhang, and Lailai Zhu. Automated code development for PDE solvers using large language models. *arXiv preprint arXiv:2509.25194*, 2025.

Yanming Xia and Angelica I. Aviles-Rivero. Do neural operators forget geometry? The forgetting hypothesis in deep operator learning. *arXiv preprint arXiv:2605.05862*, 2026.

Ke Xiao, Haoze Zhang, Runze Mao, Han Li, and Zhi X. Chen. Towards LLM-enabled autonomous combustion research: A literature-aware agent for self-corrective modeling workflows. *arXiv preprint arXiv:2601.01357*, 2026a.

Ke Xiao, Haoze Zhang, Yangchen Xu, Runze Mao, Han Li, and Zhi X. Chen. A preliminary assessment of coding agents for CFD workflows. *arXiv preprint arXiv:2602.11689*, 2026b.

Haoyang Xie and Feng Ju. Text-to-CadQuery: A new paradigm for CAD generation with scalable large model capabilities. *arXiv preprint arXiv:2505.06507*, 2025.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, 2024a.

Leidong Xu, Danyal Mohaddes, and Yi Wang. LLM agent for fire dynamics simulations. In *NeurIPS 2024 Workshop on Foundation Models for Science*, 2024b.

Wenzhuo Xu, Noelia Grande Gutierrez, and Christopher McComb. MegaFlow2D: A parametric dataset for machine learning super-resolution in computational fluid dynamics simulations. In *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*, 2023.

Liu Yang, Siting Liu, Tingwei Meng, and Stanley J. Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.

Minglang Yin, Nicolas Charon, Ryan Brody, Lu Lu, Natalia Trayanova, and Mauro Maggioni. A scalable framework for learning the geometry-dependent solution operators of partial differential equations. *Nature Computational Science*, 4(12):928–940, 2024.

Yuan Yin, Matthieu Kirchmeyer, Jean-Yves Franceschi, Alain Rakotomamonjy, and Patrick Gallinari. Continuous PDE dynamics forecasting with implicit neural representations. In *International Conference on Learning Representations*, 2023.

Ling Yue, Nithin Somasekharan, Tingwen Zhang, Yadi Cao, Zhangze Chen, Shimin Di, and Shaowu Pan. Foam-Agent: Towards automated intelligent CFD workflows. *arXiv preprint arXiv:2505.04997*, 2025a.

Ling Yue, Peijing Xu, Tingwen Zhang, Nithin Somasekharan, and Shaowu Pan. FoamGPT: Fine-tuning large language model for agentic automation of CFD simulations with OpenFOAM. In *NeurIPS 2025 Workshop on Machine Learning and the Physical Sciences*, 2025b.

Xihang Yue, Yi Yang, and Linchao Zhu. DeltaPhi: Physical states residual learning for neural operators in data-limited PDE solving. *Advances in Neural Information Processing Systems*, 38, 2025c.

Xihang Yue, Yi Yang, and Linchao Zhu. Holistic physics solver: Learning PDEs in a unified spectral-physical space. In *International Conference on Machine Learning*, 2025d.

Chenyu Zeng, Yanshu Zhang, Jiayi Zhou, Yuhan Wang, Zilin Wang, Yuhao Liu, Lei Wu, and Daniel Zhengyu Huang. Point cloud neural operator for parametric PDEs on complex and variable geometries. *Computer Methods in Applied Mechanics and Engineering*, 443: 118022, 2025.

Jenny Zhang, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. OMNI: Open-endedness via models of human notions of interestingness. In *International Conference on Learning Representations*, 2024.

Jiachen Zhang, Junyi Lao, Chenghao Liu, Siyuan Liu, Shixin Wu, Linsen Zhang, Boyu Wang, and Songfang Huang. VFEAgent: A multimodal agent framework for end-to-end automated finite element analysis. *arXiv preprint arXiv:2605.28978*, 2026.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3813–3824. IEEE, 2023.

Zheyan Zhang, Yongxing Wang, Peter K Jimack, and He Wang. MeshingNet: A new mesh generation method based on deep learning. In *International Conference on Computational Science*, pp. 186–198. Springer, 2020.

Maksim Zhdanov, Max Welling, and Jan-Willem van de Meent. Erwin: A tree-based hierarchical transformer for large-scale physical systems. In *International Conference on Machine Learning*, 2025.

Weiheng Zhong, Qibang Liu, Diab Abueidda, Seid Koric, and Hadi Meidani. A comprehensive comparison of neural operators for 3D industry-scale engineering designs. *arXiv preprint arXiv:2510.05995*, 2025.

Hang Zhou, Yuezhou Ma, Haixu Wu, Haowen Wang, and Mingsheng Long. Unisolver: PDE-conditional transformers towards universal neural PDE solvers. In *International Conference on Machine Learning*, pp. 79061–79088, 2025.

Hang Zhou, Haixu Wu, Haonan Shangguan, Yuezhou Ma, Huikun Weng, Jianmin Wang, and Mingsheng Long. Transolver-3: Scaling up transformer solvers to industrial-scale geometries. *arXiv preprint arXiv:2602.04940*, 2026.

A The autonomous data engine in detail

This appendix expands Sec. 2. Secs. A.1–A.4 describe one geometry family end to end, and Sec. A.5 describes how the engine computes many families in parallel. Fig. 2 shows one family at each stage.

Sec. 2 defines the geometry family, the geometry, and the trajectory, and this appendix uses the three terms in the same sense. The agent specifies N combinations of the geometric parameters and the Reynolds numbers of a family, and each combination is one trajectory. A family has a median of 36 trajectories. Every stage that requires the agent, the proposal of a family in Sec. A.1 and the repair of a mesh strategy in Sec. A.2, runs on a coding agent framework based on Claude Sonnet. The agent is called once to propose a family and once per repair round, so a repaired family requires 2.6 calls on average beyond its proposal, and each call returns within minutes. The agent calls of the current release cost approximately 1,000 USD in total. The cost of the dataset is therefore that of the solver, approximately 54K CPU hours for the current release (Sec. 3), of which 2K are the convergence tests.

A.1 Proposing new geometries

A language model agent generates each new geometry family as code. The agent is a coding agent that writes a short script on predefined APIs for two-dimensional geometry, and these APIs provide the geometric primitives, the boundary conditions, and the physics settings. Code 1 shows the script of one family of the dataset. The stage produces the geometry family and its geometries in four steps, shown as the four boxes of the top row of Fig. 2, (a) a design plan based on the existing families, (b) the outer boundary with its boundary conditions, (c) the physical settings, and (d) the geometric parameters and the Reynolds numbers.

The agent first proposes a design plan based on the existing geometries. It reviews the existing geometries, examines the ones closest to its concept, and prepares a design note. The note names at least three of those families and states how the new geometry differs from them in flow mechanism, application, and engineering content. Four rules constrain the plan, (a) the geometry must correspond to an actual engineering flow scenario, not to a perturbation of an existing shape, (b) it must occupy a combination of mechanism and application that the existing geometries cover sparsely, (c) its dimensions and boundary types must be physically justified, and (d) its boundary must contain interpretable features such as vanes, passages, and obstacles. A rectangular channel with one rectangular or circular obstacle is discouraged unless the scenario names a component of exactly that shape.

The agent then constructs the outer boundary as a closed loop of segments, and every segment is assigned a boundary condition. Each segment is a line, a circular arc, a circle, an ellipse, a Bézier curve, or a spline. Each segment is given one condition, velocity inlet, pressure outlet, no-slip wall, moving or rotating wall, symmetry plane, periodic pair, or time-varying inlet. In Code 1 and Fig. 3, a wall-mounted baffle with an arc-shaped tip is three lines and one arc, each with a label, and the channel is one loop that runs counter-clockwise from the bottom wall through the outlet, the top wall, and the inlet. Bodies in the flow, such as cylinders or vanes, are further closed loops inside the geometry. The predefined APIs check that every loop closes and that no loop crosses itself.

The physical settings are the Reynolds number and the output times. The agent specifies the Reynolds number with a reference velocity and length, which fix the viscosity, and the output interval of 0.1 seconds. From these quantities and the geometry, the predefined APIs generate all other solver inputs, including the initial fields and the discretization settings.

The geometric parameters and the Reynolds numbers are specified in groups, and each group isolates one effect. The agent specifies the quantities that may vary. In Code 1 these are the Reynolds number, a gap fraction, and the position of one baffle. It then specifies the groups, a Reynolds number sweep at the baseline shape, a blockage sweep at fixed Reynolds number, and an asymmetry sweep that moves one baffle, so Code 1 has three groups and $N = 21$ trajectories. Grouping by effect replaces uniform sampling, so each group traces one physical trend and the family covers the regimes of the scenario (Fig. 3). Each trajectory is named from its values with the Reynolds number first, and this name later enables the

engine to select the hardest trajectory of a family, usually the one with the highest Reynolds number.

Before the new family is added to the existing families, the engine checks it once. It regenerates every trajectory of the geometry, renders a preview of the fluid region with its boundary conditions, estimates the mesh cost of the family without meshing, and reviews the design note. A family that passes is accepted under a name that no earlier family uses, and the next proposal is based on it.

Code 1: The script for one accepted family, `arc-tipped-baffle-channel`, abridged. Top, the outer boundary of the geometry is a closed loop of segments, each with a boundary condition and a label. A wall-mounted baffle with an arc-shaped tip is three lines and one arc. Middle, the physics is the Reynolds number and two reference scales. Bottom, the family is a set of groups, each probing one effect. Every varied quantity is an argument of the geometry function, and each set of values becomes one trajectory.

```
def make_domain(Re=600.0, gap_frac=0.475, b2_x=1.40, end_time=2.0):
    L, H, w, tip = 2.8, 0.8, 0.04, 0.02      # channel, baffle half-width, tip
    b = H - gap_frac * H - tip                # baffle body height

    def bottom_baffle(x):                     # three lines and one arc
        return [
            Wall(Line((x - w, 0), (x - w, b)),               label="b1_left"),
            Wall(Arc((x - w, b), (x, b + tip), (x + w, b)),  label="b1_arc"),
            Wall(Line((x + w, b), (x + w, 0)),               label="b1_right"),
        ]

    outer = [                                 # counter-clockwise closed loop
        Wall(Line((0, 0), (0.61, 0)),                        label="bottom"),
        *bottom_baffle(0.65),
        Wall(Line((0.69, 0), (L, 0)),                        label="bottom"),
        OutletSeg(Line((L, 0), (L, H)),                      label="outlet"),
        Wall(Line((L, H), (b2_x + w, H)),                    label="top"),
        *top_baffle(b2_x),                    # mirror image, omitted
        Wall(Line((b2_x - w, H), (0, H)),                    label="top"),
        InletSeg(Line((0, H), (0, 0)), U=(1.0, 0.0, 0.0),    label="inlet"),
    ]
    return FlowDomain(outer=outer, holes=[], Re=Re, U_ref=1.0, L_ref=H,
                      end_time=end_time, write_interval=0.1)

PARAM_GROUPS = {                              # one group isolates one effect
    "re_sweep":         [dict(Re=re) for re in
                         [50, 100, 200, 300, 400, 600, 800, 1000, 1500, 2200]],
    "gap_sweep_Re0600": [dict(Re=600, gap_frac=g)
                         for g in [0.30, 0.35, 0.40, 0.45, 0.50, 0.55]],
    "b2_offset_Re0800": [dict(Re=800, b2_x=1.40 + dx)
                         for dx in [-0.20, -0.10, 0.10, 0.20, 0.30]],
}

for params in (p for group in PARAM_GROUPS.values() for p in group):
    domain = make_domain(**params)            # e.g. Re0600_G040 -> one trajectory
    domain.validate() or domain.generate(f"flow2dcases/{case_name(params)}/")
```

Each new geometry is designed with reference to the existing geometries, and this is why the output is a coherent dataset. The design has three strengths, (a) novelty is justified relative to concrete precedents, (b) engineering realism is a stated requirement, and (c) the loop runs without a solver, so a proposal costs minutes. It also has two limits, (a) novelty is assessed by the agent and checked only through the design note and the ban on reused names, so a family that is new in shape but not in mechanism can pass, and (b) the agent sees only two-dimensional internal flow, which bounds the physics the dataset can contain. Every family that completes this stage proceeds to the mesh design of Sec. A.2.

A.2 Mesh design

The mesh strategy of a family (Sec. 2) is a short script on predefined mesh APIs, and the stage verifies it on one trajectory with three meshes. The strategy assigns two global cell sizes and optional local refinement to a geometry, and scaling every size by one factor gives a coarser or finer mesh of the same design. The test builds three meshes on the hardest trajectory, usually the one with the highest Reynolds number. These are the coarse, the middle, and the fine mesh,

$$\big(\text{coarse},\ \text{middle},\ \text{fine}\big) = \big(\text{mesh-strategy}_{\times 2},\ \text{mesh-strategy}_{\times 1},\ \text{mesh-strategy}_{\times 1/2}\big), \quad (2)$$

where the subscript is the factor applied to every size except the near-wall refinement band, and it accepts the strategy when the estimated discretization error of the middle mesh is within tolerance for the velocity and pressure fields and the monitored integral quantities. Sec. B.1 defines the meshes, the quantities, and the acceptance rule in full, and this section uses only the result of the test. When the test fails, the agent proposes a new strategy from the failure report by Eq. (1), until a strategy passes. The rest of this section describes the strategy, the test, and the repair loop in this order.

A strategy is a class with two methods (Code 2). The first method is `params`, which maps a summary of the geometry to two global cell sizes. The second method is `refinement_fields`, which is optional and adds local sizing. The geometry summary contains the quantities required to derive sizes, the extent of the geometry, the fluid area and fraction, the narrowest passage, the channel width, the smallest obstacle, and the boundary labels with their types. A strategy derives every size from these quantities, so the same strategy transfers across the family it serves. Three built-in strategies are predefined. The medium one sets $h_{\min} = L/200$, capped at one fifth of the narrowest passage, and $h_{\max} = L/20$, capped at one sixth of the channel width and of the smallest obstacle, where L is the extent of the geometry corrected by the fluid fraction. A near-wall band then refines from $5h_{\min}$ to $2h_{\max}$. Sizes that violate the admissible range, such as $h_{\min} \geq h_{\max}$ or $h_{\max}/h_{\min} > 5$, are not accepted and are returned with an explanation, because a silently corrected size would falsify the history of the attempted sizes.

Code 2: A mesh strategy is a small class. `params` turns quantities of the geometry into two cell sizes, and the engine builds the three meshes of the convergence test by scaling those sizes, so the class does not mention a grid level. `refinement_fields` is optional and adds local refinement expressed relative to the already scaled sizes, which maintains the similarity of the three meshes. This strategy places five cells across the narrowest passage and refines the wake behind a baffle.

```python
class Scheme(MeshScheme):

    def params(self, geo):
        # geo: span, domain_size, fluid_area, min_gap, channel_width,
        #      obstacle_min_diameter, n_holes, patches, reynolds
        h_min = geo.min_gap / 5                  # five cells across the gap
        h_max = min(geo.domain_size / 20, geo.channel_width / 6)
        return MeshParams(h_min=h_min, h_max=h_max,   # must satisfy
                          wall_refinement=True)       # h_min < h_max <= 5 h_min

    def refinement_fields(self, ctx):
        # ctx.h_min, ctx.h_max are already scaled for the grid being built
        box = gmsh.model.mesh.field.add("Box")        # hold h_min in the wake
        gmsh.model.mesh.field.setNumber(box, "VIn",  ctx.h_min)
        gmsh.model.mesh.field.setNumber(box, "VOut", ctx.h_max)
        gmsh.model.mesh.field.setNumber(box, "XMin", 0.69)
        gmsh.model.mesh.field.setNumber(box, "XMax", 1.30)
        gmsh.model.mesh.field.setNumber(box, "YMin", 0.0)
        gmsh.model.mesh.field.setNumber(box, "YMax", 0.45)
        return [box]                                  # merged with the wall field
```

The engine verifies one trajectory of each family, the hardest one. Simulating three meshes for every trajectory would multiply the cost of the dataset by more than three, so the engine verifies the strategy on the hardest trajectory, usually the one with the highest Reynolds number, and applies the verified strategy to the entire family. The assumption is that this trajectory has the thinnest boundary layers and the strongest gradients, so its mesh error bounds that of the others. The three meshes of Eq. (2) are derived from the one strategy by scaling its sizes by 2 and 1/2 with the wall band unchanged, as shown in the middle row of Fig. 2. Every mesh is built with a build report that lists the requested and the actual sizes, the level, and the number of refinement fields the strategy created, and this report is the evidence that the strategy was applied. Before any solve, the engine builds only the middle mesh and discards the strategy if it exceeds 400,000 cells, because the fine mesh would be four times larger.

The convergence test then determines whether the strategy is accepted. The engine solves the hardest trajectory on the three meshes, compares the three solutions, and returns one result, pass or fail. The strategy passes when the estimated discretization error of the middle mesh is within tolerance. Otherwise it fails with one of three reasons, (a) an error above tolerance, (b) an achieved refinement ratio below the required minimum, because the geometry constrained the cell sizes (Eq. (4)), or (c) a quantity whose difference did not decrease under refinement (the condition $\rho \leq 1$ of Sec. B.1). A strategy that passes is applied to the entire family, and the family proceeds to the simulation stage with that strategy.

When the test fails, the agent rewrites the strategy from a failure report (Prompt 1). The failure report contains five items for the family, (a) the failure type with a brief note on what that type of failure implies for the mesh, (b) the geometry summary and the rendered preview of the fluid region, (c) a table of every earlier attempt with its sizes, cell counts, achieved ratios, and result, (d) the complete source and metrics of the most informative earlier attempts, and (e) the strategy interface of Code 2 with the acceptance criteria. The agent replies with one structured response (Response 1) that names the strategy, states its reasoning and the effect it expects, and contains the complete strategy. A proposal must then pass a validation ladder of five rungs before it costs any compute, (a) the reply must have the expected structure, (b) the name must be new and well formed, (c) the strategy must pass a static check that admits only the predefined mesh APIs and the standard mathematics module and forbids access to storage, processes, and the network, (d) the strategy must resolve on the actual geometry in an isolated process without error, and (e) a pessimistic cell estimate must remain within 30 times the cell cap. A failure at any rung is returned to the agent with its reason and retried up to three times, and these retries consume no compute. A proposal that passes the ladder becomes the next strategy of the family, and the family returns to the test.

The loop of Eq. (1) repeats until a strategy passes. Every attempt is retained with the family, the prompt, the raw reply, the strategy, the mesh build reports, the convergence report, and the result. An accepted mesh can therefore be audited from the first proposal to the accepted strategy, and the effect the agent expected is compared with the observed outcome in the next failure report.

Repair is both powerful and safe because the strategy is a script with an admissible range of sizes. The design has three strengths, (a) the agent can express local refinement beyond the two global sizes, (b) the three meshes of every test are similar by construction, because one strategy produces all of them, and (c) every size the agent specified is either built exactly or returned with a reason. It has one limit. The proposals of the agent are heuristics, so several rounds can be required before a strategy passes. Every strategy that completes this stage has passed the test of Sec. B.1, and Sec. A.3 applies it.

A.3 Simulation

Every trajectory is a solution of the two-dimensional incompressible Navier–Stokes equations for a Newtonian fluid, started from rest. The solver is `pimpleFoam` of OpenFOAM version 2406 (Weller et al., 1998). The inlet prescribes the velocity, the outlet prescribes the pressure, and the walls are no slip. A family may also specify moving or rotating walls

Prompt 1: One failure report as the agent receives it, abridged, the Apollonian packing channel at its third round. The five items of the report appear as its sections, the latest failure and what it implies, the geometry summary, the table of every earlier attempt, the complete strategy and metrics of the most informative attempts, and the task with the acceptance criteria.

```
# Mesh repair: base/apollonian-packing-channel (apollonian-Re500)

## Latest failure and what it implies
Latest verdict: fail:field_U
Velocity field differs too much between base and fine ...
When a previous trial came CLOSE (GCI 5-10%), do not nudge ...

## Geometry summary
{"span": 6.0, "min_gap": 0.120, "channel_width": 0.553,
 "obstacle_min_diameter": 0.200, "n_holes": 15, "reynolds": 500,
 "patches": {"walls": "wall", "inlet": "patch", "cyl_L1": ...}}

## Trial history (all attempts, oldest first)
| # | scheme | h_min | h_max | fields | cells c/b/f | verdict |
| 001 | default-medium | 0.024 | 0.0333 | 0 | 8.6k/34k/134k | fail:field_p |
| 002 | packed-cyl-wake-graded | 0.0075 | 0.0345 | 6 | 82k/316k/1.26M | fail:field_U |

## Most informative trials in full
### Trial 002 -- fail:field_U
Failure reasons: field U: not in asymptotic range: refinement did
not reduce the field difference (rho=0.92 <= 1) ...
[complete scheme.py and metrics of trials 002 and 001, mesh previews]

## Your task
Propose ONE new scheme that passes the acceptance criteria. Do not
repeat a failed parameter set; change what the evidence says is wrong.
```

Response 1: One mesh proposal as the agent returns it, abridged. The response names the strategy, states the reasoning and the effect the agent expects, and contains the complete class of Code 2. The stated expectation is compared with the observed outcome in the next failure report, so the agent is accountable for its own predictions.

```
{
  "scheme_name":     "wake-box-gap5",
  "rationale":       "L2(p) failed at 6.5% with the default strategy, but L2(U)
                      passed. The pressure error sits behind the baffles, so I
                      hold h_min in a box over the wake instead of lowering the
                      global h_min, which keeps the middle mesh under the cap.",
  "expected_effect": "cells 180k -> ~230k, L2(p) below 5%, L2(U) unchanged",
  "scheme_py":       "class Scheme(MeshScheme):\n    def params(self, geo): ..."
}
```

and lines of symmetry. The velocity and pressure fields are exported at an interval of 0.1 seconds on a uniform grid by resampling from the simulation mesh (Sec. A.4).

The verified strategy is applied to every trajectory of the family. For each trajectory the stage builds the middle mesh, the mesh the test verified, runs the solver, and exports the velocity and pressure fields on a uniform grid. Trajectories that share a geometry and differ only in the Reynolds number share one mesh, because the strategy derives its sizes from the geometry alone.

The trajectories share the solver settings. The time integration is implicit Euler, the convection scheme is a bounded second-order upwind scheme, and the Laplacian schemes are

linear with non-orthogonal correction. Each time step performs two outer pressure–velocity iterations with one pressure corrector and one non-orthogonal corrector. The linear solvers iterate to an absolute tolerance of 10^{-6}, with a relative tolerance of 0.01 for the pressure and 0.1 for the velocity, and to the absolute tolerance on the final corrector. The time step adapts to maintain the Courant number below 0.5 and does not exceed half the output interval.

The released trajectories are physically consistent and numerically verified solutions, and the results of the benchmark hold for the operator that produced them. Four facts support this conclusion. (a) Every trajectory is a physical flow. The equations are the incompressible Navier–Stokes equations of a Newtonian fluid, the boundary conditions are a prescribed inlet velocity, a prescribed outlet pressure, and no-slip walls, and the flow starts from rest, so each trajectory is the start-up flow of an internal geometry and the only idealization is that the geometry is uniform in the third direction. Most trajectories are below the transition of channel flow near a Reynolds number of 2,300, the range in which a two-dimensional simulation is the standard model of such an internal flow. Every trajectory is checked for mass conservation and for the Courant condition (Sec. B.2), the export conserves the integrals of the solution (Sec. A.4), and the flow features expected of each geometry appear in the reference fields, the vortex core of the hydrocyclone, the jet along the spillway steps, and the wakes behind the baffles (Figs. 7 and 9). (b) The mesh convergence test bounds the discretization error in time as well as in space. The time step follows the Courant condition, so the three meshes of the test refine the time step together with the cell size by the same factor of two, and the index of Eq. (8) bounds the combined error of the spatial schemes and the time integration, including the numerical dissipation of the first-order time integration and of the bounded upwind scheme. The observed orders of convergence are consistent with this combined refinement, with a median of 1.4 for the velocity field and 1.8 for the pressure field (Sec. B.1), between the first order of the time integration and the second order of the spatial schemes, and every family lies in the asymptotic range. The index of every family is computed with its own observed order, its median on the released mesh is 2.9%, and every family is below the gate of 5%. (c) The trajectories at higher Reynolds numbers pass the same test with the same bound, because the test verifies each family on the trajectory with its highest Reynolds number, and every trajectory is labelled with its Reynolds number, so a user can restrict an experiment to any range. (d) The conclusions of the benchmark are conclusions about the two-dimensional operator that produced the data. The dataset is two-dimensional, as are most of the datasets of Table 7. The solver is the standard transient incompressible solver of OpenFOAM with its standard schemes, the test is the standard verification procedure of computational fluid dynamics (Celik et al., 2008), and training and test trajectories are computed by the same solver with the same schemes and verified by the same test, so the error of Sec. 4 is measured relative to that operator, and the jump at a new family and the power law are relations between training and test data of one operator at every Reynolds number of the dataset. Therefore, every released trajectory is a physically consistent solution of the two-dimensional flow equations with a discretization error below 5% in space and in time, and the conclusions of the benchmark hold for the operator that produced the data.

The solution fields are exported on a uniform grid, one grid per trajectory. After the solve, the velocity and pressure at every output time are extracted from the mesh and then resampled to a cell-centred uniform grid that covers the bounding box of the geometry. The grid spacing is $\Delta x = \min(L_{\text{long}}/200,\, L_{\text{short}}/64)$, so the long axis has at least 200 cells and the short axis at least 64. This ensures that both elongated and compact geometries remain resolved at a bounded grid size, and the channel of Fig. 2 is exported as 224×64. The export contains the velocity and pressure fields at every output time of the family, a fluid mask, a boundary-type map, a boundary-label map, and the prescribed boundary values, so a neural operator receives the geometry, the conditions, and the solution in one grid format. Each trajectory is packed as one archive with its uniform-grid fields and a summary of the physics and the grid. Sec. A.4 describes the resampling step, and Sec. B.2 checks the loss it introduces.

The trajectories are comparable because they share the solver settings. The design has three strengths, (a) there is no manual tuning of individual trajectories, (b) the mesh that passed the convergence test is the mesh that produces the released trajectory, and (c) the uniform

export gives a neural operator the geometry, the conditions, and the solution in one format. It has one limit. The resampling to a uniform grid loses detail at walls, and Sec. B.2 checks this loss. The stage delivers a set of trajectories on verified meshes, and Sec. B.2 describes the checks that each trajectory passes.

A.4 Conservative resampling

The engine resamples each snapshot to the uniform grid by exact area-weighted averaging. The solver computes one value per mesh cell, and in a finite-volume discretization that value is the average of the field over the cell. The resampling assigns to each cell of the uniform grid the average of the same field over its fluid part,

$$\text{grid-cell-value} \;=\; \frac{\sum \text{overlap-area} \times \text{mesh-cell-value}}{\sum \text{overlap-area}}, \tag{3}$$

where the sums run over the mesh cells that intersect the grid cell, and the overlap area is the area shared by the grid cell and one mesh cell. A grid value is therefore the same type of quantity as a mesh value, an average of the solution over a known region. The export coarse-grains the finite-volume solution rather than interpolating it.

The overlap areas are computed exactly from the cell geometry. The two-dimensional mesh is one extruded layer of cells, so every mesh cell has a polygonal footprint in the flow plane. The engine extracts the footprint of every cell and verifies each one. The footprint area times the layer thickness must reproduce the volume of the cell. Each cell of the uniform grid is a square, and intersecting the square with the footprints it overlaps gives the overlap areas of Eq. (3). The overlap areas depend on the meshes and not on the fields, so they are computed once per trajectory and applied to every snapshot.

A grid cell cut by a wall is covered only in part by fluid, and the export retains the exact coverage. The covered fraction of every grid cell follows from the same intersections, and it is released with the fields as the fluid mask of Sec. A.3. The field value of such a cell is the average over its covered part only, so the released fields contain no wall or solid value, and the prescribed boundary values appear only in the boundary channels of the export.

The averaging preserves two properties of the finite-volume solution. First, it conserves every field over the fluid region. The overlap areas of one mesh cell sum to its footprint area, so the fluid-area integral of the velocity and of the pressure on the uniform grid equals the integral on the mesh, up to rounding. Second, it is bounded. Every grid value is an average with non-negative weights, so the released fields contain no value outside the range of the mesh solution, and sharp gradients produce no overshoot.

The released fields are coarse-grained finite-volume solutions with a known relation to the mesh solution. The design has three strengths, (a) the resampling conserves the fluid-area integral of every field, (b) a cell at a wall contains an average of fluid values together with its exact covered fraction, and (c) the overlap areas are computed once per trajectory, so resampling a snapshot costs one weighted sum per grid cell. It has two limits, (a) averaging removes the variation below the grid spacing, so peaks and quadratic quantities such as the kinetic energy are lowered, and Sec. B.2 checks this loss, and (b) the footprints exist because the mesh is one extruded layer, so the construction does not apply to a mesh that varies through the thickness. The step delivers the uniform-grid fields that the checks of Sec. B.2 examine.

A.5 Parallel execution

The engine computes many families in parallel with independent jobs. A job infers the stage of a family from what the family has produced, such as whether a convergence report exists, whether it passed, whether a new strategy awaits its test, and whether every trajectory has been exported. A summary of all families therefore always agrees with the data. Each job processes a single family, performs the next step that the family requires, and then releases it. A family assigned to one job is invisible to the other jobs, and a job that crashes releases its family immediately. Throughput is scaled by starting more identical jobs.

Each stage is parallel over the items that suit it. Proposing and mesh design run one family per job, and simulation runs one trajectory per job. First, proposing new geometries requires

the agent and no solver, so several agent sessions generate families concurrently, and the engine admits them sequentially, which is where a duplicate name is detected. Second, mesh design is one family per job. The job solves the three meshes of its family concurrently, and a node runs 16 such tests concurrently. Within this stage, the agent's repairs are separated from the tests that evaluate them. A repair requires the agent and runs wherever the agent is reachable, and the failure reports of many failed families are sent as concurrent requests. Each proposal that passes the ladder is retained with its family as the strategy awaiting a test, and the next job that processes the family runs that test. Proposing therefore does not wait for compute, and testing does not wait for the agent. Third, simulation is one trajectory per job rather than one family, so a family with a thousand trajectories is shared by many jobs, and a node runs 64 families with four trajectories each. Each trajectory runs on a single core, because the two-dimensional meshes contain at most 400,000 cells and many concurrent trajectories give higher throughput than one trajectory split across cores.

Every stage resumes from what the family has produced, so an interrupted batch does not repeat finished work. In the simulation stage, a trajectory whose exported fields exist is skipped, a trajectory with solver output but no exported fields is only post-processed, and any other trajectory is solved from the start. In the mesh stage, a family with a completed test is not tested again unless a new strategy awaits. A job continues to process families until none remains. When no unassigned family remains, the job waits until its peers have completed the families in progress and then exits. On the cluster each job occupies one node with 256 logical cores and its full memory, and a batch is a set of identical jobs over the same families.

Failures are contained, so one failing family cannot stall the batch. A step that fails is retried, and every solve runs under a time cap.

The design is simple and robust because it has no central scheduler. It has four strengths, (a) it scales by adding jobs, (b) it restarts from any interruption without repair, (c) it does not duplicate work, and (d) it retains every step with the family. It has two limits, (a) single-core solves bound the speed of any one trajectory, so the most expensive families dominate the tail of every batch, and (b) the design requires shared storage on which a job can reserve a family exclusively, which rules out some network mounts. Producing the dataset consumed approximately 54K CPU hours. Every family of the dataset passed the mesh convergence test.

B Dataset verification

Two separate guarantees support every trajectory of the dataset. The first bounds the error that arises from the finite mesh, and the mesh convergence test establishes it for each family. The second covers the error sources that a convergence test cannot detect, which are mesh quality, solver convergence, and the export step. We establish it by examining the delivered results directly.

B.1 Mesh convergence verification

Every family of the dataset has an estimated numerical uncertainty for the fields that we release. A mesh strategy S (Sec. 2) assigns a target cell size to every point of the geometry Ω. From one strategy we build the three meshes of Eq. (2), written M_k, $k = 1, 2, 3$ from fine to coarse, by multiplying all target sizes of S by $r^{\,k-2}$ with nominal refinement ratio $r = 2$, and we simulate the hardest trajectory of the family on all three. The middle mesh M_2 is the mesh that produces the released trajectories, so the uncertainty we report is the uncertainty of the delivered data rather than of a finer solution that we do not release. A user of the dataset therefore receives an error bar that applies to the fields they use.

The three meshes are refined systematically, and we verify this from the cell counts. Because all three are derived from the same strategy with its size field multiplied by a constant, the target size scales by the same factor at every point of the geometry. The near-wall refinement band is the one quantity that remains unchanged across the three meshes, because it is a region of space rather than a cell size. If it scaled, the wall and the channel core would be refined by different factors, and the geometric similarity that the analysis assumes would be lost. The refinement that the mesh generator achieved is then computed from the cell

counts N_k. For a two-dimensional geometry of fluid area A the representative cell size and the achieved ratio of a mesh pair are

$$h_k = \left(\frac{A}{N_k}\right)^{1/2}, \qquad r_{21} = \frac{h_2}{h_1} = \left(\frac{N_1}{N_2}\right)^{1/2}, \qquad r_{32} = \frac{h_3}{h_2} = \left(\frac{N_2}{N_3}\right)^{1/2}. \tag{4}$$

The test fails when $\min(r_{21}, r_{32}) < 1.3$, the separation recommended for a three-mesh test (Celik et al., 2008). Meshes that did not separate contain no information about mesh error, even if their differences appear negligible. The achieved ratios of the released families have a median of 1.96, and none is below 1.3.

The test monitors two fields and four integral quantities, all evaluated on the fluid region. Let $\Omega_f \subset \Omega$ be the part of the geometry occupied by fluid, $\mathbf{u}$ the velocity field and p the kinematic pressure. Every monitored quantity is computed from the time average over the second half of the simulated span $[0, T]$ of the test, where T is the end time of the simulation,

$$\bar{\phi}(\mathbf{x}) = \frac{2}{T} \int_{T/2}^{T} \phi(\mathbf{x}, t)\, \mathrm{d}t, \tag{5}$$

which suppresses the sampling noise of individual snapshots. Table 1 lists the monitored quantities, their definitions, and the rule that each must satisfy. The three solutions are resampled to one common uniform grid before they are compared, because a pointwise field comparison requires a common grid.

The uncertainty of each quantity is estimated with the grid convergence index. For a scalar quantity with values ϕ_1, ϕ_2, ϕ_3 on the three meshes we form the differences and their ratio,

$$\varepsilon_{21} = \phi_2 - \phi_1, \qquad \varepsilon_{32} = \phi_3 - \phi_2, \qquad \rho = \frac{\varepsilon_{32}}{\varepsilon_{21}}. \tag{6}$$

The observed order of convergence p is the positive root of

$$\rho = r_{21}^{p}\, \frac{r_{32}^{p} - 1}{r_{21}^{p} - 1}, \tag{7}$$

which reduces to $p = \ln \rho / \ln r$ when the two achieved ratios are equal (Celik et al., 2008). We solve Eq. (7) by bisection, because the achieved ratios of Eq. (4) are not exactly equal and the classical fixed-point iteration can fail for unequal ratios. The uncertainty of the released mesh M_2 is then

$$\mathrm{GCI}_2 = F_s\, \frac{|\varepsilon_{32}|}{|\phi_1|}\, \frac{1}{r_{32}^{p} - 1}, \qquad F_s = 1.25, \tag{8}$$

with the safety factor recommended for three-mesh tests (Roache, 1994; Celik et al., 2008). A quantity with $\rho \leq 1$ has not reduced its error under refinement, so no positive p exists and the triplet lies outside the asymptotic range. We then assign no uncertainty and treat the quantity as failed.

Fields use the same estimator with a norm in place of a difference. For a field $\bar{\phi}$ we replace Eq. (6) by

$$\varepsilon_{21} = \left\|\bar{\phi}_2 - \bar{\phi}_1\right\|_{L^2(\Omega_f)}, \qquad \varepsilon_{32} = \left\|\bar{\phi}_3 - \bar{\phi}_2\right\|_{L^2(\Omega_f)}, \tag{9}$$

and normalise both by $\|\bar{\phi}_1\|_{L^2(\Omega_f)}$, so their ratio ρ is exact. Equations (7) and (8) then give an observed order and an uncertainty for the complete velocity and pressure fields. A norm cannot change sign, so for a field the single condition $\rho \leq 1$ is the complete statement that refinement did not reduce the error.

The index of every family is evaluated with its own observed order of convergence. Over the families of the dataset the observed order of the velocity field has a median of 1.4 with an interquartile range of 1.1 to 1.8, and that of the pressure field a median of 1.8 with a range of 1.3 to 2.3. A family that converges slowly therefore receives a larger uncertainty for the same difference between its meshes.

Reporting GCI_2 rather than the raw difference between two meshes is important, because the two can disagree by a large factor. The difference ε_{21} is the change of the solution in one refinement step. Eq. (8) instead estimates the remaining distance between the released solution and the exact one, and that distance depends on how fast the sequence converges. A difference of 5 percent with $p \approx 3$ implies a much lower uncertainty than the same difference

Algorithm 1 Mesh convergence verification of one family

Require: mesh strategy S and tolerance τ
for $k = 1, 2, 3$ **do**
  build M_k from S with all target sizes scaled by $r^{\,k-2}$
  simulate the hardest trajectory on M_k and obtain the cell count N_k
end for
compute r_{21}, r_{32} from Eq. (4)
if $\min(r_{21}, r_{32}) < 1.3$ **then**
  return fail, the three meshes are not separated
end if
resample the three solutions to one uniform grid and time-average by Eq. (5)
for all monitored quantities ϕ of Table 1 **do**
  form $\varepsilon_{21}, \varepsilon_{32}, \rho$ by Eq. (6) or Eq. (9)
  if $\rho \le 1$ **then**
    mark ϕ failed ▷ outside the asymptotic range
  else
    solve Eq. (7) for p and evaluate GCI_2 by Eq. (8)
  end if
end for
return accept if every gated quantity of Table 1 satisfies its rule

Table 1: Monitored quantities of the mesh convergence test and the rule each must satisfy. All quantities are time-averaged by Eq. (5) and evaluated on the fluid region Ω_f. GCI_2 is the uncertainty of the released mesh, Eq. (8). Inlet and outlet averages are taken over the fluid cells adjacent to the corresponding boundary patch.

Quantity	Symbol	Definition	Acceptance rule
Velocity field	$\mathbf{u}$	$L^2(\Omega_f)$ norm, Eq. (9)	$\mathrm{GCI}_2 < 5\%$
Pressure field	p	$L^2(\Omega_f)$ norm, Eq. (9)	$\mathrm{GCI}_2 < 5\%$
Mean speed	$\bar{U}$	$\lvert\Omega_f\rvert^{-1}\int_{\Omega_f}\lvert\mathbf{u}\rvert\,\mathrm{d}A$	$\mathrm{GCI}_2 < 5\%$
Kinetic energy	E	$\int_{\Omega_f}\frac{1}{2}\lvert\mathbf{u}\rvert^2\,\mathrm{d}A$	$\mathrm{GCI}_2 < 5\%$
Pressure drop	Δp	$\langle p\rangle_{\mathrm{inlet}} - \langle p\rangle_{\mathrm{outlet}}$	$\mathrm{GCI}_2 < 5\%$
Peak speed	$U_{\max}$	$\max_{\Omega_f}\lvert\mathbf{u}\rvert$	$\mathrm{GCI}_2 < 5\%$

with $p \approx 1$. The index is also the quantity a reader can compare with a tolerance, because it is an error estimate rather than an increment. Algorithm 1 states the complete procedure.

This procedure follows standard verification practice (Celik et al., 2008). Its strengths are that every family has its own uncertainty, that systematic refinement is verified through Eq. (4), and that the reported number applies to the mesh we release. Its limit is that a scalar quantity whose differences change sign between mesh pairs, that is $\rho < 0$, is discarded instead of being treated by the oscillatory branch of the standard procedure. This choice is conservative. The index bounds the discretization error, and Sec. B.2 checks the remaining error sources.

B.2 Dataset sanity checks

A convergence test bounds one error source, so the engine checks the others on the delivered results. Three conditions remain unverified after a family converges, (a) the mesh is of usable quality, (b) the solver converged on that mesh, and (c) the export step preserved the solution. Table 2 lists the quantity that verifies each condition, its source, and the failure it detects. All of them are produced by the engine as it runs.

The engine checks the quality of every mesh. For each of the three meshes it obtains the maximum non-orthogonality $\theta_{\max}$, the maximum skewness $\sigma_{\max}$, and the maximum aspect ratio $\mathrm{AR}_{\max}$ from the mesh checker. This check is independent of convergence, because a defect shared by all three meshes degrades the discretization without producing any disagreement between them, and Eq. (8) would then return a low uncertainty for a solution that is uniformly inaccurate.

Every trajectory is checked for solver convergence. The engine obtains the cumulative continuity error c at the end of the trajectory, which indicates how well the pressure solution conserved mass, the maximum Courant number $\mathrm{Co}_{\max}$, which bounds the time-step size relative to the mesh, and the time-step collapse ratio

$$\kappa = \frac{\Delta t_{\mathrm{final}}}{\max_t \Delta t}. \tag{10}$$

The last quantity detects a specific failure mode. The solver does not report an error when a solution diverges. It shrinks its adaptive time step instead, so a diverged trajectory can terminate normally after computing no physical progress. This check ensures that no diverged trajectory enters the dataset. Iterative convergence is an assumption of every discretization analysis, and these three numbers are the evidence that the assumption is satisfied here.

The released fields are produced by resampling each solution to a uniform grid, and the engine checks what the resampling costs. The resampling is the conservative averaging of Sec. A.4. It preserves the fluid-area integral of every field, but it removes the variation below the grid spacing. The cost of this step is the relative difference between an integral quantity ϕ of Table 1 computed on the uniform grid and the same quantity computed on the mesh,

$$\eta_\phi = \frac{\left|\phi^{\mathrm{uniform}} - \phi^{\mathrm{mesh}}\right|}{\left|\phi^{\mathrm{mesh}}\right|}, \tag{11}$$

which separates the loss introduced after the physics was solved from the error present in the solution.

Table 2: Checks applied to every trajectory of the dataset, beyond the mesh convergence test. Each targets a failure that the convergence test cannot detect.

Quantity	Symbol	Source	Failure it detects
Max non-orthogonality	$\theta_{\max}$	mesh checker	poorly shaped cells on all three meshes
Max skewness	$\sigma_{\max}$	mesh checker	cell centres far from face centres
Max aspect ratio	$\mathrm{AR}_{\max}$	mesh checker	extreme stretching near walls
Cumulative continuity	c	solver log	mass not conserved by the pressure solution
Max Courant number	$\mathrm{Co}_{\max}$	solver log	time step too large for the mesh
Time-step collapse	κ, Eq. (10)	solver log	silent divergence with a frozen clock
Resampling loss	η_ϕ, Eq. (11)	export step	error introduced after the solve

A verified trajectory is a trajectory that passed the convergence test and these checks. The strength of the checks is that each targets a failure the others cannot detect. The mesh checks detect a defect shared by all three meshes, the solver checks detect a trajectory that finished without converging, and the export check detects a loss introduced after the physics was solved. Their limit is that they are diagnostics for known failure modes. Every trajectory of the dataset is therefore computed on a mesh whose discretization uncertainty is bounded by Eq. (8) and reported with the data, is computed by a solver that converged, and is exported with a checked resampling loss.

C Dataset statistics and figure construction

This appendix expands Sec. 3 with the complete statistics of the dataset, the construction of Figs. 4 and 5, and the repair of five families in detail.

C.1 Statistics of the dataset

The dataset is diverse in geometry, in the number of inlets and obstacles, and in the geometric parameters and Reynolds numbers, and Fig. 4a reports its key numbers and distributions. It contains 459 families, 3,948 distinct geometries, and 16,512 trajectories. The number of geometries and of Reynolds numbers differs across families. A family sweeps between 1 and 81 geometries and between 1 and 56 Reynolds numbers, and the largest family contains 326 trajectories. Every trajectory provides the velocity and pressure fields at an interval of 0.1 seconds. The fields are resampled to a uniform grid with a spacing of

$\min(L_{\text{long}}/200, L_{\text{short}}/64)$, so the long axis of a geometry has at least 200 cells and the short axis at least 64 (Sec. A.3). In total the dataset contains 458,001 snapshots and 10.8 billion grid points. The flow is incompressible and two-dimensional, solved with OpenFOAM (Weller et al., 1998) (Sec. A.3). The Reynolds number spans 100 to 40,000, with a median of 500. The aspect ratio of the geometry spans 1 to 23, and the fraction of the bounding box occupied by fluid spans 0.12 to 1. The released meshes contain between 14k and 386k cells, with a median of 119k (Sec. C.3).

The families fall into 10 categories (Fig. 4b), which Sec. C.2 describes sequentially.

The map of Fig. 4a places every geometry in a shape space by dimension reduction. Each distinct geometry is rasterized from its fluid region on a 48 by 96 canvas, with the longer side of the geometry filling the canvas. The raster is reduced by a principal component analysis to 40 components, the logarithms of the longer side and of the aspect ratio are appended as two further features, and t-SNE embeds the result in two dimensions. Each dot is one geometry, coloured by the category of its family.

Three statistics of Sec. 3.1 quantify the family structure of this space, computed from the rasters of the 3,948 geometries. (a) The share of the shape variance between families. The total variance is the mean squared distance of a raster from the mean raster of the dataset, the within-family variance is the mean squared distance of a raster from the mean raster of its own family, and the share between families is one minus their ratio, 92%. (b) The number of principal components that explain 90% of the variance. It is 70 for the dataset and a median of 4 over the 298 families with at least six geometries, each analysed on its own. (c) The separation of the families. In the space of the first 40 components, the distance from the mean of a family to the nearest mean of another family is divided by the root mean square distance of the family's geometries from their mean. The median of this ratio over the 417 families with more than one geometry is 3.8, and the ratio exceeds 1 for 90% of them. The three statistics are generated by the script that renders Fig. 4.

The remaining panels of Fig. 4 are computed from the released families as follows. The tinted regions of the map contain the denser half of the families of one category, and the drawings on the map are families near the centre of their neighbourhood, placed on empty parts of the map. The category of a family is assigned from the words of its name by a fixed list of rules. The six statistics are kernel density estimates over the families, or over the trajectories for the Reynolds number and the mesh cells, drawn from the minimum to the maximum of the axis, with the median as a white line. The tables count families. The eight drawings of a category in Fig. 4b are selected to span the outlines of the category, by clustering the rasters of its families into eight groups and drawing the family with the most boundary segments of each group, so simple geometries are under-represented in the drawings.

Fig. 4 in detail. **a**, Left, the dataset in numbers. The six density bands show the coverage of six axes from the minimum to the maximum of each axis, with the median as a white line. The three tables count the families by inlets and outlets, by Reynolds numbers and geometries in the sweep, and by the curve primitives of their boundaries, and the shade of a cell is its share of all families. Right, the shape space of the 3,948 geometries. Each dot is one geometry, placed by the embedding described above and coloured by the category of its family, with the ten categories numbered as in b. The tinted regions contain the denser half of the families of each category. The framed drawings are families at their positions on the map. The ten small maps show one category each over the whole set in grey, with the number of its geometries. **b**, The ten categories, each with its number of families and geometries and eight of its families drawn from their own code, walls in grey, inlets in teal, outlets in green.

C.2 The ten application categories

Fig. 4b groups the families into 10 categories, which are described below in the order of their number of families. Each entry names the flows of the category and three of the families the engine generated for it.

Cyclones (108 families, 823 geometries) are swirl separators, in which a tangential inlet drives the flow through a conical or tapered body. Examples are the conical hydrocyclone, the tapered classifier, and the planar desander.

Machinery (70 families, 633 geometries) contains the flow passages of rotating machines, volutes and scrolls, blade cascades, and the clearances of seals and bearings. Examples are the log-spiral volute with a cutwater, the Kaplan wicket gate cascade, and the spiral-groove mechanical seal.

Hydraulics (48 families, 408 geometries) contains open-channel hydraulic structures, spillway crests with their chutes, stilling basins, and measuring flumes. Examples are the ogee spillway with a flip bucket, the Parshall flume, and the braided river.

Settlers (45 families, 381 geometries) contains gravity and inertial separators, packs of inclined plates, chevron packs, and arrays that displace a stream laterally. Examples are the lamella inclined-plate clarifier, the chevron mist eliminator, and the deterministic lateral displacement separator.

Benchmarks (43 families, 427 geometries) contains the canonical configurations of the literature, obstacles and obstacle arrays in a channel, walls with ribs or dimples, and wedges and steps. Examples are the hexagonal pin array, the Jeffery–Hamel wedge, and the channel whose wall is a random Bézier curve.

Mixing (42 families, 346 geometries) contains mixers and heat exchangers, meandering and split-and-recombine passages, grooved and finned walls, and plate packs. Examples are the split-and-recombine micromixer, the staggered herringbone groove, and the pin-fin cold plate.

Flow control (34 families, 296 geometries) contains the devices that accelerate, deflect, or straighten a stream, nozzles and venturis, diffusers and valves, and jets and impingement chambers. Examples are the swing check valve, the multi-jet impingement chamber, and the honeycomb venturi straightener.

Biomedical (27 families, 233 geometries) contains vessel and valve geometries, stenoses and aneurysms, arterial bifurcations, and the sinuses of heart valves. Examples are the tandem stenosis, the curved Dean bifurcation, and the bileaflet aortic sinus.

Piping (25 families, 229 geometries) contains networks of connected passages, junctions and trifurcations, manifolds and distributors, and loops and bypasses. Examples are the Wheatstone bridge network, the parallel venturi manifold, and the ring main.

Fuel cells (17 families, 172 geometries) contains the flow fields of bipolar plates, serpentine and multi-pass channels separated by lands, and fields whose channel is supported by pillars. Examples are the PEM electrolyser serpentine flow field, the four-pass field with pillar supports, and the alkaline electrolyser with pillar lands.

C.3 Mesh repair statistics

Of the 459 families in the dataset, 35 passed the test on the default mesh strategy and 424 after repair (Fig. 5a). The repaired families passed after 2.6 repair rounds on average, each round being one strategy proposed by the agent. The most common number of rounds is 2, with 112 families. The longest repair took 6 rounds, in 10 families. In total the agent proposed 1,107 mesh strategies for the repaired families.

The default mesh strategy fails most often on the velocity or the pressure field (Fig. 5a, second chart). Among the 424 repaired families, the first check the default mesh failed is the velocity field in 47% and the pressure field in 26%. The pressure drop, the kinetic energy or the mean speed fails first in 3%. In the remaining families the test did not run. The released mesh, the middle of the three, exceeded the cap of 400k cells in 19% of the repaired families, and the solver failed in 6%.

The strategy that passed changes the cell size globally in half of the repaired families and grades the mesh in the other half (Fig. 5a, third chart). The global change is a refinement in 36% of the repaired families and a coarsening in 16%. The graded strategies are finer at the inlet and outlet in 33% of the repaired families, in local regions in 24%, at the outer walls in 21% and at obstacle walls in 17%. Many graded strategies refine several of these places, so the four shares add up to more than the graded half.

Repair brings the grid convergence index under the gate and the cell count under the cap (Fig. 5a, fourth and fifth charts). The larger of the two field indices has a median of 15% on the default mesh and of 2.9% on the released mesh. The default meshes have between

3.4k and 245M cells, and 81 of them exceed the cap of 400k cells. The released meshes have between 14k and 386k cells, with a median of 119k cells and an interquartile range of 63k–213k cells.

The five charts of Fig. 5a are computed over the families of the dataset, from the test history the engine retains with each family. (a) The repair rounds of a family are the number of strategies the agent proposed before the one that passed, so a family that passed on the default strategy has zero rounds. (b) The failure of the default mesh is the first check it failed, in the order of Table 1, or the reason the test did not run. (c) The repair strategy is classed on the released mesh. The mesh is uniform if its cell size at the 90th percentile of the area of the geometry is less than twice the size at the 10th percentile, and a uniform mesh is a global refinement or coarsening according to its cell count relative to the default mesh. A graded mesh is classed by the boundaries and regions its strategy refines, with each boundary named by the geometry as an inlet or outlet, an outer wall or an obstacle, and with boxes and balls as local regions. (d) The grid convergence index of a family is the larger of the two field indices, Eq. (8), on the default and on the released mesh, over the 301 repaired families for which both exist. (e) The cell count is that of the released mesh before and after repair, for all 424 repaired families.

Fig. 5a in detail. The five charts report the mesh repair statistics over the families of the dataset, left to right. The first chart counts the agent repair rounds per family, where 0 (grey) means that the default mesh passed and k rounds means acceptance at round Rk+1. The second chart counts the failure of the default mesh, in red a gated quantity of the convergence test and in grey a test that could not run because the mesh exceeded the 400k-cell cap or the solver failed. The third chart classes the repair strategy. The released mesh is uniform (grey) if its cell size is within a factor of two between the 10th and the 90th percentile of the area of the geometry, and a uniform mesh is a global size change relative to the default mesh. Otherwise the mesh is graded (blue) and finer at the inlet and outlet, in local regions (boxes and balls around features), at the outer walls, or at obstacle walls, and a graded mesh counts in every row it refines. The fourth chart shows the grid convergence index before repair (default mesh, grey) and after repair (released mesh, blue). For each family and mesh the index is the larger of the GCI_2 of the velocity and of the pressure field, Eq. (8), over the 301 repaired families for which the index exists on both meshes. The curves are percentile curves, and the height at percentile q is the value below which q% of the families lie. The red line is the 5% gate. The fifth chart shows the cell count of the released mesh before repair (default mesh, grey) and after repair (released mesh, blue) as percentile curves in the same way, with the 400k-cell cap as the red line.

C.4 Mesh repair examples

Fig. 9 shows the repair of five families, one per row. In each family the default mesh is close to uniform and fails on at least one field, and the strategy that passed grades the mesh. In the Apollonian packing the default mesh fails on the pressure field and the kinetic energy. A uniform refinement to 316k cells passes the pressure field but loses the index of the velocity field. Collars around the cylinders over a coarse bulk pass every check with 128k cells. In the flip-bucket spillway the default mesh fails on the pressure field and the pressure drop. Three overlapping wall bands with a halved cell size leave the pressure drop at 8%. One band per boundary passes with 127k cells. In the gradient generator the default mesh of 10k cells fails on all five quantities. A uniform mesh with 30 cells per lane passes the velocity field but leaves the pressure field at 5.5%. Refinement at the septa ends with coarser lanes passes with 133k cells. In the cyclonic louver scroll the default mesh fails on the pressure field and the pressure drop. Bands on the louvers reduce the pressure index to 7.4%, and wider bands then resolve the shedding wake, so the velocity index is lost. Narrow bands with finer ports pass with 300k cells. In the multi-jet impingement the default mesh fails on both fields and the pressure drop. A uniform fine mesh leaves the velocity field and the pressure drop above the gate, and wall and port bands over a coarse bulk leave the pressure field at 5.4%. A bulk cell size 15% finer passes with 32k cells. In four of the five families the released mesh has fewer cells than a mesh the test had failed.

The repair loop places cells where the flow requires them. In every family of Fig. 9 the default mesh is nearly uniform and fails where the flow is complex, and the strategy that passed



Figure 9: **Mesh repair examples.** Panel b of the mesh repair figure, one family per row. Left to right, the geometry of the hardest trajectory (inlet teal, outlet green, solids grey), then the mesh of every repair round R1, R2, … in one or two fixed windows of the geometry (dashed, i and ii where there are two), coloured by cell size from light (coarse) to dark (fine) and labelled with the cell count, then the solved trajectory as speed at the last output time from blue (slow) to red (fast). R1 is the default mesh strategy. Every later round is the agent's response to the previous failure report, and what it changed is written on the connector under the robot mark. Under each mesh are five of the six quantities of the convergence test (Table 1), the velocity and pressure fields $\mathbf{u}$, p, the pressure drop Δp, the kinetic energy E and the mean speed $\bar{U}$, each with its GCI_2 in percent and a mark, check for passed and cross for failed. n/a marks a field whose three-mesh differences did not decrease under refinement, so no index exists. A tilde marks a scalar whose three-mesh values agree within the number shown, in percent, so no index exists. The framed mesh passed every check and is the mesh of all trajectories of the family.

grades the mesh, with collars around the cylinders, one band per boundary, refinement at the septa ends, narrow louver bands with finer ports, and a finer bulk over wall and port bands. Across the dataset, half of the repaired families pass with a graded mesh, and the released mesh has fewer cells than a failed mesh in 68% of the families for which both exist. The loop therefore verifies a family at the cost of 2.6 repair rounds on average, each of three solver runs of its hardest trajectory, and every strategy it produces is a short script that a human can review. The loop has one limit. The released mesh is the first strategy that passed, so a later strategy could pass with fewer cells.

D Benchmark details

This appendix expands Sec. 4 with the construction of the splits, the architectures and the training protocol, and the metrics with the construction of Fig. 7.

D.1 Benchmark splits

The benchmark constructs four test splits along the ladder, G1-1, G1-2, G2, and G3, and one validation split. Each test split changes one factor relative to the training split. G1 changes the Reynolds number of a trained geometry, G2 changes the geometry within a trained family, and G3 changes the family. The construction depends only on the geometries and the Reynolds numbers, so it uses no information from the reference solutions.

The benchmark is built on a fixed release of the dataset, so every architecture is trained and evaluated on the same families and every reported number is reproducible. The splits are drawn from the 243 families of this release, 206 in the training split and 37 in the G3 test split. The engine has continued to produce families after this release, and the 459 families of Sec. 3.1 form the base of the next release of the benchmark.

G1 comprises two test splits. G1-1 is interpolation to an interior Reynolds number of a trained geometry, with both a lower and a higher Reynolds number of the same geometry in training. G1-2 is extrapolation to the trajectory with the highest Reynolds number of that geometry, so training contains only lower Reynolds numbers of the same geometry. The geometry is unchanged in both sets.

The test splits differ in depth. The G3 test split consists of whole families, the G2 test split of geometries, and the G1-1 and G1-2 test splits of single trajectories. A geometry of a family is the set of its trajectories whose rasterized fluid regions are identical. Each geometry is represented by the 16 leading principal components of its rasterized fluid region, and a family by the mean of its geometries. Flow descriptors are excluded from the construction.

The G3 test split is drawn as whole clusters of families. Families are clustered by the distance between their mean shape embeddings, with a threshold equal to the median distance from a family to its nearest neighbor, and families whose names share a token that appears in at most eight families are placed in one cluster, because descendant families generated by the engine inherit the concept in their names. Clusters are drawn in a random order fixed by a seed until the families in the test split reach 15 percent of the dataset, and a cluster larger than 8 percent of the dataset is not drawn. A drawn family that lies closer to a training family than the median neighbor distance returns to training with its cluster, and a replacement cluster is drawn. The loop repeats until every family in the test split is farther from every training family than this distance, so a G3 test family has no close relative in training.

The G2 test split consists of geometries of the remaining families. A family with at least three geometries contributes 15 percent of them, under two constraints, (a) the geometry with the most trajectories remains in training, and (b) the Reynolds numbers of a contributed geometry lie inside the range of the family's remaining geometries. The second constraint maintains the geometry as the only changed factor.

The G1-1 test split, the G1-2 test split, and the validation split consist of trajectories of the geometries that remain in training. For each such geometry with at least five Reynolds numbers, the highest value goes to G1-2. For each geometry with at least four remaining Reynolds numbers, the two endpoint values remain in training, one interior value goes to the validation split, and one interior value goes to G1-1, taken at one third and at two thirds of the sorted values. A G1-1 trajectory therefore has its exact geometry in training at both lower and higher Reynolds numbers. A G1-2 trajectory has its exact geometry in training only at lower Reynolds numbers. All remaining trajectories form the training split.

The validation split is the only split that influences training. Early stopping, checkpoint selection, and learning-rate selection use the validation split, and the test splits are evaluated once, after training and selection are complete. The reported unseen accuracy is therefore the accuracy of neural operators selected by seen accuracy, the condition a deployed surrogate faces.

The design has three strengths, (a) each test split changes one factor, so the error increase from one set to the next is attributable to that factor, (b) the construction uses no information from the reference solutions, so a split cannot encode the difficulty it is meant to test, and (c) the distance rule of G3 removes related families that clustering alone missed. It has two limits, (a) the G3 test families differ from the training families in geometric composition, so G3 error combines geometric novelty with composition, and (b) training receives no unseen feedback, so a neural operator cannot be tuned for transfer without violating the protocol. The construction delivers fixed training, validation, and test splits to the training protocol of Sec. D.2.

D.2 Architectures and training protocol

Every architecture receives the geometry as fields on the solution grid, a binary fluid mask, the boundary type of each cell, and the prescribed velocity and pressure on the boundary cells. It receives one snapshot of the velocity and predicts the next, and a rollout applies this step to its own output (Sec. D.3).

Every architecture is trained and evaluated through one interface, so the differences in Sec. 4 are differences of architecture. The input is a tensor of ten channels on the uniform grid, the two velocity components at the current output time, the fluid mask, the boundary type, the three prescribed boundary values (two velocity components and pressure), the two physical coordinates of each cell, and the Reynolds number as a constant channel. The output is the velocity at the next output time. Three operations are applied outside the architecture and are identical for all of them, (a) every channel is normalized before the architecture and the prediction is denormalized after it, (b) the architecture predicts an increment, which is added to the normalized current velocity, and (c) the loss is computed on the fluid cells only. An architecture therefore maps ten normalized channels to two normalized increments. The increment is the residual form of the step. The architecture learns the difference between two close physical states, as residual learning for neural operators does with the difference between a state and a similar reference state (Yue et al., 2025c).

Normalization is a per-channel affine map computed once on the training split. The mean and standard deviation of each channel are computed over the fluid cells of 256 training samples, and a channel is mapped to zero mean and unit variance. The channels differ in scale by orders of magnitude. The velocity is of order one, the coordinates are in physical length units and exceed ten units for the largest geometries, the prescribed pressure varies with the trajectory, and the Reynolds number is given as $\log(1+\mathrm{Re})/10$. Without normalization, a shared first layer receives these scales together, and the loss of a joint training across geometries develops spikes. The velocity channels share one statistic on the input side and the output side, computed on the current and the next snapshot together, so the increment is added in one coordinate system and the residual step is exact. A channel with a standard deviation below 10^{-3}, such as a mask that is constant on a trajectory, is left at its scale. The loss is computed in the normalized space, and all metrics in Sec. D.3 are computed in physical units after denormalization.

Every architecture receives position through the two coordinate channels. The coordinates are the physical cell centres of the uniform grid, so an architecture sees the absolute position and the size of the geometry, and two geometries at different scales are distinguishable. There is no learned positional embedding in any architecture. The convolutional and spectral architectures obtain position from the coordinate channels and from their own translation structure, and the transformer architectures, whose attention is invariant to the order of the cells, obtain position from the coordinate channels only, except HNO, which adds a learned bias for the relative position of two patches. UNet-c is the exception and receives position a second time through a coordinate path, described below. The prescribed boundary values are constant channels as well, so an architecture receives the inflow velocity and the outlet pressure at the cells where they apply and zero elsewhere.

Two U-Nets are evaluated, and they differ only in how they receive position. UNet is the plain network, the encoder-decoder structure of the U-Net (Ronneberger et al., 2015) in the modern form of PDEArena (Gupta & Brandstetter, 2023), an encoder and a decoder of three scales with a base width of 64, two convolutions with group normalization and GELU per scale, skip connections between the scales, and a bottleneck of four linear-attention blocks. Its coordinates are two of the ten input channels, and the first 3×3 convolution mixes them with the velocity and the boundary channels, as in CoordConv (Liu et al., 2018). UNet-c retains this main path unchanged and adds a coordinate path. The path takes four channels per cell, the two coordinates and the logarithms of the cell spacing in the two directions, maps them through two 1×1 convolutions to a feature of 32 channels, and average-pools the feature to the three scales of the main path. At five points of the main path, after the two encoder blocks, at the entry and the exit of the bottleneck, and after the two decoder blocks, a 1×1 network maps the coordinate feature of that scale to a scale γ and a shift β per cell and per channel, and the feature of the main path becomes $(1+\gamma)\,\mathrm{feature}+\beta$. This is feature-wise linear modulation (Perez et al., 2018) in its spatially adaptive form (Park

et al., 2019), with the coordinate map in place of a semantic map. The last layer of every modulation network is initialized at zero (Zhang et al., 2023), so UNet-c equals UNet at the start of training and learns how much position to use. The coordinate path has no spatial kernel, so it is invariant to the resolution of the grid, and it adds 0.13 million parameters to the 6.54 million of UNet. Neural PDE solvers use conditioning of this form for scalar quantities such as time and the PDE parameters (Gupta & Brandstetter, 2023; Takamoto et al., 2023). Here the condition is a field, the position of each cell.

The architectures follow the main configuration of their reference implementation in structure, with width and depth brought to one scale. The structural elements, the form of attention, the number of heads and slices, the number of Fourier modes, and the block structure, are those of the reference (Table 3). Width and depth are set to a hidden dimension of 32 to 64 and 4 to 8 layers for every architecture, so no architecture is wider or deeper than the others by a large factor. Model sizes are not equalized and are reported. FNO has width 32, 12 modes, and 4 layers. F-FNO has width 64, 16 modes, and 8 layers, with the layer normalization and the residual feed-forward block of the reference, and the depth of 8 replaces the 24 layers of the reference. U-NO has width 32 and three scales, with the number of modes scaled with the resolution of each scale. DilRes has 48 channels and four blocks of seven dilated convolutions with dilations 1, 2, 4, 8, 4, 2, and 1. CNO has 32 channels and three scales, and its up- and down-sampling use a fixed Gaussian low-pass filter with bilinear interpolation in place of the sinc filters of the reference, so it is a simplified re-implementation. UNet-c and UNet have a base width of 64 and three scales. Galerkin has dimension 64, 8 layers, and 4 heads. Transolver has dimension 64, 8 layers, 8 heads, and 32 slices. EqGINO (Kim et al., 2026) is the grid variant of the equivariant geometry-informed operator. On the uniform grid its point-cloud encoder and decoder reduce to the identity, and it retains the spectral layers of FNO with one weight shared by all Fourier modes of equal magnitude and a full Fourier transform, at width 32, 12 modes, and 4 layers. The vector fields are treated as scalar channels. HNO (Pei et al., 2026) follows the patch configuration of the reference, patches of 4 cells, dimension 128, 4 layers, and 4 heads, with attention weights from the hyperbolic distance between patches and a relative position bias. DeepONet is the classical branch and trunk architecture. The branch is the spectral encoder of FNO pooled to an 8×8 sensor grid and a code of 128, the trunk is four layers of width 192 on the coordinates, and the output at a cell is the inner product of the two. A local variant with an unpooled branch and a per-cell product of branch and trunk reaches a relative L2 of 0.081 at G1-1, but its rollouts diverge on 14 of the 194 G3 trajectories, so the classical form is the one reported. HPM is the holistic physics mixer (Yue et al., 2025d) with 5 layers, dimension 64, 8 heads, and 32 basis functions, and its geometry-adaptive basis is a sine basis restricted to the fluid mask in place of the Laplacian eigenbasis of the reference. The model sizes are 0.29 million parameters for Galerkin, 0.44 million for Transolver, 0.49 million for EqGINO, 0.57 million for HPM, 0.59 million for DilRes, 1.69 million for CNO, 2.24 million for F-FNO, 2.36 million for FNO, 4.96 million for U-NO, 5.20 million for HNO, 6.18 million for DeepONet, 6.54 million for UNet, and 6.67 million for UNet-c.

Training is one-step prediction with one protocol for every architecture. The loss is the relative L2 error of the predicted velocity relative to the reference, per sample and per channel on the fluid cells in the normalized space, averaged over the batch. The optimizer is AdamW with a learning rate of 10^{-3}, a weight decay of 10^{-4}, a cosine schedule over 100 epochs, and gradient clipping at norm 1. A batch contains 8 or 16 samples, and gradients are accumulated over one to four batches, so the effective batch is 16 or 32. The effective batch is 32 for DilRes, UNet-c, Transolver, and HPM and 16 for the remaining architectures. Every architecture trains once with seed 42, and the checkpoint with the lowest validation loss on the validation split is evaluated. There is no rollout training. The architecture does not receive its own prediction during training, and the rollouts of Sec. D.3 apply the one-step map to its own output.

Repeated training has a limited effect on the results. The local DeepONet variant of Sec. D.2 was trained twice under the same protocol. Its relative L2 differs between the two runs by 5 percent at G1-1 and by at most 1 percent at G1-2, at G2, and in the one-step error at G3. The difference between two runs is therefore much smaller than the difference between architectures at G1 and G2 and than the factor of 3 to 6 from G2 to G3.

The implementation fixes everything except the architecture. The design has three strengths, (a) the residual step, the normalization, and the masked loss are applied outside the architecture, so a difference in the results is attributable to the architecture, (b) the normalization with one statistic for the current and the next velocity gives an exact increment and a stable joint training across geometries, and (c) every architecture receives the same geometric and positional information through the same channels, so no architecture has private access to the geometry. It has three limits, (a) each architecture trains once with one learning rate, so the ranking within a level includes the variance of one seed and of an untuned learning rate, (b) CNO is a simplified re-implementation and UNet-c is U-Net with an added coordinate path, so their numbers are those of this implementation, and (c) position is given as raw coordinates, and an architecture that would benefit from a Fourier or learned positional encoding does not receive one. Three directions follow from the results of Sec. 4, architectures that receive the geometry as an object, such as a signed distance function or a boundary point cloud (Li et al., 2023c;d), training that includes the rollout, such as the pushforward step, and positional encodings that separate the position within the geometry from the position on the grid. The protocol delivers one checkpoint per architecture to the evaluation of Sec. D.3.

D.3 Metrics

Every metric is computed on the autoregressive rollout of a test trajectory, because a deployed surrogate receives its own predictions as input after the first step. Accuracy computed with reference inputs at every step overstates this condition. The rollout starts from the first snapshot of the trajectory and predicts every later snapshot. All metrics are computed on the fluid region of the geometry and on fields in physical units.

Field metrics quantify the accuracy of the predicted fields. Let Ω_f denote the fluid cells of the grid and c index the channels. The relative L2 error at rollout step k is

$$\varepsilon_k \;=\; \left(\frac{\sum_c \sum_{x \in \Omega_f} \big(\hat{u}_{k,c}(x) - u_{k,c}(x)\big)^2}{\sum_c \sum_{x \in \Omega_f} u_{k,c}(x)^2} \right)^{1/2}, \tag{12}$$

where $\hat{u}_k$ and u_k are the predicted and reference fields, with the two velocity components taken together, so the channels share one denominator. The reported relative L2 is the mean of ε_k over the rollout. The correlation time is the first step at which the predicted and the reference velocity decorrelate,

$$\tau \;=\; \min\{\, k \mid \rho_k < 0.8 \,\}, \tag{13}$$

where ρ_k is the Pearson correlation of $\hat{u}_k$ and u_k over Ω_f, averaged over the two velocity components. The threshold of 0.8 follows Lippe et al. (2023). A rollout that stays above the threshold to its end contributes its length, so τ is a lower bound for that trajectory. It reports how many steps of a rollout are usable.

Spectral metrics quantify fidelity at resolved wavenumbers. The velocity field is masked to Ω_f and transformed by a two-dimensional discrete Fourier transform, and the power of the two components is summed into one spectrum $S(\mathbf{k})$. The radial wavenumber $|\mathbf{k}|$ is divided into three equal bands up to $\min(n_y, n_x)/2$, and the energy of band B is $E_B = \sum_{\mathbf{k} \in B} S(\mathbf{k})$. The spectral deviation of a band is

$$\sigma_B \;=\; \frac{\hat{E}_B - E_B}{E_B}, \tag{14}$$

and the reported quantity is σ_B of the highest band. It reports whether a prediction retains the smallest resolved scales.

Physical metrics quantify consistency with incompressibility and with the energy budget. The divergence of a velocity field is computed by central differences on the fluid cells that have fluid neighbours, and its root mean square is made dimensionless by the local velocity gradient scale,

$$D(u) \;=\; \frac{\big\langle (\nabla \cdot u)^2 \big\rangle^{1/2}}{\big\langle |u|^2 \big\rangle^{1/2} / \Delta x}, \tag{15}$$

Table 3: Configuration of the evaluated architectures. Structure follows the reference implementation. Width and depth are set to one scale. Model sizes are those of the trained networks.

Architecture	Family	Reference	Structure, width, depth	Size
DilRes	convolutional	Stachenfeld et al., ICLR 2022	4 blocks of 7 dilated 3×3 convolutions, 48 channels	0.59M
UNet-c	convolutional	Ronneberger et al., MICCAI 2015	U-Net in the form of Gupta & Brandstetter (TMLR 2023), 3 scales, base 64, group normalization and GELU, linear-attention bottleneck, and a coordinate path with per-cell modulation at every scale	6.67M
UNet	convolutional	Ronneberger et al., MICCAI 2015	the same U-Net without the coordinate path, coordinates as input channels only	6.54M
CNO	convolutional	Raonić et al., NeurIPS 2023	3 scales, 32 channels, anti-aliased resampling	1.69M
FNO	spectral	Li et al., ICLR 2021	4 layers, width 32, 12 modes	2.36M
F-FNO	spectral	Tran et al., ICLR 2023	8 layers, width 64, 16 modes, factorized	2.24M
U-NO	spectral	Rahman et al., TMLR 2023	3 scales, width 32, modes scaled with resolution	4.96M
Galerkin	transformer	Cao, NeurIPS 2021	8 layers, dimension 64, 4 heads, Galerkin attention	0.29M
Transolver	transformer	Wu et al., ICML 2024	8 layers, dimension 64, 8 heads, 32 slices	0.44M
EqGINO	spectral	Kim et al., ICML 2026	4 layers, width 32, 12 modes, spectral weights shared over frequency orbits, full FFT (grid variant of the 3D architecture)	0.49M
HNO	transformer	Pei et al., ICML 2026	4 layers, dimension 128, 4 heads, patch 4, hyperbolic-distance attention with relative position bias	5.20M
DeepONet	operator net	Lu et al., Nat. Mach. Intell. 2021	branch = the spectral encoder of FNO (width 56, 10 modes, 4 layers) pooled to an 8×8 sensor grid and a code of 128, trunk = 4 layers of width 192 on the coordinates, and one inner product per cell	6.18M
HPM	spectral	Yue et al., ICML 2025	5 layers, dimension 64, 8 heads, 32 basis functions, with a sine basis on the fluid mask in place of the Laplacian eigenbasis of the reference	0.57M

where $\langle\cdot\rangle$ is the mean over those cells and Δx the grid spacing. The divergence excess is $D(\hat{u}) - D(u)$. The reference divergence on the uniform grid is nonzero after resampling, so the excess isolates the violation added by the prediction. The kinetic energy of a field is $E = \left\langle \frac{1}{2}|u|^2 \right\rangle$ over Ω_f, and the kinetic-energy deviation is

$$\delta_E \;=\; \frac{\hat{E} - E}{E}. \tag{16}$$

It reports the error of the energy budget of a prediction. The vorticity relative L2 is Eq. (12) applied to the out-of-plane vorticity $\omega = \partial_x u_y - \partial_y u_x$, computed by the same central differences.

Each metric is aggregated over snapshots, then over trajectories, then over families. The energy and spectral deviations are signed per snapshot, so each is averaged as an absolute value, because deviations of opposite sign cancel in a plain mean. The signed averages are

retained with the released results, because their sign separates dissipation from amplification. Across families the benchmark reports the arithmetic mean and the geometric mean. The error of a neural operator on a family is approximately the product of a family difficulty and an operator factor, and the geometric mean cancels the difficulty factor family by family. A comparison between two neural operators under the geometric mean is therefore independent of how difficult each family is, and every family contributes with equal weight. The arithmetic mean is dominated by the most difficult families. Fig. 7 and Tables 5 and 6 report the arithmetic mean over trajectories from one training seed.

Panel a of Fig. 7 is computed from the evaluation of every architecture on the four test splits. Each architecture is trained once with the protocol of Sec. D.2 and evaluated on the autoregressive rollout of every test trajectory, from the first snapshot to the last output time of the trajectory. Every metric is first averaged over the steps of a rollout, then arithmetically over the trajectories of a test split, so a family with more test trajectories has more weight. The dashed line of the correlation-time chart is the mean rollout length over the four test splits, 27.6 steps. The bar chart reports the relative L2 at G2 and at G3 of each architecture, sorted by the G3 value, and its title names the architecture with the lowest error at G2 and at G3. The four line charts have logarithmic axes, and the correlation-time axis starts at 10 steps.

Panel b of Fig. 7 is rendered from the same rollouts on nine test trajectories. The three G1-1 trajectories are one trajectory each of three training families. The three G2 trajectories are one new geometry of each of those families. The three G3 trajectories are one trajectory each of three test families with no family in training. Each cell shows the speed $|u|$ at the last output time on the fluid region, the reference from the simulation and every other row from the rollout of the architecture. The colour scale of a column runs from zero to the 99th percentile of the reference frame of that column, so the rows of a column share one scale. A portrait geometry is rotated by 90 degrees, and every column maintains the aspect ratio of its geometry within the range 0.30 to 0.55 by adding white margins. The number in a cell is Eq. 12 on the velocity of that single frame, so it differs from the relative L2 of panel a in two ways, one trajectory and one frame. The geometry of a column is drawn from the fluid area fraction of the uniform grid. The boundary is the 0.5 iso-line of the fraction, and each segment of the boundary takes the colour of the boundary type of the nearest boundary cell, grey for a wall, teal for an inlet, and green for an outlet.

Fig. 7 in detail. **a**, Rollout metrics of nine architectures on the four test splits (127, 51, 157, and 194 trajectories), as the arithmetic mean over the trajectories of each split from one training seed. Colour is the architecture family (green convolutional, blue spectral, orange transformer) and the marker the architecture, as in the legend. The top row shows, left to right, the relative L2 (Eq. 12, lower is better), the correlation time in steps (higher is better, with the mean rollout length of 28 steps as a dashed line), and the relative L2 at G2 (solid) and at G3 (light) of every architecture, sorted by the G3 value. The bottom row shows the root-mean-square divergence of the prediction divided by that of the reference (the dashed line at 1 is the reference), the kinetic-energy deviation, and the relative L2 of the vorticity, on logarithmic axes. Each title states the median over the architectures of the increase at G3, and the shaded column indicates G3 in every line chart. **b**, Speed at the last output time on nine test trajectories, three per level, one column per trajectory with its Reynolds number. The rows are, top to bottom, the geometry (solids grey, fluid pale blue, inlet teal, outlet green), the reference, and the last predicted frame of each architecture in the order of a. Colour is speed from zero (blue) to the 99th percentile of the reference frame (red), per column. Portrait geometries are rotated to landscape. The number in a cell is the relative L2 of the velocity of that frame. The G1-1 trajectories are the involute hydrocyclone (Re 1000, R_{vf} 0.32), the stepped spillway (Re 800, step 0.38), and the sinuous counter-flow channel (Re 300, A 20). The G2 trajectories are a new geometry of each of these families (Re 2000, R_{vf} 0.52, then Re 1600, step 0.22, then Re 600, A 0). The G3 trajectories are the conical hydrocyclone (Re 1600, cone 1.05), the Parshall flume (Re 1200, ht 0.45), and the meridional hydrocyclone H (Re 1000, h 1.8).

The metrics evaluate a neural operator under its deployment condition. The design has three strengths, (a) every reported number is computed on the autoregressive rollout, the condition a deployed surrogate faces, (b) the geometric mean gives comparisons between

neural operators that are independent of family difficulty, and (c) the spectral and physical metrics detect errors in the smallest resolved scales, in incompressibility, and in the energy budget that a low field error does not show. It has two limits, (a) the correlation time of a rollout that remains correlated to the end of its trajectory is reported as the trajectory length, a lower bound, and (b) one-step accuracy is reported only for the experiment of Sec. E.3, and the released results retain it for every architecture. The metrics deliver the numbers of Fig. 7 and Tables 5 and 6.

E Full results

Tables 5 and 6 report all five metrics of all 13 evaluated architectures on the four test splits, and Table 4 summarizes four of the metrics on one page. Sec. E.1 reports the results of Sec. 4 metric by metric and architecture by architecture.

E.1 The results of the nine architectures in detail

This section reports the nine architectures of Fig. 7 in detail, then UNet, the U-Net without the coordinate path (Sec. D.2), and DeepONet (Lu et al., 2021). Every number is computed on the autoregressive rollout of a test trajectory and averaged over the trajectories of the test split (Sec. D.3).

Relative L2 remains within a factor of 1.4 of its G1-1 value at G1-2 and G2 and rises by a factor of 3 to 6 from G2 to G3 (Fig. 7a, top left). At G1-1 UNet-c is lowest at 0.052, then F-FNO 0.066, HNO 0.067, U-NO 0.077, CNO 0.083, DilRes 0.084, FNO 0.109, Transolver 0.116, and Galerkin 0.131. At G1-2 eight of the nine architectures are at or below their G1-1 value, and UNet-c is lowest at 0.040, so the highest Reynolds number of a trained geometry costs no accuracy. At G2 every value remains below 0.14, with UNet-c 0.061, HNO 0.079, F-FNO 0.080, CNO 0.081, DilRes 0.089, U-NO 0.105, Transolver 0.128, and FNO and Galerkin 0.135. At G3 the values are 0.32 to 0.67, with HNO 0.324, F-FNO 0.338, CNO 0.361, DilRes 0.385, UNet-c 0.390, FNO 0.416, U-NO 0.425, Transolver 0.428, and Galerkin 0.669.

The order at G3 is resolved by a paired bootstrap over the 194 G3 trajectories, on which every architecture is evaluated. HNO is below every other architecture in at least 97 percent of 3,000 resamples, F-FNO is below the remaining seven in at least 96 percent, and CNO is below the remaining six in at least 77 percent. DilRes and UNet-c are not separated (57 percent), and UNet-c is below FNO, U-NO, and Transolver in at least 86 percent. The ratio of the G3 mean to the G2 mean has a 95 percent interval above 2.6 for every architecture. The intervals of the means themselves are wide, because the error is distributed with a heavy tail over the trajectories, with a standard deviation at G3 close to the mean, so the tables report the means only.

Correlation time and the physical metrics locate the loss at G3 as well (Fig. 7a, top right and bottom row, and Tables 5 and 6). At G1-1, G1-2, and G2 the correlation time is 22 to 29 steps of a rollout of 28 steps on average, and F-FNO at G1-2 remains correlated for the entire rollout. At G3 the correlation time falls to 16 to 24 steps, and HNO retains the longest, 23.8. The divergence excess increases from 0.003 to 0.019 at G1-1 to 0.020 to 0.077 at G3, a factor of 2 to 9 per architecture. The kinetic-energy deviation increases from 0.017 to 0.063 at G1-1 to 0.15 to 1.29 at G3, and the deviation of Galerkin exceeds the reference energy. The vorticity relative L2 increases from 0.18 to 0.42 at G1-1 to 0.74 to 1.53 at G3. The architecture with the lowest value at G3 differs per metric, F-FNO for kinetic energy (0.147), CNO for vorticity (0.74), and DilRes for divergence (0.020). Relative to the reference, the divergence of the predictions is 1.7 times that of the reference at G1-1 and 14 times at G3, as medians over the architectures (Fig. 7a, bottom left).

Speed at the last output time on nine test trajectories shows where those numbers appear in the field (Fig. 7b). The columns are three trajectories per level, the involute hydrocyclone, the stepped spillway, and the sinuous counter-flow channel at G1-1, a new geometry of each of the three families at G2, and the conical hydrocyclone, the Parshall flume, and the meridional hydrocyclone at G3. At G1-1 and G2 the convolutional and spectral architectures retain the reference structure, the vortex core of the involute, the jet along the spillway steps, and the counter-flow between the sinuous partitions. The relative L2 of these last frames

is 0.07 to 0.50 on the spillway and the sinuous channel, and 0.47 to 0.86 on the involute, whose core is the most demanding feature of the nine. Galerkin and Transolver weaken the spillway jet (0.35 to 0.57) and lose the involute core (1.0 to 1.2), whereas HNO, whose attention includes a local convolutional path, retains both (0.35 and 0.23 on the spillway, 0.84 and 0.95 on the involute). At G3 every architecture departs from the reference. The conical jet is displaced or split, the Parshall jet spreads over the basin, and the meridional inflow jet is smeared or misplaced, with a last-frame relative L2 of 0.49 to 1.34 in every cell.

The four lowest errors at G3 belong to architectures built on local operations, HNO with its convolutional path, F-FNO, CNO, and DilRes. The two architectures of the nine without a local path, Galerkin and Transolver, are the last two. A local feature such as a wall corner or a narrow gap appears in the same form in a new geometry, and the layout of a geometry does not. The benchmark does not test this explanation, so it remains a hypothesis.

The coordinate path of UNet-c lowers the relative L2 at every level and most on new families (Tables 5 and 6). UNet, the same network with the coordinates as input channels only, reaches 0.060, 0.050, 0.075, and 0.496 at G1-1, G1-2, G2, and G3, compared with 0.052, 0.040, 0.061, and 0.390 for UNet-c, a difference of 0.008 to 0.014 at G1-1, G1-2, and G2 and of 0.106 at G3. The correlation time of the two is within 0.3 steps at every level, and the divergence excess of UNet is 1.2 to 1.5 times that of UNet-c. At G3 the kinetic-energy deviation of UNet is 0.817 compared with 0.267, and the high-band deviation is the one metric where UNet is lower, 2.41 compared with 6.40. Position given a second time through the per-cell modulation is therefore worth most where the geometry is new.

DeepONet is the least accurate architecture at every level, 0.243 at G1-1 and 0.590 at G3, because a code of 128 numbers must contain the whole field.

E.2 Tables

Table 4: Relative L2, correlation time, divergence excess, and kinetic-energy deviation of all 13 architectures on the four test splits, as a summary of Tables 5 and 6. Appendix D.2 lists every reference.

	Relative L2 ↓				Correlation time τ ↑				Divergence, 10^{-3} ↓				Energy deviation ↓			
Architecture	G1-1	G1-2	G2	G3	G1-1	G1-2	G2	G3	G1-1	G1-2	G2	G3	G1-1	G1-2	G2	G3
DeepONet	0.243	0.237	0.340	0.590	17.3	17.4	13.7	11.1	8.5	6.5	12.4	**10.4**	0.102	0.134	0.157	0.429
FNO	0.109	0.108	0.135	0.416	25.6	25.9	24.9	18.6	14.4	16.8	14.3	53.5	0.030	0.027	0.039	0.168
Galerkin	0.131	0.112	0.135	0.669	22.3	21.6	23.6	16.2	18.7	15.9	17.9	76.7	0.063	0.067	0.086	1.289
DilRes	0.084	0.082	0.089	0.385	25.7	25.6	26.7	21.9	4.9	5.7	4.0	19.8	0.061	0.073	0.032	0.483
F-FNO	0.066	0.048	0.080	0.338	26.2	**28.9**	27.1	22.5	6.9	6.5	7.4	43.6	0.032	0.011	0.036	**0.147**
U-NO	0.077	0.084	0.105	0.425	**27.1**	27.4	26.7	18.6	8.7	12.8	10.4	52.6	0.022	0.027	0.030	0.184
UNet	0.060	0.050	0.075	0.496	26.9	28.1	26.8	20.7	3.7	3.6	4.2	35.3	0.020	**0.010**	**0.023**	0.817
UNet-c	**0.052**	**0.040**	**0.061**	0.390	**27.1**	28.2	26.8	20.8	**3.0**	**2.5**	**2.8**	27.0	**0.017**	0.012	0.024	0.267
CNO	0.083	0.068	0.081	0.361	25.5	27.1	26.7	21.9	6.0	6.2	5.4	30.1	0.027	0.017	0.024	0.171
Transolver	0.116	0.110	0.128	0.428	25.4	25.7	25.5	20.7	17.7	21.8	16.0	41.0	0.026	0.019	0.032	0.296
HPM	0.180	0.170	0.177	0.734	21.4	20.8	22.8	12.0	30.0	32.9	25.2	73.2	0.034	0.032	0.040	2.342
EqGINO	0.169	0.165	0.190	0.484	22.0	20.7	23.0	15.8	26.3	35.7	27.1	63.7	0.054	0.048	0.067	0.255
HNO	0.067	0.055	0.079	**0.324**	26.9	27.2	**27.5**	**23.8**	7.7	7.5	8.0	40.4	**0.017**	0.011	0.024	0.176

Table 5: Rollout metrics of all 13 architectures on the two G1 test splits. Metrics are grouped by field, spectral, and physical quantities (Sec. D.3). Lower is better, except correlation time τ (higher is better). Arithmetic mean over trajectories, one training seed. Bold is the best value of a block, underlined the second best. Fig. 7 reports the nine architectures of the three families.

	Field		Spectral	Physical	
Architecture	Relative L2	Correlation time τ	High-band deviation	Divergence excess	Kinetic-energy deviation
G1-1, a new Reynolds number					
DilRes	0.084	25.7	0.454	0.0049	0.061
UNet-c	**0.052**	**27.1**	<u>0.148</u>	**0.0030**	**0.017**
CNO	0.083	25.5	0.279	0.0060	0.027
FNO	0.109	25.6	0.285	0.0144	0.030
F-FNO	0.066	26.2	0.238	0.0069	0.032
U-NO	0.077	**27.1**	**0.124**	0.0087	0.022
HPM	0.180	21.4	1.65	0.0300	0.034
UNet	<u>0.060</u>	<u>26.9</u>	0.161	<u>0.0037</u>	<u>0.020</u>
Galerkin	0.131	22.3	2.30	0.0187	0.063
Transolver	0.116	25.4	0.600	0.0177	0.026
DeepONet	0.243	17.3	2.17	0.0085	0.102
EqGINO	0.169	22.0	0.542	0.0263	0.054
HNO	0.067	<u>26.9</u>	0.156	0.0077	**0.017**
G1-2, extrapolation in Re					
DilRes	0.082	25.6	0.556	0.0057	0.073
UNet-c	**0.040**	<u>28.2</u>	<u>0.079</u>	**0.0025**	0.012
CNO	0.068	27.1	0.171	0.0062	0.017
FNO	0.108	25.9	0.405	0.0168	0.027
F-FNO	<u>0.048</u>	**28.9**	**0.076**	0.0065	<u>0.011</u>
U-NO	0.084	27.4	0.174	0.0128	0.027
HPM	0.170	20.8	1.44	0.0329	0.032
UNet	0.050	28.1	0.180	<u>0.0036</u>	**0.010**
Galerkin	0.112	21.6	1.75	0.0159	0.067
Transolver	0.110	25.7	0.531	0.0218	0.019
DeepONet	0.237	17.4	2.53	0.0065	0.134
EqGINO	0.165	20.7	0.590	0.0357	0.048
HNO	0.055	27.2	0.118	0.0075	<u>0.011</u>

Table 6: Rollout metrics of all 13 architectures on the G2 and G3 test splits, as Table 5.

Architecture	Field		Spectral	Physical	
	Relative L2	Correlation time τ	High-band deviation	Divergence excess	Kinetic-energy deviation
G2, new geometries of trained families					
DilRes	0.089	26.7	0.151	0.0040	0.032
UNet-c	**0.061**	26.8	**0.118**	**0.0028**	0.024
CNO	0.081	26.7	0.189	0.0054	0.024
FNO	0.135	24.9	0.385	0.0143	0.039
F-FNO	0.080	27.1	0.198	0.0074	0.036
U-NO	0.105	26.7	0.213	0.0104	0.030
HPM	0.177	22.8	1.28	0.0252	0.040
UNet	0.075	26.8	0.137	0.0042	**0.023**
Galerkin	0.135	23.6	2.16	0.0179	0.086
Transolver	0.128	25.5	0.611	0.0160	0.032
DeepONet	0.340	13.7	2.35	0.0124	0.157
EqGINO	0.190	23.0	0.916	0.0271	0.067
HNO	0.079	**27.5**	0.156	0.0080	0.024
G3, new families					
DilRes	0.385	21.9	1.20	0.0198	0.483
UNet-c	0.390	20.8	6.40	0.0270	0.267
CNO	0.361	21.9	**0.383**	0.0301	0.171
FNO	0.416	18.6	0.676	0.0535	0.168
F-FNO	0.338	22.5	0.520	0.0436	**0.147**
U-NO	0.425	18.6	0.514	0.0526	0.184
HPM	0.734	12.0	3.86	0.0732	2.342
UNet	0.496	20.7	2.41	0.0353	0.817
Galerkin	0.669	16.2	9.78	0.0767	1.289
Transolver	0.428	20.7	0.879	0.0410	0.296
DeepONet	0.590	11.1	1.56	**0.0104**	0.429
EqGINO	0.484	15.8	1.41	0.0637	0.255
HNO	**0.324**	**23.8**	0.862	0.0404	0.176

E.3 Approximate power laws on geometric diversity

The experiment of Sec. 4.4 separates the number of training families from the number of trajectories per family. UNet-c is trained on K families with d trajectories each, with K of 32, 64, and 128 and d of 4, 8, 16, and the whole family, which contains 28 to 38 trajectories. HNO and F-FNO are trained on the same families with the whole family. Every cell of the grid receives the same budget of 200,000 optimizer steps, so a cell with more trajectories gives each trajectory fewer updates. Sixteen probe families are present in the training split of every cell.

Every cell is evaluated on the same trajectories, the 194 G3 trajectories, the 32 probe trajectories of the sixteen probe families, and the new geometries of those families. The reported error is the one-step relative L2 as a geometric mean over families. The geometric mean is used because the arithmetic mean of a cell is set by its few diverging trajectories, and the one-step error for the same reason, unlike the rollout errors of Table 4. F-FNO was trained twice at 32 families with batch sizes of 16 and 8. Its one-step G3 error differs by 3 percent between the two runs, much smaller than the 29 percent decrease from 32 to 128 families.

The other metrics at G3 follow the same law. The interior, wall, vorticity, and spectral errors decrease with exponents of 0.33 to 0.37 for UNet-c, 0.29 to 0.42 for HNO, and 0.20 to 0.34 for F-FNO. The divergence excess does not change with the number of families.

Adding families at a fixed budget costs accuracy on the trained families only through the updates each of them receives. On the probe families the error of UNet-c rises with K, from 0.0033 to 0.0039 at 4 trajectories per family, and at G2 and at G1-2 it does not change with K. The probe families are $16/K$ of the training split, so their trajectories receive 3.8 times fewer updates at $K = 128$ than at $K = 32$. A run on the probe families alone, shortened to the same number of updates per trajectory, lies within 15 percent of the runs on 32, 64, and 128 families, 0.0030 compared with 0.0033, 0.0034 compared with 0.0038, and 0.0045 compared with 0.0039.

Fig. 8 plots the G3 error as a function of K on logarithmic axes for UNet-c, HNO, and F-FNO trained on the whole family. Each line is the least-squares fit in log-log coordinates through the three cells of its architecture, with its equation in the lower left, and it is extrapolated to $K = 206$. The exponents are 0.38, 0.30, and 0.24 for UNet-c, HNO, and F-FNO, which correspond to 23, 19, and 16 percent per doubling of the number of families. The points at $K = 206$ are the runs of Table 4 on all 206 training families under the training protocol of the benchmark, drawn with the marker of their architecture, and they are not part of the fit. Their errors are within 1 percent of the extrapolated line for UNet-c and F-FNO and within 5 percent for HNO. Fig. 10 plots the four rows of UNet-c in the same way, from 4 trajectories per family to the whole family. The four lines fall together. At $K = 32$ the rows lie between 0.045 and 0.047, at $K = 128$ between 0.026 and 0.030, and their exponents are 0.31 to 0.38. At a fixed number of 512 training trajectories the error still decreases with the number of families, from 0.046 at 32 families with 16 trajectories each to 0.030 at 128 families with 4 trajectories each. The number of trajectories per family counts within a trained family. At G2 the error is 0.006 to 0.007 with 4 trajectories per family and 0.004 to 0.005 with the whole family, at every K. The number of families therefore sets the error on new families, and the number of trajectories per family does not.

F Related work in detail

This appendix expands Sec. 5. Each subsection reviews one line of prior work and ends with the property of KaleidoFlow that the prior work lacks. Sec. F.1 compares the existing flow datasets with KaleidoFlow, Sec. F.2 describes how neural operators receive the geometry, Sec. F.3 describes the scaling laws reported for neural operators, Sec. F.4 describes the language model agents that set up simulations and generate tasks, and Sec. F.5 describes the verification of the discretization error.

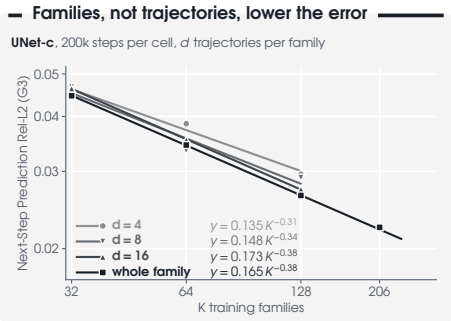

Figure 10: **One-step G3 error of UNet-c as a function of the number of training families, by trajectories per family.** One line per number of trajectories per family d, from 4 to the whole family of 28 to 38 trajectories, each the least-squares fit in log-log coordinates through its three cells, with its equation in the lower left. The whole-family line is extrapolated to the run of Table 4 on all 206 training families, which is not part of the fit.

F.1 Flow datasets

No existing flow dataset changes the geometry from family to family, verifies the discretization error of every geometry, and tests on new geometry families. Existing datasets fall into three groups by how they vary the geometry, and Table 7 compares nine of them with KaleidoFlow. The first group fixes the geometry. The second group varies the obstacle within one family. The third group provides several fixed geometries. The paragraphs below describe the three groups and then the three properties that separate KaleidoFlow from all of them.

The first group fixes the geometry and varies the initial field or the physics. PDEBench (Takamoto et al., 2022) provides eight PDE systems on boxes in one to three dimensions, with 100 to 10,000 trajectories per system and a random split. PDEArena (Gupta & Brandstetter, 2023) provides two-dimensional Navier–Stokes and shallow-water flows on a closed box. The Well (Ohana et al., 2024) collects 16 datasets of 15 TB, each on one geometry. APEBench (Koehler et al., 2024), FD-Bench (Wang et al., 2025a), and CFDONEval (Liu et al., 2025a) evaluate many architectures on fixed periodic boxes or on a fixed set of canonical problems, and the pretraining set of Poseidon (Herde et al., 2024) consists of Euler and Navier–Stokes trajectories on the periodic unit square. BubbleML (Hassan et al., 2023), LagrangeBench (Toshev et al., 2023), BLASTNet (Chung et al., 2023), the Johns Hopkins turbulence database (Li et al., 2008), and the turbulence data of Kohl et al. (2026) provide boiling, particle, combustion, and turbulent flows in fixed boxes. These datasets test generalization in the initial condition and in the physical parameters, and the geometry is not a variable.

The second group varies the obstacle within one family. DeepCFD (Ribeiro et al., 2020) contains 981 obstacles built from five primitives in one channel, and the earlier convolutional surrogates of Guo et al. (2016) and Bhatnagar et al. (2019) were trained on primitive shapes and on airfoils in the same way. Airfoil datasets are the largest family. Thuerey et al. (2020) simulate 1,500 UIUC airfoils, AirfRANS (Bonnet et al., 2022) contains 1,000 airfoils of the NACA four- and five-digit series, UniFoil (Kanchi et al., 2025) extends the family to transitional and turbulent regimes, and AFBench (Liu et al., 2024b) contains 200,000 airfoils for inverse design. Car bodies are the second largest. ShapeNet-Car (Umetani & Bickel, 2018), DrivAerNet (Elrefaie et al., 2025a) and DrivAerNet++ (Elrefaie et al., 2024), DrivAerML (Ashton et al., 2024c), AhmedML (Ashton et al., 2024b), WindsorML (Ashton et al., 2024a), and DrivAerStar (Qiu et al., 2025) contain 355 to 12,000 variants of one car or bluff body in three dimensions, and CarBench (Elrefaie et al., 2025b) adds a test on unseen car archetypes to DrivAerNet++. Ship hulls (Bagazinski & Ahmed, 2023), piano-key weirs (Lüddecke et al., 2026), and the industrial parts of Zhong et al. (2025) are further single families. The datasets released with MeshGraphNets (Pfaff et al., 2021), Geo-FNO (Li et al., 2023c), and GINO (Li et al., 2023d) vary a cylinder, an airfoil, a pipe, or an Ahmed body in the same way. Random two-dimensional shapes drawn from one generator, Bézier obstacles at low Reynolds number (Viquerat & Hachem, 2020; Chen et al., 2021), randomized cylinders (Michel et al., 2026), random building layouts (Lee et al., 2026), and

Table 7: Existing flow datasets for neural operators and KaleidoFlow. Every row is taken from the paper or the repository of the dataset. Trajectories are counted as the dataset counts its samples, one trajectory per initial condition for the fixed geometries and one steady solution per obstacle for the obstacle families.

Dataset	Dim.	Distinct geometries	Geometry families	Trajectories	Error verified	Test on unseen geometries
PDEBench (Takamoto et al., 2022)	1–3D	1, a box	8 PDE systems	100 to 10,000 per system	no	no, random split
PDEArena (Gupta & Brandstetter, 2023)	2D	1, a closed box	2 PDE systems	5,200 to 5,600 per system	no	no
The Well (Ohana et al., 2024)	2D, 3D	1 per dataset	16 datasets	8 to 10,000 per dataset	no	no, random split
CFDBench (Luo et al., 2023)	2D	120 sizes of 4 geometries	4 problems	739	stated, no numbers	new sizes, same geometries
DeepCFD (Ribeiro et al., 2020)	2D	981 obstacles, 5 primitives	1 problem	981	no	new obstacles, same primitives
AirfRANS (Bonnet et al., 2022)	2D	1,000 airfoils	NACA 4 and 5 digit	1,000	against experiment	new airfoils, same family
EAGLE (Janny et al., 2023)	2D	600 scenes, 3 types	1 problem	1,184	no	one scene type unseen
DrivAerNet++ (Elrefaie et al., 2024)	3D	8,000 cars	DrivAer	8,000	three meshes, no index	no
FlowBench (Tali et al., 2025)	2D, 3D	3 shape families	3 problems	10,650	against literature	no, random or by Re
KaleidoFlow	2D	3,948	459 families, 10 categories	16,512	index per family, released	G2 new geometry, G3 new family

the parametric channels of MegaFlow2D (Xu et al., 2023), are the closest to KaleidoFlow in spirit. Every geometry of these datasets has the same outer boundary, the channel, the far field, or the wind tunnel, and only the obstacle inside it changes. A test geometry is therefore a new member of a trained family.

The third group provides several fixed geometries. CFDBench (Luo et al., 2023) provides four canonical problems, each at up to 120 sizes, and tests on new sizes of the same geometries. EAGLE (Janny et al., 2023) provides 600 scenes of three types, in which a drone flies through a hall, and tests on one of the three types. FlowBench (Tali et al., 2025) provides three shape families in two and three dimensions with 10,650 samples, split at random or by Reynolds number, and Rabeh et al. (2025) train on one of its shape classes and test on another. ChannelFlow-Tools (Kavane et al., 2025) generates 450 obstructed channel flows from six obstacle families with a test on an unseen family. REALM (Mao et al., 2025) provides reactive multiphysics flows on a fixed set of scenarios in two and three dimensions. In every dataset of this group the set of geometries is fixed and enumerable, and a test geometry shares its outer boundary or its family with the training set, except for the single unseen type or family of EAGLE, FlowBench, and ChannelFlow-Tools.

Three properties separate KaleidoFlow from all three groups. (a) The geometry changes from family to family, in the shape of the channel, the number of obstacles, and the size of the geometry, over 459 families and 3,948 distinct geometries, whereas the largest existing datasets vary one obstacle or draw from one shape generator. (b) The discretization error

of every family is verified by a mesh convergence test and released with the family, whereas CFDBench states grid independence without numbers, AirfRANS and DrivAerStar validate their solver against experimental data on one family, and DrivAerNet++ compares three meshes on one car without a convergence index. (c) The G3 test split contains 37 new geometry families at a controlled distance in shape space, whereas EAGLE, FlowBench, ChannelFlow-Tools, and CarBench test on one unseen type or class and the other datasets split at random, by size, or by Reynolds number. Therefore, generalization across geometry families with a known discretization error can be evaluated on KaleidoFlow and on none of the datasets of Table 7.

F.2 How neural operators receive the geometry

Neural operators receive the geometry in five ways, and no design has been evaluated across geometry families or compared with another design under one interface. The five ways are (a) fields on a regular grid with the geometry as a mask or distance channel, (b) a mesh or graph, (c) a point cloud attended by tokens, (d) coordinates queried by a neural field, and (e) a geometry-native construction that maps the geometry to a reference or decomposes it. Conditioning mechanisms that inject position or parameters cut across the five ways. The paragraphs below describe each way and then the architectures the benchmark evaluates.

Grid-based architectures take the fields on a uniform grid, and the geometry is given as a fluid mask, a signed distance, and boundary channels. The first convolutional surrogates did so with a distance channel on primitive shapes and airfoils (Guo et al., 2016; Bhatnagar et al., 2019; Thuerey et al., 2020; Ribeiro et al., 2020). FNO (Li et al., 2021) with its factorized and multiscale variants F-FNO (Tran et al., 2023) and U-NO (Rahman et al., 2023), the convolutional U-Net (Ronneberger et al., 2015; Gupta & Brandstetter, 2023), DilRes (Stachenfeld et al., 2022), and CNO (Raonic et al., 2023), the Clifford layers of Brandstetter et al. (2023), and the grid transformers Galerkin (Cao, 2021), FactFormer (Li et al., 2023b), and PDE-Transformer (Holzschuh et al., 2025) take the same input, and the spectral-physical HPM (Yue et al., 2025d) and the hyperbolic HNO (Pei et al., 2026) are the most recent members. PDE-Refiner (Lippe et al., 2023) refines the rollout of such an architecture, DeltaPhi (Yue et al., 2025c) replaces the direct map of any of them by the residual between a state and a similar reference state, and Rabeh et al. (2025) report that the choice between a mask and a distance channel depends on the architecture. Mesh-based architectures take the simulation mesh or the particles as a graph and pass messages along its edges. GNS (Sanchez-Gonzalez et al., 2020), MeshGraphNets (Pfaff et al., 2021), MP-PDE (Brandstetter et al., 2022), X-MeshGraphNet (Nabian et al., 2024), and RIGNO (Mousavi et al., 2025) give the geometry as node types and edge vectors.

Point-cloud architectures take the mesh points as an unordered set and attend to them through a fixed number of latent tokens. PointNet (Kashefi et al., 2021) regresses the fields from the coordinates. GNOT (Hao et al., 2023), OFormer (Li et al., 2023a), LSM (Wu et al., 2023), and LNO (Wang & Wang, 2024) cross-attend between the points and a latent set, Transolver (Wu et al., 2024) slices the points into physical states, and Transolver++ (Luo et al., 2025), Transolver-3 (Zhou et al., 2026), GeoTransolver (Adams et al., 2025), GFocal (Fei et al., 2025), and LA2Former (Koh et al., 2025) scale or localize the slicing. UPT (Alkin et al., 2024) and AB-UPT (Alkin et al., 2025) pool the points into supernodes, GAOT (Wen et al., 2025) and GINOT (Liu et al., 2025b) embed the geometry before the transformer, DoMINO (Ranade et al., 2025) gathers the local geometry by ball queries, Erwin (Zhdanov et al., 2025) attends along a ball tree, and PCNO (Zeng et al., 2025) corrects the kernel for the point density. Coordinate-based architectures decode the solution at any query point. DeepONet (Lu et al., 2021) takes the coordinates through a trunk network, DINo (Yin et al., 2023), CORAL (Serrano et al., 2023), and AROMA (Serrano et al., 2024) evolve a latent that a neural field decodes, and CViT (Wang et al., 2025b) queries a vision transformer by cross-attention.

Geometry-native architectures map the geometry to a reference or decompose it. Geo-FNO (Li et al., 2023c) learns a deformation from the mesh to a computational grid on which the Fourier layers act, and the PolyCube map of Umetani & Bickel (2018) does so for cars. GINO (Li et al., 2023d) takes a point cloud through graph layers around an FNO, EqGINO (Kim et al., 2026) adds equivariance to this design, and GANO (Sun et al.,

2026) applies a geometry-aware operator to shape optimization and inversion. NUNO (Liu et al., 2023) splits non-uniform data into subdomains, RNO (Cheng et al., 2024) predicts the solution as a deformation of a reference pair, and DIMON (Yin et al., 2024) learns a diffeomorphism from each geometry to a reference geometry. The most recent works target unseen geometries directly. SNI (Huang et al., 2026) trains a local operator on random polygons and stitches an arbitrary geometry by domain decomposition, Han et al. (2026) derive the geometric dependence from a kernel integral, Geo-NeW (Shaffer et al., 2026) learns a structure-preserving discrete operator on the given mesh, DiSOL (Bai et al., 2026) learns the assembly and solve procedure, the Laplace neural operator (Tang et al., 2025) uses the eigenbasis of each mesh, and Xia & Aviles-Rivero (2026) report that spectral and attention operators lose the geometry with depth. These works evaluate on Poisson and elasticity problems or on one family of shapes.

Conditioning mechanisms add the position, the parameters, or the boundary conditions to any of these architectures. CoordConv (Liu et al., 2018), FiLM (Perez et al., 2018), SPADE (Park et al., 2019), and ControlNet (Zhang et al., 2023) come from vision, CAPE (Takamoto et al., 2023) conditions on the PDE parameters, BOON (Saad et al., 2023) enforces the boundary condition in the operator kernel, Unisolver (Zhou et al., 2025) conditions a transformer on the complete set of PDE components, and PDE-Transformer (Holzschuh et al., 2025) conditions on the PDE class by adaptive layer normalization. UNet-c uses a coordinate path of this type (Sec. D.2).

Every one of these architectures has been reported on one geometry or one family. The grid, mesh, point-cloud, and coordinate designs report on airfoils, cars, or a fixed set of shapes, Transolver++, Transolver-3, and AB-UPT report on million-cell car and aircraft meshes of one family, and the works that target unseen geometries report on Poisson and elasticity problems. Each paper uses its own input encoding and training protocol, so the results of two papers are not comparable. The benchmark evaluates the grid-based architectures and DeepONet and EqGINO under one interface and one protocol (Sec. 4.2) at three distances from the training geometries. Therefore, a difference in its results is a difference of architecture, and the ladder locates the level at which each architecture fails, which is the family level for every one of them (Sec. 4.3).

F.3 Scaling laws for neural operators

Power laws of neural operators have been reported in the number of training samples, and the law of Sec. 4.4 is the first in the number of geometry families. Scaling laws relate the error of a model to the size of its training data. Hestness et al. (2017) and Kaplan et al. (2020) report power laws for language models in data, parameters, and compute, Hoffmann et al. (2022) balance data and parameters at a fixed compute budget, and Bahri et al. (2024) relate the exponent to the intrinsic dimension of the data. Sorscher et al. (2022) show that the power law in the number of samples is set by redundant examples, so the coverage of a dataset counts more than its size. For neural operators, Lanthaler et al. (2022) and Liu et al. (2024a) derive algebraic rates in the sample size and the model size, Raonic et al. (2023) fit power laws in the number of training samples for several architectures on one geometry, Subramanian et al. (2023) characterize the error as a function of the number of training samples and the model size on several PDE systems and study the transfer between the systems, and Goswami et al. (2022) transfer DeepONets to new geometries under conditional shift. MPP (McCabe et al., 2024), UPT (Alkin et al., 2024), UPS (Shen et al., 2024), OmniArch (Chen et al., 2025), DPOT (Hao et al., 2024), Poseidon (Herde et al., 2024), Walrus (McCabe et al., 2026), and the Earth-system model Aurora (Bodnar et al., 2025) pretrain one operator on many PDE datasets and report the accuracy as a function of the number of pretraining samples or datasets, and ICON (Yang et al., 2023) reports that in-context operator learning improves with the diversity of the operators in the training set. Setinek et al. (2025) decompose the data axis of a CFD surrogate into the compute budget and the mix of fidelities and report the compute-performance behaviour without an exponent. In all of these datasets the samples of one system share one geometry, so the number of samples and the geometric diversity increase together or not at all.

Other fields have separated the diversity of the training data from its quantity. In robot manipulation, Lin et al. (2025) report a power law in the number of training environments

and objects, whereas additional demonstrations per environment saturate, and Shi et al. (2025) find that task diversity counts more than demonstrations per task. In instruction tuning, Bukharin et al. (2024) find that the diversity of the instructions improves robustness at a fixed size. The benchmark separates the two axes for neural operators, the number of geometry families and the number of trajectories per family, which no dataset of Sec. F.1 can vary independently because each contains one family. Therefore, the power law in the number of families of Sec. 4.4 is the geometric counterpart of the environment scaling law of Lin et al. (2025), and it is observable on KaleidoFlow because the dataset contains hundreds of families. To our knowledge, it is the first power law observed for the generalization of neural operators across geometries, and a convolutional, a transformer, and a spectral architecture follow it.

F.4 Language model agents for simulation and task generation

In existing agentic CFD frameworks, the user specifies the geometry of a flow simulation and the language model configures the solver. The language model does not propose the geometry, and the output is not verified by a mesh convergence test. Four lines of work come close, agents that set up simulations, language models that generate geometry as code, autonomous discovery loops, and data engines that generate tasks. Language model agents have begun to automate the set-up of flow simulations, and in all of them the user specifies the geometry of each simulation. MetaOpenFOAM (Chen et al., 2024) decomposes a request in natural language into the files of an OpenFOAM simulation with several cooperating agents and runs it. OpenFOAMGPT (Pandey et al., 2025) retrieves solver documentation and tutorials to configure a simulation and corrects it from the solver output. Foam-Agent (Yue et al., 2025a) plans the simulation, retrieves similar tutorials, and corrects it from the solver log in a loop. FoamGPT (Yue et al., 2025b) and Dong et al. (2025) fine-tune a language model on OpenFOAM tutorials, FoamPilot (Xu et al., 2024b) and FlamePilot (Xiao et al., 2026a) apply the same loop to fire and combustion solvers, Xiao et al. (2026b) assess general coding agents on OpenFOAM tasks, and SIGA (Ho et al., 2026) adapts a coding agent to arbitrary simulators. MechAgents (Ni & Buehler, 2024), FeaGPT (Qi et al., 2025), and VFEAgent (Zhang et al., 2026) do the same for finite element analysis, FeaGPT with a Gmsh mesh generated by the agent. CodePDE (Li et al., 2026) and Wu et al. (2025) generate the solver code, and CFDLLMBench (Somasekharan et al., 2025), FEABench (Mudur et al., 2025), and PDEAgent-Bench (Hang et al., 2026) benchmark these abilities. The result of every one of these agents is one simulation of a geometry that the user provided.

Language models also generate geometry and meshes as code, and the outputs are checked for shape or for a downstream analysis. Shafiq et al. (2025) evaluate nine language models on Gmsh geometry scripts. Text2CAD (Khan et al., 2024), Text-to-CadQuery (Xie & Ju, 2025), CAD-Coder (Guan et al., 2025), CADCodeVerify (Alrashedy et al., 2025), CAD-Recode (Rukhovich et al., 2025), and LLM4CAD (Li et al., 2025) generate parametric CAD code from text, images, or point clouds and verify the executed geometry by its shape, and Son et al. (2026) verify the assemblies with a finite element analysis in a loop. The agent of the KaleidoFlow engine generates the geometry and the mesh strategy as code in the same way, and the check is a mesh convergence test of the flow solution.

Autonomous discovery loops pair a language model that proposes with an external check that verifies. Coscientist (Boiko et al., 2023) and ChemCrow (Bran et al., 2024) plan and execute chemistry with robotic platforms, FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Novikov et al., 2025) evolve programs that a systematic evaluator scores, LLM-SR (Shojaee et al., 2025) evolves equations fitted to data, and the AI Scientist (Lu et al., 2024), Agent Laboratory (Schmidgall et al., 2025), and Co-Scientist (Gottweis et al., 2026) run loops of hypothesis, experiment, and report, as surveyed by Wei et al. (2025). In FunSearch and AlphaEvolve every candidate is retained only if the evaluator accepts it, and the engine applies the same rule with the mesh convergence test as the evaluator.

Data engines and task generators enlarge a dataset with a model in the loop. Segment Anything (Kirillov et al., 2023) alternates between model-assisted annotation and training, and Self-Instruct (Wang et al., 2023), WizardLM (Xu et al., 2024a), and Magicoder (Wei et al., 2024) generate instruction and code data from seeds. In robotics, RoboGen (Wang

et al., 2024b), GenSim (Wang et al., 2024a), and GenSim2 (Hua et al., 2024) generate simulation tasks and scenes as code, RoboTwin (Mu et al., 2025), RoboCasa365 (Nasiriany et al., 2026), and SimFoundry (Ranawaka et al., 2026) generate scenes and their variations as digital twins, Eureka (Ma et al., 2024) generates reward code, Eurekaverse (Liang et al., 2024) generates terrain code as a curriculum, and Genie (Bruce et al., 2024) learns an environment generator from video. Open-ended learning generates environments and their solutions together. POET (Wang et al., 2019) co-evolves terrains and agents, OMNI (Zhang et al., 2024) selects learnable and interesting tasks with a foundation model, and OMNI-EPIC (Faldor et al., 2025) generates the environment code of the next task from an archive of earlier tasks. The KaleidoFlow engine is a generator of this type for flow data. The agent proposes the next geometry from the existing geometries, generates and successfully repairs its mesh strategy, and every family passes the mesh convergence test before it is added to the dataset. Therefore, the engine differs from the four lines in three respects. The agent proposes the geometries, the check is a physical verification of the flow solution, and the loop runs without human intervention.

F.5 Verification of the discretization error

Mesh convergence verification is the standard of computational fluid dynamics for reporting the discretization error, and no flow dataset for neural operators reports it for every geometry. Richardson extrapolation (Richardson, 1911; Richardson & Gaunt, 1927) estimates the exact solution from solutions on two meshes of known order. The grid convergence index of Roache (1994; 1997; 1998) converts the estimate into an error band for a mesh from three meshes of the same design and their observed order of convergence. Stern et al. (2001), Oberkampf & Trucano (2002), and Roy (2005) define the hierarchy of code verification, solution verification, and validation, Celik et al. (2008) prescribe the reporting procedure that the engine follows, Eça & Hoekstra (2014) extend the index to a least-squares fit over more than three meshes, and Phillips & Roy (2014) compare the estimators on a suite of test problems. The NASA Turbulence Modeling Resource (Diskin et al., 2015) applies the procedure to reference problems on families of nested meshes.

Automated meshing produces the meshes that the test compares, and learned meshing predicts a sizing field. Gmsh (Geuzaine & Remacle, 2009) builds unstructured meshes from a scripted sizing field, and the mesh strategies of the engine are scripts on its API. Metric-based adaptation (Alauzet & Loseille, 2016) derives the sizing from the solution in a loop, and MeshingNet (Zhang et al., 2020), Fidkowski & Chen (2021), and AMBER (Freymuth et al., 2025) learn the sizing field from error estimates or from expert meshes. The engine differs in that the agent specifies the sizing rule in terms of the geometry, and the rule is verified by the convergence test rather than by a learned error estimate. Among the datasets of Table 7, CFDBench states that its meshes are grid independent without numbers, AirfRANS validates its solver settings against experimental data on the airfoil family, DrivAerStar (Qiu et al., 2025) and BubbleML (Hassan et al., 2023) validate against wind-tunnel and boiling experiments, and DrivAerNet++ compares three meshes on one car without an index. The engine applies the procedure to every family it produces, on its hardest trajectory, and releases the index with the family (Sec. B.1). Therefore, the dataset generated by KaleidoFlow is the only one in Table 7 whose every family has a grid convergence index.