Iterative Sizing Field Prediction for Adaptive Mesh Generation From Expert Demonstrations

Niklas Freymuth [1] Philipp Dahlinger [1] Tobias Würth [2] Philipp Becker [1] Aleksandar Taranovic [1]
Onno Grönheim [3] Luise Kärger [2] Gerhard Neumann [1]

Abstract

Many engineering systems require accurate simulations of complex physical systems. Yet, analytical solutions are only available for simple problems, necessitating numerical approximations such as the Finite Element Method (FEM). The cost and accuracy of the FEM scale with the resolution of the underlying computational mesh. To balance computational speed and accuracy meshes with adaptive resolution are used, allocating more resources to critical parts of the geometry. Currently, practitioners often resort to hand-crafted meshes, which require extensive expert knowledge and are thus costly to obtain. Our approach, Adaptive Meshing By Expert Reconstruction (AMBER), views mesh generation as an imitation learning problem. AMBER combines a graph neural network with an online data acquisition scheme to predict the projected sizing field of an expert mesh on a given intermediate mesh, creating a more accurate subsequent mesh. This iterative process ensures efficient and accurate imitation of expert mesh resolutions on arbitrary new geometries during inference. We experimentally validate AMBER on heuristic 2D meshes and 3D meshes provided by a human expert, closely matching the provided demonstrations and outperforming a single-step CNN baseline.

1. Introduction

Simulating complex physical systems is an integral part of most engineering disciplines. These simulations build on fundamental laws of physics, such as conservation of mass and energy, and are often expressed through Partial Differential Equations (PDEs). Due to their complexity, these PDEs are analytically intractable, requiring numerical approximations such as the Finite Element Method (FEM) (Brenner & Scott, 2008; Reddy, 2019; Anderson et al., 2021). In the FEM, a continuous problem geometry is partitioned into a mesh, i.e., a set of finite and simple elements. Such a mesh allows efficient approximations of the PDE solution, with cost and accuracy scaling with the number of elements.

For problems of realistic complexity, naively constructed meshes are often insufficient and adaptive meshes are required. Those can have varying element sizes allowing for a finer resolution in regions of interest and coarser resolution in areas that are simpler to simulate. To find appropriate meshes Adaptive Mesh Refinement (AMR) methods can refine existing meshes by allocating elements based on heuristics. Similarly, Adaptive Mesh Generation (AMG) creates meshes based on the problem geometry and initial conditions, often omitting the need for intermediate solutions. Yet, both approaches are limited in efficiency and adaptability (Mukherjee, 1996; Kita & Kamiya, 2001; Yano & Darmofal, 2012; Cerveny et al., 2019; Wallwork, 2021) and the required heuristics are often costly (Giles & Pierce, 2000; Giannakoglou & Papadimitriou, 2008). Due to these challenges of existing tools, generating meshes for real-world applications still requires tedious manual labor and extensive domain knowledge.

Several recent approaches have explored AMR with machine learning. One popular paradigm uses Reinforcement Learning (RL) (Sutton & Barto, 2018) to frame AMR as a sequential decision-making process (Yang et al., 2023a; Foucart et al., 2023; Freymuth et al., 2023). While promising, these RL approaches rely on application-specific and complex reward functions (Foucart et al., 2023) or expensive, fine-grained uniform reference meshes (Freymuth et al., 2023). These factors, together with other constraints, such as a maximum refinement depth of the reference meshes (Freymuth et al., 2023) limit the practical applicability of these approaches. Alternatively, recent supervised approaches aim to learn from expert meshes, using standard Convolutional Neural Networks (CNNs) (Huang et al., 2021) or Multilayer Perceptrons (MLPs) (Zhang et al., 2020;

[1]Autonomous Learning Robots, Karlsruhe Institute of Technology, Karlsruhe, Germany [2]Institute of Vehicle Systems Technology, Karlsruhe Institute of Technology, Karlsruhe [3]EVAGO GmbH, Leonberg, Germany. Correspondence to: Niklas Freymuth <niklas.freymuth@kit.edu>.

Proceedings of the ICML 2024 AI for Science worksop, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

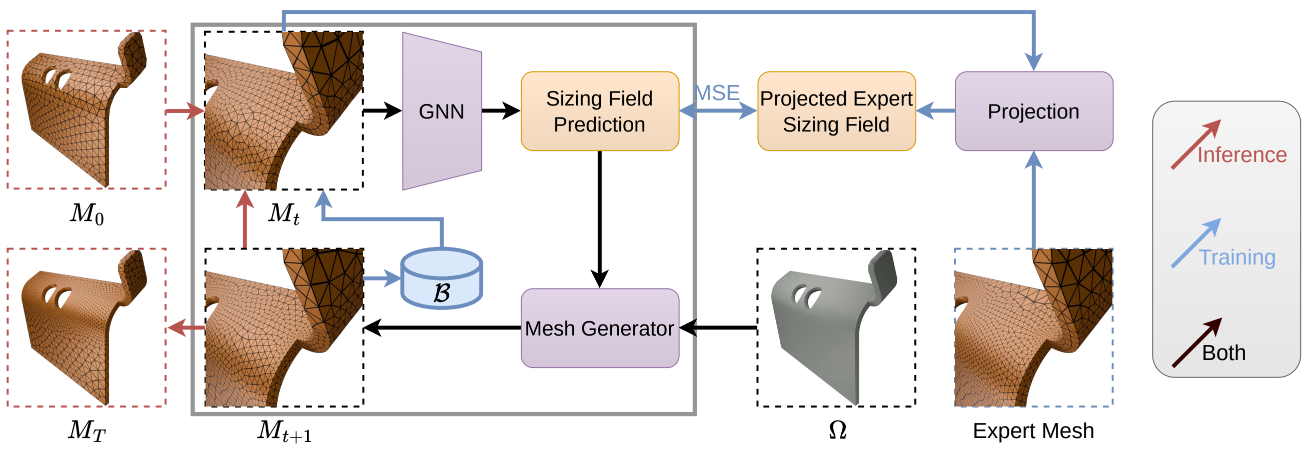

Figure 1: Schematic overview of Adaptive Meshing By Expert Reconstruction (AMBER). During inference, AMBER takes an initial mesh M_0, predicts a sizing field per element, and combines this with the underlying geometry in a mesh generator which produces an improved mesh. This process is repeated until a final mesh M_T is obtained. When training, AMBER is tasked to predict the projected sizing field of an expert mesh for samples from a replay buffer, adding samples to this buffer to maintain a large and accurate distribution of training meshes.

Lock et al., 2024) to predict the target mesh. However, these approaches are limited by the inherent properties of CNNs, such as a fixed resolution or missing rotational equivariance.

In this work, we instead frame AMG as an imitation learning problem, aiming to create a mesh that matches expert behavior by iteratively predicting a sizing field on an intermediate mesh to create a more accurate next mesh. Our approach, AMBER, uses a Graph Neural Network (GNN) that consumes a graph representation of the current mesh and outputs a piecewise-constant sizing field on the mesh elements. During inference, it then takes a geometry and a coarse initial mesh and iteratively improves this mesh by predicting a sizing field that is used to create the next mesh. During training, this predicted sizing field is trained to be close to a sizing field that is projected from the expert mesh using a simple regression loss. As the algorithm can create arbitrary intermediate meshes during inference, we maintain a replay buffer (Lin, 1992; Fedus et al., 2020) during training, regularly adding meshes produced by our algorithm to this buffer and training on them. Training on these buffered meshes instead of only using the initial meshes ensures there is no distribution shift between the simple regression during training and the iterative mesh generation procedure during inference. We automatically label these intermediate meshes by projecting the expert sizing field onto them. This process resembles the DAgger (Ross et al., 2011) in which an expert relabels out of distribution samples. In AMBER, we implement a similar approach, however, we use an oracle for re-labeling. Figure 1 provides a schematic overview.

We experimentally validate our approach against a CNN baseline on heuristic expert meshes generated on a series of 2D Poisson Equations on randomly generated L-Shaped geometries. Additionally, we experiment on a set of complex real-world 3D geometries with meshes created by a human expert. We find that AMBER is particularly well-suited for highly adaptive meshes and produces accurate imitations of both heuristic and human expert behavior, regardless of the underlying mesh resolution. We further show the effectiveness of AMBER's individual design choices through a series of ablations.[1]

To summarize our contributions, we **(i)** propose AMBER, a novel imitation learning approach that generates expert-like meshes through a series of sizing field predictions, **(ii)** introduce challenging 2D and 3D datasets consisting of heuristic and human expert meshes and corresponding geometries, and **(iii)** evaluate our method on these datasets, demonstrating high similarity to the provided expert meshes and outperforming a strong CNN-based baseline.

2. Related Work

Meshing for Simulation. The Finite Element Method (FEM) is a well-established method to numerically approximate physical systems, especially on complex, irregular geometric problem domains (Brenner & Scott, 2008; Reddy, 2019). The FEM solves a system of equations by dividing the geometry into a mesh consisting of smaller, simple elements. Here, the simulation cost and accuracy directly depend on the mesh size, often necessitating an adaptive mesh that focuses more elements on important parts of the geometry for efficient simulations (Plewa et al., 2005; Huang & Russell, 2010). Modern meshing approaches can be cat-

[1]Code and datasets are available at https://github.com/NiklasFreymuth/AMBER.

egorized into Adaptive Mesh Refinement (AMR) (Plewa et al., 2005; Fidkowski & Darmofal, 2011), which adapts a given mesh, and Adaptive Mesh Generation (AMG) (Yano & Darmofal, 2012; Remacle et al., 2013; Si, 2008), which generates a new mesh from an estimated sizing field or related properties of the geometry. These traditional AMR approaches rely on heuristics (Zienkiewicz & Zhu, 1992) or error estimates (Nemec et al., 2008; Bangerth & Rannacher, 2013), which can quickly become inaccurate, unreliable or computationally expensive (Bangerth & Rannacher, 2013; Cerveny et al., 2019; Wallwork, 2021).

Learning Based Approaches for Meshing. In recent years, GNNs (Bronstein et al., 2021), and particularly Message Passing Networks (MPNs), have become popular architectures for learning to simulate on meshes. These Graph Network Simulators (GNSs) have been applied to mesh-based deformable (Pfaff et al., 2021; Linkerhägner et al., 2023) and rigid (Allen et al., 2022; 2023; Lopez-Guevara et al., 2024) object simulations. We also employ MPNs acting on meshes to improve the speed and accuracy of physical simulations. Yet, instead of directly learning to simulate, we generate application-specific efficient meshes for downstream classical FEM solvers, which is more robust and safe as the actual simulation is done numerically. Here, a recent suite of RL-based AMR approaches has emerged for creating adaptive meshes (Foucart et al., 2023; Freymuth et al., 2023; Yang et al., 2023a). These methods use an RL policy to iteratively determine which mesh elements to subdivide. They typically depend on complex and costly reward functions, which need careful design to handle complex problems (Freymuth et al., 2023). Despite this, the reward function requires a concrete underlying system of equations, restricts the maximum mesh resolution (Yang et al., 2023a; Freymuth et al., 2023) or encodes a specific refinement criterion (Foucart et al., 2023). These limitations are sub-optimal for real-world applications, where meshes often need to be highly locally refined, e.g., due to localized large gradients (Larson & Bengzon, 2013). We instead learn to refine based on expert meshes, omitting the need for a reward function and thus alleviating its limitations.

Other works consider supervised learning for mesh refinement. These approaches use recurrent neural networks to learn mesh refinement strategies (Bohn & Feischl, 2021), optimize element stretching ratio and direction from a given error estimation (Fidkowski & Chen, 2021), and employ hand-crafted features to directly compute error estimates to solve an adjoint problem (Roth et al., 2022; Wallwork et al., 2022). Further work trains a surrogate model with supervised learning for predicting the solution error (Chen & Fidkowski, 2021; Zhang et al., 2020; 2021), which in turn can be used for AMG by setting the sizing field to the scaled inverse of the error estimation (Zhang et al., 2021). Other approaches directly predict a sizing field using MLPs for simulations of magnetic devices (Dyck et al., 1992), and CNNs for fluid dynamics problem (Huang et al., 2021).

Interactive Imitation Learning. Imitation Learning (IL) is a common paradigm to learn behavior from expert demonstrations (Pomerleau, 1988; Osa et al., 2018; Shafiullah et al., 2022), which is particularly applicable in scenarios where a reward function is difficult to define or can lead to undesired behavior (Skalse et al., 2024). A major challenge in IL occurs when the model is confronted with data absent from the expert data as it diverges from the expert behavior. Interactive IL methods, such as DAgger (Ross et al., 2011), address this issue by allowing experts to intervene and label new data. In some variants of this approach (Menda et al., 2019; Hoque et al., 2022), the model actively requests new data from the expert when needed to improve efficiency further. Similarly, AMBER also actively acquires new data during learning. However, instead of querying a human for new expert meshes, we maintain a static expert dataset and automatically generate and label new intermediate meshes based on the induced expert sizing fields.

3. Adaptive Meshing by Expert Reconstruction (AMBER)

AMBER views mesh generation as an iterative imitation problem aiming to match an expert mesh based on a series of intermediate meshes. In contrast to previous AMR methods (Zienkiewicz & Zhu, 1992; Yang et al., 2023a; Freymuth et al., 2023), we learn directly from meshes optimized for their respective downstream applications by experts. This view greatly simplifies the mesh generation process, as our problem consists of matching an expert rather than optimizing for problem-specific and potentially complex adaptive mesh criteria (Larson & Bengzon, 2013; Huang et al., 2021; Dolejší & May, 2022). We approach this problem by training a MPN to iteratively predict sizing fields on a series of intermediate meshes, generating a new mesh from each prediction. These projections are regressed against a projected expert sizing field, which is dynamically produced for meshes that are added to a replay buffer during training (Ross et al., 2011; Kelly et al., 2019). Figure 1 provides a schematic overview of AMBER, while the following sections describe its different aspects in more detail.

3.1. Preliminaries

We are given a training dataset $\mathcal{D} = \{(\Omega, M^*)\}_{n=1}^{N}$ consisting of pairs of a geometry Ω and a corresponding expert mesh M^*. Each geometry describes a physical body in 2D or 3D, and the mesh discretizes this body into a number of simplical elements M_j^*. We aim to generate a mesh M^T that imitates the expert mesh M^*, i.e., that minimizes a distance $d(M^T, M^*)$ for unseen meshes during inference.

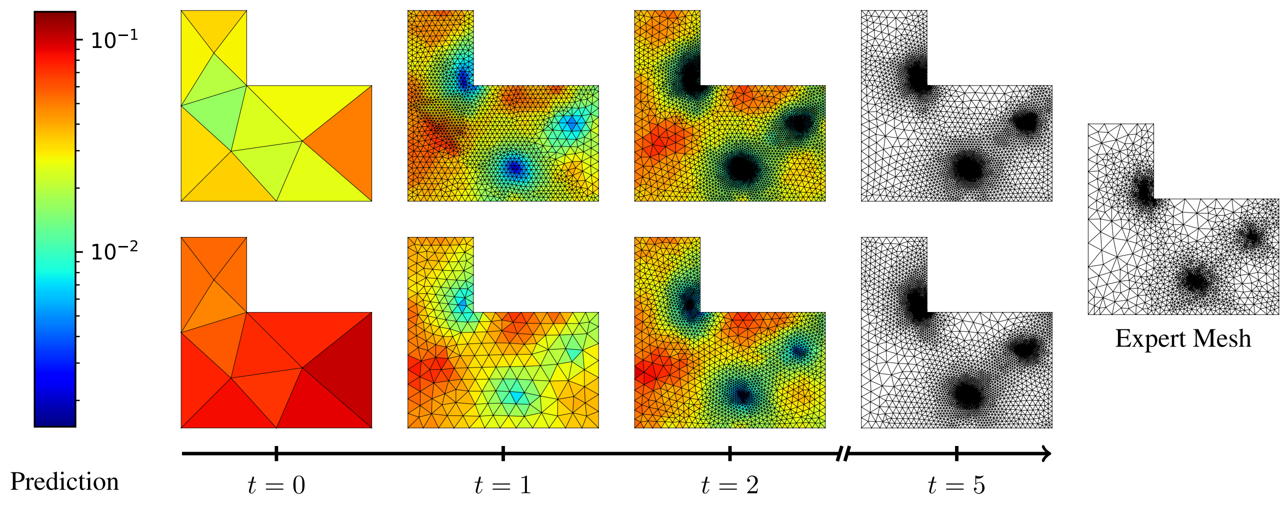

Figure 2: Intermediate and final AMBER meshes on Poisson's equation for an expert mesh with 50 reference refinements. The colorbar on the left denotes the predicted sizing field per element for each intermediate mesh. This prediction is given to a mesh generator to produce the next mesh. **Top:** AMBER (Mean) converges in a few generation steps at the cost of additional elements in intermediate steps. **Bottom:** AMBER (Max) instead yields more conservative predictions, which take longer to converge but produce less total mesh elements. In both cases, the mesh generator and the MPN architecture favor smooth solutions, generating a mesh that closely matches the expert but has less abrupt variations in local element size.

This generation is done using out-of-the-box approaches. In this work, we use the Frontal Delaunay algorithm implemented in *gmsh* (Geuzaine & Remacle, 2009). The mesh generator consumes a sizing field and a geometry and returns a mesh that conforms to this sizing field. It additionally optimizes the mesh with respect to certain element criteria, such as the element's aspect ratio, which results in a smooth transition between element sizes. In turn, the sizing field is a function $\Omega \to \mathbb{R}$ that describes the average desired edge length of the mesh's elements in space. We can thus precisely a given mesh by the piece-wise constant sizing field that is induced by its elements. Given the volume $V(M_i)$ of the d-dimensional simplical element M_i, its corresponding sizing field is given as the average edge length

$$f(M_i) = \left(V(M_i)\frac{d!}{\sqrt{d+1}}\right)^{\frac{1}{d}}.$$

To obtain a graph representation, we view the mesh elements as nodes $\mathcal{V}$ and their neighborhood relations as edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. We then construct a bidirectional graph $\mathcal{G}_{\Omega^t} = \mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{x}_\mathcal{V}, \mathbf{x}_\mathcal{E})$ using node features $\mathbf{x}_\mathcal{V}$ and edge features $\mathbf{x}_\mathcal{E}$. We process the graph using a MPN (Pfaff et al., 2021), a type of GNN (Bronstein et al., 2021). MPNs encompass the function class of several classical PDE solvers (Brandstetter et al., 2022), making them a popular choice for learning representations on meshes (Pfaff et al., 2021; Linkerhägner et al., 2023; Würth et al., 2024). Appendix A provides details for the MPN architecture.

3.2. Mesh generation with AMBER

When generating a mesh for a given geometry, AMBER starts with a coarse uniform initial mesh M^t for $t = 0$. It then encodes the current mesh M^t as a graph and feeds it through an MPN to predict a target size per mesh element. The combination of these target sizes forms a piece-wise constant sizing field, which together with the underlying geometry is given to a mesh generator, which produces a new mesh M^{t+1}. This process is iterated over T steps, resulting in a final mesh M^T.

This iteration is necessary, as the current mesh limits the resolution of the predicted sizing field in every step. As we can only learn one target size for each current element of M^t, the corresponding finer resolution elements in M^{t+1} will all have roughly the same size. However, by iterating this process, the final mesh converges to an appropriate mesh, even for problems with highly non-uniform meshes.

3.3. Training AMBER

AMBER's iterative mesh generation process necessitates a specialized training procedure. To compute the target sizing field for an element $M_i^t \subseteq M^t$ of an intermediate mesh M^t, we first determine all expert elements $M_j^* \in M^*$ whose midpoints $p_{M_j^*}$ lie in M_i^t, and then aggregate over the sizing fields induced by these elements, i.e.,

$$y(M_i^t) = \bigotimes_j \mathbb{I}(p_{M_j^*} \in M_i^t) f(M_j^*).$$

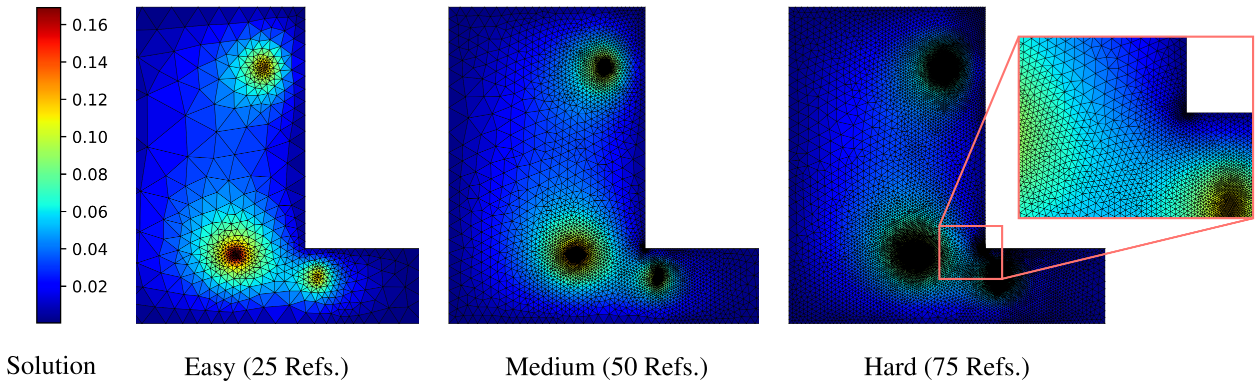

Figure 3: Exemplary AMBER (Mean) refinements for Poisson's Equation trained on expert meshes with (**Left**) 25, (**Middle**) 50, and (**Right**) 75 refinement steps. We plot the solution of Poisson's Equation as a color plot. The right figure includes a zoom on the concave edge of the domain.

Here, $\bigotimes$ is a permutation-invariant aggregation function and we consider using either a maximum or a mean operator. This results in two variants, called AMBER (Mean) and AMBER (Max.). Intuitively, we thus set each element's target element size to either the average or the maximum size of all of the expert elements it contains. As the mean target size is always smaller or equal to the maximum target size, AMBER (Mean) tends to refine more aggressively than AMBER (Max) resulting in faster convergence while producing more and potentially unnecessary elements in the process. In both cases, the target sizes for each element of the intermediate mesh will be smaller than their actual size, and iterating this process converges to the expert mesh.

If an element in M^t to contains no midpoint $p_{M_j^*}$, we set its target sizing field to that of the element in the expert mesh that contains their midpoint. Further, different meshes may approximate the same underlying geometry differently, leading to elements in one mesh that lie outside of the other. We thus assign elements in the expert mesh that lie outside of the intermediate mesh to their nearest neighbor in the intermediate mesh. This approximation ensures that all elements in the fine mesh contribute to the target sizing field of the intermediate mesh, and becomes more accurate the more closely the two meshes align. Once the targets sizing field are obtained, we use them to train on a simple node-level Mean Squared Error (MSE) loss. This training scheme allows us to equally apply AMBER to both problem-specific meshes optimized for a given PDE, and general-purpose meshes intended for different downstream analyses.

AMBER produces a series of intermediate meshes in an auto-regressive fashion during inference, but only trains on individual sizing field predictions. To prevent distribution shifts during the iterative mesh generation, we thus maintain a replay buffer (Lin, 1992; Fedus et al., 2020) over AMBER-generated intermediate meshes during training. This replay buffer contains a fixed set of one coarse initial mesh per provided training sample. Every k training steps, we sample a mesh from the replay buffer, predict its projected sizing field, generate a new mesh from this prediction, project the expert field onto this new mesh to generate its labels, and store this new pair of meshes and labels in the buffer. To stabilize the training process, we assign each intermediate mesh a 'depth' that corresponds to the number of generating steps it has gone through, and store an equal amount of meshes for each depth up to a maximum. Each training step consists of randomly drawing a number of meshes from the buffer. Since the meshes greatly vary in size, we fill a batch until a number of nodes is reached, rather than using a fixed number of graphs. Appendix B provides details on the stratified replay buffer and the batch generation process.

4. Experiments

Setup. To use an MPN for meshes, we set the node features to the corresponding element's volume and the edge features $\mathbf{x}_{\mathcal{E}}$ are to the Euclidean distance between element midpoints. MPNs are permutation-equivariant by design and as we only provide Euclidean distances as spatial information, we additionally obtain equivariance to the Euclidean group (Bronstein et al., 2021). Since the sizing field's values are always positive we apply a softplus transformation to the MPN outputs, which greatly stabilizes training. Similarly, we normalize the input graph features using the statistics of the replay buffer. Appendix E provides further details of the architecture, training procedure, and hyperparameters.

We repeat all methods for 4 random seeds. We evaluate on a set of unseen test geometries, and evaluate the quality of each generated mesh by comparing it to a reference expert mesh on this geometry. For this comparison, we use the Density-Aware Chamfer Distance (DCD) (Wu et al., 2021)

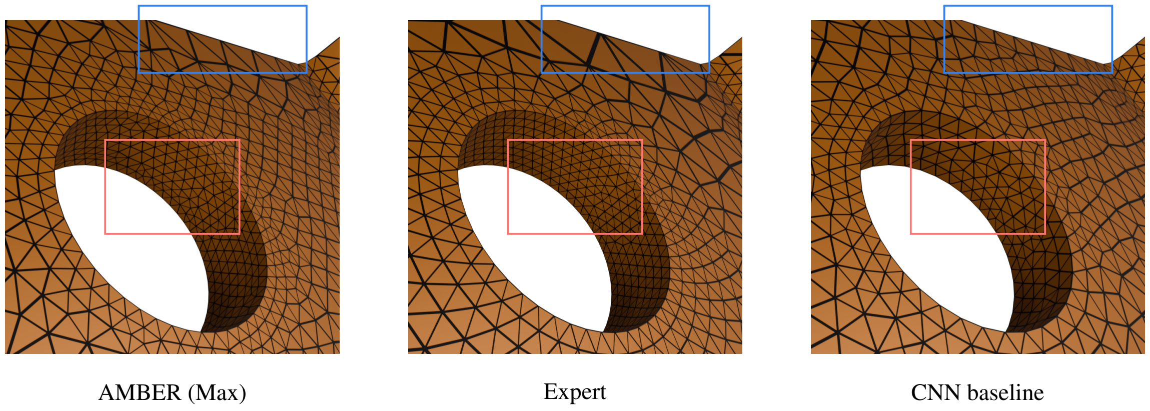

Figure 4: Zoom-in of a final generated meshes for an unseen test domain on the Console task. AMBER (**Left**) closely matches the expert mesh (**Middle**), producing finer elements near the hole and coarser elements on the upper border of the mesh. In comparison, the CNN baseline (**Right**), which acts on a $64 \times 64 \times 64$ 3D image of the domain, has less variation in the element size and matches the expert less closely.

over the sets of element midpoints of both meshes. The DCD is defined as a symmetric, exponentiated Chamfer distance with normalization factors to account for elements in one set that match multiple elements in the other. Intuitively, it measures the closest exponential distance of every midpoint in one mesh to the other mesh, weighting this distance by the number of matches. It is bound in $[0, 1]$, and 0 if and only if both sets are the same. We provide a mathematical description of the DCD in Appendix C. We also explore an alternate metric that numerically integrates the difference in element volumes between the two meshes.

Poisson's Equation. We consider problem-specific meshes using Poisson's Equation with a load function $f(\boldsymbol{x})$ and test function $v(\boldsymbol{x})$, given as

$$\int_\Omega \nabla u \cdot \nabla v \mathrm{d}\boldsymbol{x} = \int_\Omega f v \mathrm{d}\boldsymbol{x} \quad \forall v \in V.$$

We enforce zero Dirichlet boundary conditions, i.e., $u(\boldsymbol{x}) = 0$ on $\partial\Omega$. As the training data, we randomly generate L-shaped geoemtries and employ a Gaussian Mixture Model with three components for the load function. For this task, we evaluate the load function at each face's midpoint and use it as a node feature. Additionally, we solve the equation for every intermediate mesh and use the mean and standard deviation of the solution on each element's vertices as additional node features. We generate the problem-specific expert meshes using an error-based refinement heuristic (Binev et al., 2004; Bangerth et al., 2012; Foucart et al., 2023), which marks all elements for refinement for which $\mathrm{err}(M_i) > \theta \cdot \max_j \mathrm{err}(M_j)$ for $0 < \theta < 1$. The error is estimated heuristically based on the load function and gradient jumps on the element facets. We use 20 randomly drawn systems of equations and corresponding expert meshes as the training data, and test on a disjoint, fixed set of 100 unseen system of equations and expert meshes.

We vary the task's difficulty by adapting the number of refinement steps that the heuristic applies, considering *easy*, *medium*, and *hard* variants with 25, 50, and 75 refinement steps respectively. Appendix D provides further details of the dataset creation and the expert heuristic.

Console Task. In the *Console task*, we use data obtained from a real-world scenario in the automotive industry. We have a parameterized family of 3D geometries representing a car's seat crossmembers. The geometries are obtained using Onshape[2] and feature various sharp bends as well as up to 2 circular holes. Tetrahedral meshes for this task are generated by human experts using ANSA[3]. The experts are initially presented with a coarse mesh, on which they iteratively select regions to refine, specifying the target element size of each selected region. The resulting meshes are optimized for downstream strength and durability analyses, but our experiments are conducted solely on the meshes and their underlying geometry. The middle of Figure 6 provides an example close-up of an expert mesh. The final dataset contains 22 meshes and corresponding geometries. The meshes range from 6,284 to 41,856 elements and are randomly split into fixed sets of 15 training, 2 evaluation, and 5 test meshes.

Baselines and Ablations. We compare AMBER to a CNN-baseline that takes a binary image mask of the geometry and outputs a pixel-wise sizing field on the geometry. This baseline is conceptually similar to (Huang et al., 2021), but with slight adaptations that improve the training stability and performance. Instead of smoothing the expert sizing field and evaluating each pixel, we create an auxiliary mesh with roughly one element per pixel, calculate the target sizing field on this mesh as done in Section 3.3, and map

[2]https://www.onshape.com/
[3]https://www.beta-cae.com/ansa.htm

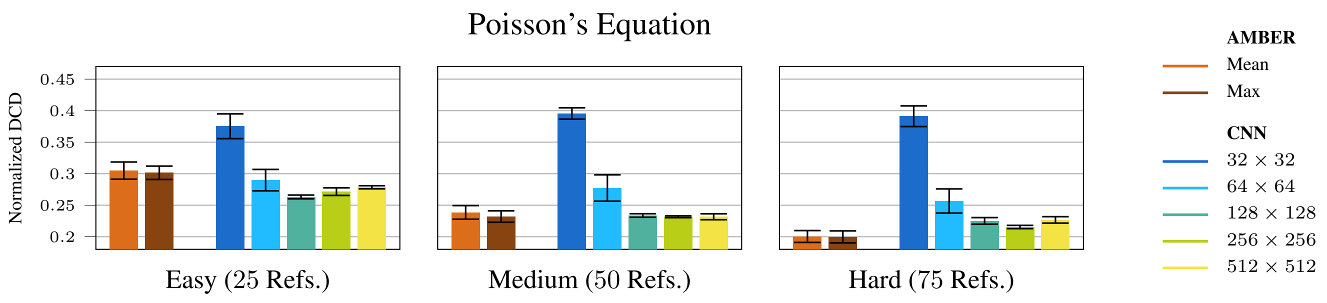

Figure 5: Mean and quantiles of the normalized DCD for AMBER and the CNN baseline for different input image resolutions. (**Left**) Poisson Easy; the resulting meshes are comparatively coarse and can be reproduced well with the CNN baseline. (**Middle**) Poisson Medium; the CNN starts to require a higher input resolution for good refinements. (**Right**) Poisson Hard; the expert mesh covers elements across multiple scales. Here, the CNN fails to provide good refinements for lower image resolutions and begins to overfit for higher resolutions. In contrast, AMBER directly acts on intermediate meshes and can thus dynamically adapt the sampling resolution of the sizing fields it predicts.

the resulting labels to each pixel. Since the CNN does no iterative generation, we use the mean label projection method for each pixel. Further, we train the approach on the logarithm of the sizing field, i.e., set the loss to

$$\mathcal{L}_{\text{CNN}} = \frac{1}{N}\sum_{i=1}^{N}\left(\text{CNN}(x_i) - \log(y_i)\right)^2, \tag{1}$$

and subsequentially exponentiate the CNN's output to recover the predicted sizing field. This change prevents larger mesh elements from dominating the CNN's loss. The CNN can only predict one target element size per pixel and we thus train separate CNNs on different image resolutions. We also provide the same input features and normalization as for AMBER. Since the training sets are relatively small, we use all images for every training step, performing a total of $25,600$ training steps or equivalently epochs of training.

5. Results

Qualitative Results. Figure 2 visualizes exemplary AMBER refinements trained on the Poisson task with 50 oracle heuristic refinement steps. On the top row, AMBER (Mean) converges to a relatively accurate mesh after only 2 prediction steps. However, the approach tends to locally over-refine some elements, causing potentially unnecessary refinements in intermediate meshes. AMBER (Max.) is shown on the bottom row. It produces fewer elements per step and thus requires more iterations until it converges to the expert mesh. Figure 3 shows AMBER (Mean) refinements for the same test example when trained on expert meshes that use 25, 50 and 75 heuristic refinement steps. Our approach closely matches the expert heuristic, producing meshes of varying granularity depending on the expert meshes that it sees during training. Figure 4 compares AMBER (Max) and the CNN baseline to an expert mesh on an exemplary test geometry on the Console task, finding that AMBER matches the expert's local element resolution well, while the CNN produces more uniform meshes and does not provide accurate sizing field predictions for the different geometric features. Appendix F provides visualizations for AMBER and the CNN baseline for all tasks.

Quantitative Results. We measure mesh quality via the similarity to the expert mesh using the normalized Density-Aware Chamfer Distance (DCD) (Wu et al., 2021) for AMBER for both mean and max sizing field interpolations and the CNN baseline for different image resolutions. Figure 5 evaluates Poisson's Equation for 25, 50 and 75 heuristic refinement steps of the expert mesh. While the CNN baseline works well for 25 and even 50 steps, its fixed image resolution eventually causes issues as the expert meshes become more complex. In contrast, AMBER directly acts on intermediate meshes of arbitrary resolution. Here, AMBER works well on all 3 task difficulties, as the method learns to adapt to the complexity of the expert's meshes. This trend is more pronounced on the 3D console task, as shown on the left side of Figure 6. AMBER naturally generalizes to the more complex 3D setting as it directly acts on meshes. Thus, each AMBER forward pass is linear in the number of mesh elements. In contrast, the CNN baseline quickly becomes computationally expensive due to cubic scaling with the number of voxels per dimension. Finally, Figure 6 shows the benefits of AMBER's iterative mesh generation process, finding that both AMBER (Mean) and AMBER (Max) benefit from up to 5 intermediate mesh generation steps. Appendix C.2 evaluates all tasks on an alternate metric that measures the difference in element volumes between the two meshes, yielding consistent results.

Further Experiments. Figure 7 shows that AMBER performs slightly worse when trained on the CNN loss (c.f. Equation 1), likely because AMBER already implicitly weights small elements more strongly as there are more of them. In contrast, the CNN baseline performs worse on AMBER's MSE loss, presumably because the same rela-

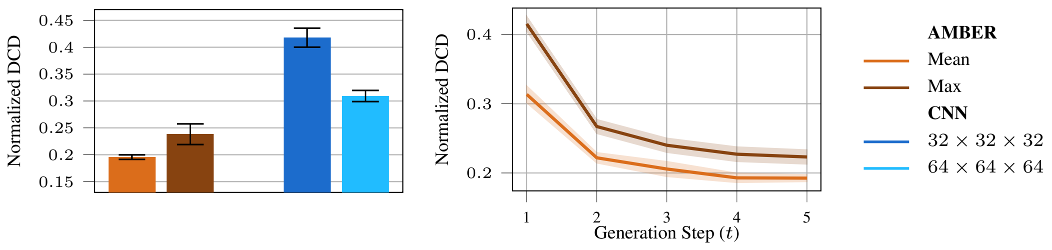

Figure 6: Mean and quantiles of normalized DCD for AMBER and the CNN baseline for the Console task. (**Left**) AMBER naturally generalizes to the more complex 3D domains of the console task, generating meshes that accurately match that of the human expert on unseen domains. In contrast, the CNN baseline does not scale well to 3 dimensions, as evaluating the resulting 3D image quickly becomes prohibitively expensive with an increased image resolution. (**Right**) For both AMBER (Mean) and AMBER (Max) mesh quality smoothly increases with more mesh generation steps.

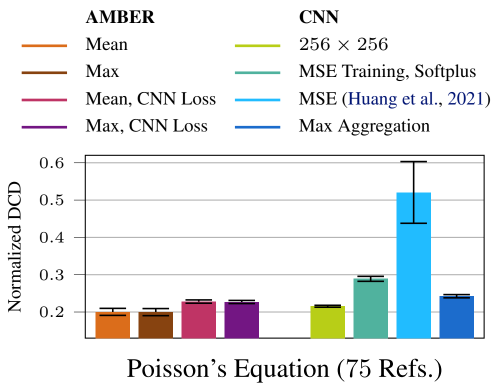

Figure 7: Mean and quantiles of normalized DCD for AMBER and the CNN baseline with a 256 × 256 resolution for different loss functions and sizing field interpolation types. AMBER performs worse when using the CNN loss of Equation 1. In contrast, the CNN baseline yields worse meshes when trained on AMBER's MSE loss. Omitting the softplus transformation of the predicted outputs results in a method similar to (Huang et al., 2021), but leads to a significantly less robust algorithm. Finally, using the maximum expert sizing field for the labels of the CNN baseline yields worse results as the target sizing file is too coarse.

tive error makes a larger difference in mesh generation for pixels responsible for smaller elements when using this loss. Omitting the softplus transformation of the output on this loss, which results in a method similar to that of (Huang et al., 2021), further decreases stability and performance. Finally, using a maximum operator to project the sizing fields for the CNN baseline leads to more conservative target estimation and thus too few elements, yielding worse results. We explore additional variants and ablations of AMBER in Appendix F and find that AMBER benefits from additional data but provides strong generalization performance to unseen scenarios from as few as 5 expert meshes on Poisson's Equation. Appendix F validates the effectiveness of normalizing the input features, using a softplus transformation to predict the strictly positive sizing field, and maintaining an equal number of meshes for each intermediate generation step instead of randomly adding samples.

6. Conclusion

We introduce Adaptive Meshing By Expert Reconstruction (AMBER), an iterative adaptive mesh generation algorithm based on imitation learning. AMBER combines a Message Passing Graph Neural Network and a replay buffer with automatically labeled intermediate meshes to learn how to imitate meshes produced by experts. During mesh generation, AMBER takes an arbitrary problem geometry and generates a series of increasingly accurate intermediate meshes by predicting a sizing field on each mesh and feeding this sizing field to an out-of-the-box mesh generator. Rather than optimizing any particular metric, AMBER learns to imitate an expert's meshing behavior from examples, enabling it to create suitable meshes for various engineering applications.

Experimental results on heuristic 2D meshes on Poisson problems, and human expert meshes on real-world 3D geometries demonstrate the strong performance of our approach. AMBER is data efficient and generates highly accurate meshes, significantly outperforming a strong CNN baseline when evaluated on more complex task setups.

Limitations and Future Work. AMBER requires a dataset with consistent expert behavior and always produces meshes that align with this single behavior. We plan to extend AMBER to allow conditioning the mesh generation process on a desired expert behavior, e.g. producing varying numbers of final elements with the same model (Yang et al., 2023b). Additionally, we aim to replace the current piece-wise constant sizing field over the elements with a piece-wise linear sizing field over the mesh's vertices. This change reduces the size of the mesh's graph representation while increasing the information provided in each sizing field.

Acknowledgements

NF was supported by the BMBF project Davis (Datengetriebene Vernetzung für die ingenieurtechnische Simulation). This work is also part of the DFG AI Resarch Unit 5339 regarding the combination of physics-based simulation with AI-based methodologies for the fast maturation of manufacturing processes. The financial support by German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) is gratefully acknowledged. The authors acknowledge support by the state of Baden-Württemberg through bwHPC, as well as the HoreKa supercomputer funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the German Federal Ministry of Education and Research.

References

Allen, K. R., Guevara, T. L., Rubanova, Y., Stachenfeld, K., Sanchez-Gonzalez, A., Battaglia, P., and Pfaff, T. Graph network simulators can learn discontinuous, rigid contact dynamics. *Conference on Robot Learning (CoRL).*, 2022.

Allen, K. R., Rubanova, Y., Lopez-Guevara, T., Whitney, W. F., Sanchez-Gonzalez, A., Battaglia, P., and Pfaff, T. Learning rigid dynamics with face interaction graph networks. In *The Eleventh International Conference on Learning Representations*, 2023.

Anderson, R., Andrej, J., Barker, A., Bramwell, J., Camier, J.-S., Cerveny, J., Dobrev, V., Dudouit, Y., Fisher, A., Kolev, T., et al. Mfem: A modular finite element methods library. *Computers & Mathematics with Applications*, 81: 42–74, 2021.

Ba, L. J., Kiros, J. R., and Hinton, G. E. Layer normalization. *CoRR*, abs/1607.06450:21, 2016.

Bangerth, W. and Rannacher, R. *Adaptive Finite Element Methods for Differential Equations*. Birkhäuser, 2013.

Bangerth, W., Burstedde, C., Heister, T., and Kronbichler, M. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Transactions on Mathematical Software (TOMS)*, 38(2):1–28, 2012.

Binev, P., Dahmen, W., and DeVore, R. Adaptive finite element methods with convergence rates. *Numerische Mathematik*, 97:219–268, 2004.

Bohn, J. and Feischl, M. Recurrent neural networks as optimal mesh refinement strategies. *Computers & Mathematics with Applications*, 97:61–76, 2021.

Brandstetter, J., Worrall, D. E., and Welling, M. Message passing neural pde solvers. In *International Conference on Learning Representations*, 2022.

Brenner, S. C. and Scott, L. R. *The mathematical theory of finite element methods*, volume 3. Springer, 2008.

Bronstein, M. M., Bruna, J., Cohen, T., and Velickovic, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478, 2021.

Carstensen, C. An adaptive mesh-refining algorithm allowing for an h 1 stable l 2 projection onto courant finite element spaces. *Constructive Approximation*, 20:549–564, 2004.

Cerveny, J., Dobrev, V., and Kolev, T. Nonconforming mesh refinement for high-order finite elements. *SIAM Journal on Scientific Computing*, 41(4):C367–C392, 2019.

Chen, G. and Fidkowski, K. J. Output-based adaptive aerodynamic simulations using convolutional neural networks. *Computers & Fluids*, 223:104947, 2021.

Dolejší, V. and May, G. *Anisotropic Hp-Mesh Adaptation Methods: Theory, Implementation and Applications*. Nečas Center Series. Springer International Publishing, Cham, 2022. ISBN 978-3-031-04278-2 978-3-031-04279-9. doi: 10.1007/978-3-031-04279-9.

Dyck, D., Lowther, D., and McFee, S. Determining an approximate finite element mesh density using neural network techniques. *IEEE Transactions on Magnetics*, 28(2):1767–1770, March 1992. ISSN 1941-0069. doi: 10.1109/20.124047.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pp. 3061–3071. PMLR, 2020.

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

Fidkowski, K. J. and Darmofal, D. L. Review of output-based error estimation and mesh adaptation in computational fluid dynamics. *AIAA journal*, 49(4):673–694, 2011.

Foucart, C., Charous, A., and Lermusiaux, P. F. Deep reinforcement learning for adaptive mesh refinement. *Journal of Computational Physics*, 491:112381, 2023.

Freymuth, N., Dahlinger, P., Würth, T., Reisch, S., Kärger, L., and Neumann, G. Swarm reinforcement learning for adaptive mesh refinement. *Advances in Neural Information Processing Systems*, 36, 2023.

Geuzaine, C. and Remacle, J.-F. 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.

Giannakoglou, K. C. and Papadimitriou, D. I. Adjoint methods for shape optimization. *Optimization and computational fluid dynamics*, pp. 79–108, 2008.

Giles, M. B. and Pierce, N. A. An introduction to the adjoint approach to design. *Flow, turbulence and combustion*, 65:393–415, 2000.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hoque, R., Balakrishna, A., Novoseller, E., Wilcox, A., Brown, D. S., and Goldberg, K. Thriftydagger: Budget-aware novelty and risk gating for interactive imitation learning. In Faust, A., Hsu, D., and Neumann, G. (eds.), *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 598–608. PMLR, 08–11 Nov 2022.

Huang, K., Krügener, M., Brown, A., Menhorn, F., Bungartz, H.-J., and Hartmann, D. Machine learning-based optimal mesh generation in computational fluid dynamics. *arXiv preprint arXiv:2102.12923*, 2021.

Huang, W. and Russell, R. D. *Adaptive moving mesh methods*, volume 174. Springer Science & Business Media, 2010.

Kelly, M., Sidrane, C., Driggs-Campbell, K., and Kochenderfer, M. J. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8077–8083. IEEE, 2019.

Kita, E. and Kamiya, N. Error estimation and adaptive mesh refinement in boundary element method, an overview. *Engineering Analysis with Boundary Elements*, 25(7): 479–495, 2001.

Larson, M. G. and Bengzon, F. *The Finite Element Method: Theory, Implementation, and Applications*, volume 10 of *Texts in Computational Science and Engineering*. Springer, Berlin, Heidelberg, 2013. ISBN 978-3-642-33286-9 978-3-642-33287-6. doi: 10.1007/978-3-642-33287-6.

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321, 1992.

Linkerhägner, J., Freymuth, N., Scheikl, P. M., Mathis-Ullrich, F., and Neumann, G. Grounding graph network simulators using physical sensor observations. In *The Eleventh International Conference on Learning Representations*, 2023.

Lock, C., Hassan, O., Sevilla, R., and Jones, J. Predicting the Near-Optimal Mesh Spacing for a Simulation Using Machine Learning. In Ruiz-Gironés, E., Sevilla, R., and Moxey, D. (eds.), *SIAM International Meshing Roundtable 2023*, volume 147, pp. 115–136. Springer Nature Switzerland, Cham, 2024. ISBN 978-3-031-40593-8 978-3-031-40594-5. doi: 10.1007/978-3-031-40594-5_6.

Lopez-Guevara, T., Rubanova, Y., Whitney, W. F., Pfaff, T., Stachenfeld, K., and Allen, K. R. Scaling face interaction graph networks to real world scenes. *arXiv preprint arXiv:2401.11985*, 2024.

Menda, K., Driggs-Campbell, K., and Kochenderfer, M. J. Ensembledagger: A bayesian approach to safe imitation learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5041–5048. IEEE, 2019.

Mukherjee, A. *An adaptive finite element code for elliptic boundary value problems in three dimensions with applications in numerical relativity*. The Pennsylvania State University, 1996.

Nemec, M., Aftosmis, M., and Wintzer, M. Adjoint-based adaptive mesh refinement for complex geometries. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, pp. 725, 2008.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., Peters, J., et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7 (1-2):1–179, 2018.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

Plewa, T., Linde, T., Weirs, V. G., et al. *Adaptive mesh refinement-theory and applications*. Springer, 2005.

Pomerleau, D. A. Alvinn: An autonomous land vehicle in a neural network. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988.

Reddy, J. N. *Introduction to the finite element method*. McGraw-Hill Education, 2019.

Remacle, J.-F., Henrotte, F., Carrier-Baudouin, T., Béchet, E., Marchandise, E., Geuzaine, C., and Mouton, T. A frontal delaunay quad mesh generator using the l-infinity norm. *International Journal for Numerical Methods in Engineering*, 94(5):494–512, 2013.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241. Springer, 2015.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Roth, J., Schröder, M., and Wick, T. Neural network guided adjoint computations in dual weighted residual error estimation. *SN Applied Sciences*, 4(2):62, 2022.

Shafiullah, N. M., Cui, Z., Altanzaya, A. A., and Pinto, L. Behavior transformers: Cloning k modes with one stone. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22955–22968. Curran Associates, Inc., 2022.

Si, H. Adaptive tetrahedral mesh generation by constrained delaunay refinement. *International Journal for Numerical Methods in Engineering*, 75(7):856–880, 2008.

Skalse, J., Howe, N. H. R., Krasheninnikov, D., and Krueger, D. Defining and characterizing reward hacking. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9781713871088.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Wallwork, J. G. *Mesh adaptation and adjoint methods for finite element coastal ocean modelling*. PhD thesis, Imperial College London, 2021.

Wallwork, J. G., Lu, J., Zhang, M., and Piggott, M. D. E2n: Error estimation networks for goal-oriented mesh adaptation. *arXiv preprint arXiv:2207.11233*, 2022.

Wu, T., Pan, L., Zhang, J., Wang, T., Liu, Z., and Lin, D. Density-aware chamfer distance as a comprehensive metric for point cloud completion. *arXiv preprint arXiv:2111.12702*, 2021.

Würth, T., Freymuth, N., Zimmerling, C., Neumann, G., and Kärger, L. Physics-informed meshgraphnets (pi-mgns): Neural finite element solvers for non-stationary and nonlinear simulations on arbitrary meshes. *arXiv preprint arXiv:2402.10681*, 2024.

Yang, J., Dzanic, T., Petersen, B. K., Kudo, J., Mittal, K., Tomov, V., Camier, J.-S., Zhao, T., Zha, H., Kolev, T., Anderson, R., and Faissol, D. Reinforcement learning for adaptive mesh refinement. *26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2023a.

Yang, J., Mittal, K., Dzanic, T., Petrides, S., Keith, B., Petersen, B., Faissol, D., and Anderson, R. Multi-agent reinforcement learning for adaptive mesh refinement. *22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2023b.

Yano, M. and Darmofal, D. L. An optimization-based framework for anisotropic simplex mesh adaptation. *Journal of Computational Physics*, 231(22):7626–7649, 2012.

Zhang, Z., Wang, Y., Jimack, P. K., and Wang, H. Meshingnet: A new mesh generation method based on deep learning. In *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part III 20*, pp. 186–198. Springer, 2020.

Zhang, Z., Jimack, P. K., and Wang, H. MeshingNet3D: Efficient generation of adapted tetrahedral meshes for computational mechanics. *Advances in Engineering Software*, 157–158:103021, July 2021. ISSN 0965-9978. doi: 10.1016/j.advengsoft.2021.103021.

Zienkiewicz, O. C. and Zhu, J. Z. The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique. *International Journal for Numerical Methods in Engineering*, 33(7):1331–1364, 1992.

A. Message Passing Networks

MPNs iteratively update latent features for the graph nodes and edges over L message-passing steps. Using linear embeddings $\mathbf{x}_v^0 = \mathbf{x}_v$ and $\mathbf{x}_e^0 = \mathbf{x}_e$ of the initial node and edge features, each step l builds on the previous one, computing features

$$\mathbf{x}_e^{l+1} = f_{\mathcal{E}}^l(\mathbf{x}_v^l, \mathbf{x}_u^l, \mathbf{x}_e^l), \text{ with } e = (u, v),$$
$$\mathbf{x}_v^{l+1} = f_{\mathcal{V}}^l(\mathbf{x}_v^l, \bigoplus_{e=(v,u)\in\mathcal{E}} \mathbf{x}_e^{l+1}).$$

The operator $\bigoplus$ is a permutation-invariant aggregation, such as sum, mean, or maximum operator. Each $f_\cdot^l$ is parameterized as a learned MLP. The final output of the MPN is a learned representation $\mathbf{x}_v^L$ for each node $v \in \mathcal{V}$. We feed this representation into a decoder MLP to yield a scalar value per node that we train to approximate the target sizing field per mesh element. Each $f_\cdot^l$ is a learned function, usually parameterized as a simple MLP. The final output of the MPN is a learned representation $\mathbf{x}_v^L$ for each node $v \in \mathcal{V}$.

B. AMBER Training details

B.1. Replay Buffer

To prevent memory issues during training, we only add new meshes to the buffer if their total number of elements does not exceed a threshold of 1.2 times the largest expert mesh. Additionally, we count the number of generation steps each intermediate mesh has undergone, setting a maximum depth in the buffer to prevent the replay buffer from eventually filling up with too-fine meshes. We use this depth parameter to ensure an even distribution over mesh depths in the buffer by adding new meshes with a stratified mesh generation procedure. Here, we first select a target depth uniformly at random, and then choose a random mesh from this depth to select the next mesh to refine and add to the buffer.

B.2. Train Batch Construction

As different meshes may have very different sizes, we do not train on a fixed number of meshes in each batch. Instead, we define our batch size over the maximum number of nodes in a batch of graphs and fill up batches by iteratively adding a graph without replacement to this batch until the maximum number of nodes is reached. To prevent smaller graphs from being sampled disproportionally often, we count the number of training steps each graph has been used for, and prefer the least used graphs in the batch construction if possible.

C. Metrics

C.1. Density-Aware Chamfer Metric

This section describes our primary evaluation metric, the normalized Density-Aware Chamfer Distance (DCD), in more detail. The DCD is defined as a symmetric, exponentiated Chamfer distance with normalization factors to account for elements in one set that match multiple elements in the other.

Mathematically, we first define the closest point function

$$\hat{z}(z, S) = \arg\min_{w \in S} \|z - w\|_2.$$

Using this function, we define $n_{\hat{z}}$ as the number of matches that z has in the complementary set $\overline{S}$:

$$n_{\hat{z}} = |\{w \in \overline{S} \mid \hat{w}(w, S) = z\}|.$$

Given a point $z \in S$, let

$$C(S, z, \overline{S}) = 1 - \frac{1}{n_{\hat{z}}} e^{-\|z - \hat{z}(z, \overline{S})\|_2},$$

be the contribution function. The full metric is then defined as

$$d_{DCD}(S_1, S_2) = \frac{1}{2}\left(\frac{1}{|S_1|}\sum_{x \in S_1} C(S_1, x, S_2) + \frac{1}{|S_2|}\sum_{y \in S_2} C(S_2, y, S_1)\right).$$

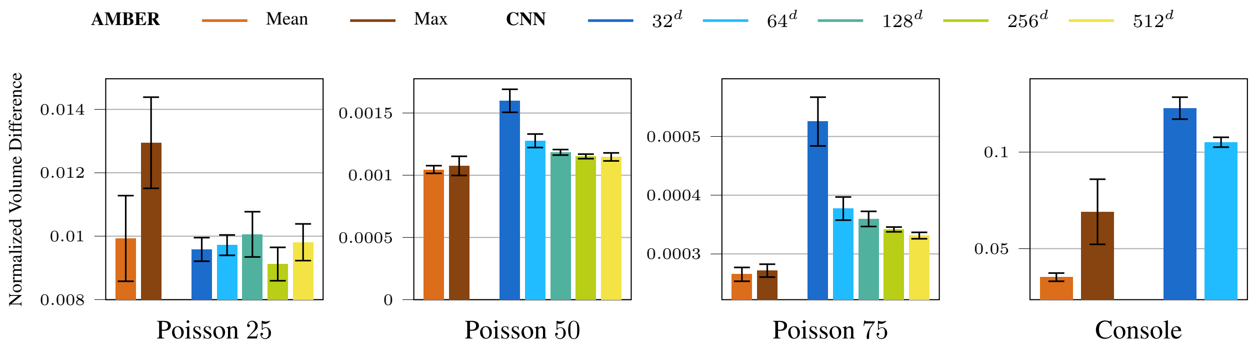

Figure 8: Normalized Volume Difference for AMBER and the CNN baseline for all considered tasks. The performance on this metric is largely consistent with that on the DCD across methods and tasks.

To accommodate the inaccuracies of the mesh generator, we report a normalized version of this metric as

$$d_{NDCD}(M^t, M^*) = \frac{d_{DCD}(\{p_{M_i^t}\}, \{p_{M_j^*}\}) - d_{DCD}(\{p_{M_i^0}\}, \{p_{M_j^*}\})}{d_{DCD}(\{p_{\hat{M}_i}\}, \{p_{M_j^*}\}) - d_{DCD}(\{p_{M_i^0}\}, \{p_{M_j^*}\})},$$

where $\hat{M}$ is the reconstruction of M^* that we obtain by directly feeding the sizing field of M^* into the mesh generator.

C.2. Volume Difference Metric

In addition to the normalized DCD of Section C.1, we evaluate the difference in element volumes between the generated mesh M^T and the expert M^*. Let $V(M_i^T)$ be the volume of element M_i^T, and $p_{M_i^T}$ be its midpoint. For an index i, we define $j^*(i)$ as the index of the element $M^*_{j^*(i)}$ which contains the midpoint $p_{M_i^T}$. Analogously, we set $j^T(i)$ as the index of the element $M^T_{j^T(i)}$ which contains the midpoint $p_{M_i^*}$. We then calculate the volume difference between two meshes as

$$d_{\text{VD}} = \sum_{\{i|M_i^T \subseteq M^T\}} |V(M_i^T) - V(M^*_{j^*(i)})| + \sum_{\{i|M_i^* \subseteq M^*\}} |V(M_i^*) - V(M^T_{j^T(i)})| \tag{2}$$

Intuitively, this metric numerically approximates the integrated absolute difference in element volumes across the generated and the expert mesh using the elements of both meshes as integration points.

Figure 8 evaluates the volume difference metric of Equation 2 for all tasks. While the results for the volume difference metric have more variance than the normalized DCD, they are consistent with that of Figure 3 and Figure 6.

D. Tasks and Data Collection

For Poisson's equation, we generate random L-shaped geometries, defined as $(0,1)^2 \backslash (p_0 \times (1,1))$, where the lower left corner p_0 is sampled from $U(0.2, 0.95)$ in x and y direction. For the load function, we employ a Gaussian Mixture Model with three components. The mean of each component is drawn from $U(0.1, 0.9)^2$ and re-sampled until it is within the geometry. We then independently draw diagonal covariances from a log-uniform distribution $\exp(U(\log(0.0001, 0.001)))$ and randomly rotate them to yield a full covariance matrix. Component weights are generated from $\exp(N(0,1)) + 1$ and subsequently normalized. For this task, we evaluate the load function at each face's midpoint and use it as a node feature. Additionally, we solve the equation for every intermediate mesh, and use the mean and standard deviation of the solution on each element's vertices as additional node features.

We generate the expert meshes using an error-based heuristic on a refinement threshold θ (Binev et al., 2004; Bangerth et al., 2012; Foucart et al., 2023), which marks all elements for refinement for which $\text{err}(M_i) > \theta \cdot \max_j \text{err}(M_j)$. The error is estimated by a heuristic

$$\text{err}(\Omega_i^t) = h^2 ||f||^2 + h|| \, [[\nabla u \cdot \mathbf{n}]] \, ||^2,$$

which assumes high errors in areas with high values of the load function f and with large gradients jumps on the element edges or facets, respectively. Once elements with a high error are marked, the mesh is processed by a remesher, which compute a conforming refined mesh using the red-green-blue refinement method (Carstensen, 2004). We apply Laplacian smoothing after each refinement step.

E. Network Architectures and Hyperparameters

All methods are trained on an NVIDIA A100 GPU, with each method given a computational budget of up to 48 hours. For mesh generation, we clip all predicted sizing field to the minimum sizing field of any expert mesh in the training data.

The MPN of AMBER consists of 10 separate message passing steps. Each message passing step uses separate 2-layer MLPs and LeakyReLU activations for its node and edge updates. We independently apply residual connections (He et al., 2016) and layer normalization (Ba et al., 2016) to the node and edge updates. The final node features are fed into a 2-layer MLP decoder. All MLPs use a latent dimension of 64. We provide an overview of AMBER hyperparameters in Table 1.

For the CNN baseline, we use a U-Net (Ronneberger et al., 2015) architecture with 32 initial channels and 4 down- and up-convolution blocks. Each convolution block consists of 2 convolutions with a kernel size of 3, followed by batch normalization and a ReLU activation function. After each down-convolution, we use max-pooling with a kernel size and stride of 2 to halve the image resolution, and double the number of channels. This process is reversed for the up-convolutions, and we add skip connections between the respective depths. We use 2D and 3D convolutions, batch normalization and pooling operations for the 2D and 3D tasks, respectively.

Table 1: AMBER hyperparameters and experiment configurations

Parameter	Value
Optimizer	ADAM
Learning rate	$3.0e$-4
Latent dimension	64
MPN aggregation function	mean
MPN steps	10
MPN activation function	Leaky ReLU
MPN edge dropout	0.1
MLP layers	2
Maximum replay buffer depth	5
Maximum buffer size	1 000 meshes
Training steps	128 000
Buffer addition frequency	every 8 training steps
AMBER batch size	400 000 graph nodes
AMBER inference steps	5

F. Extended Results

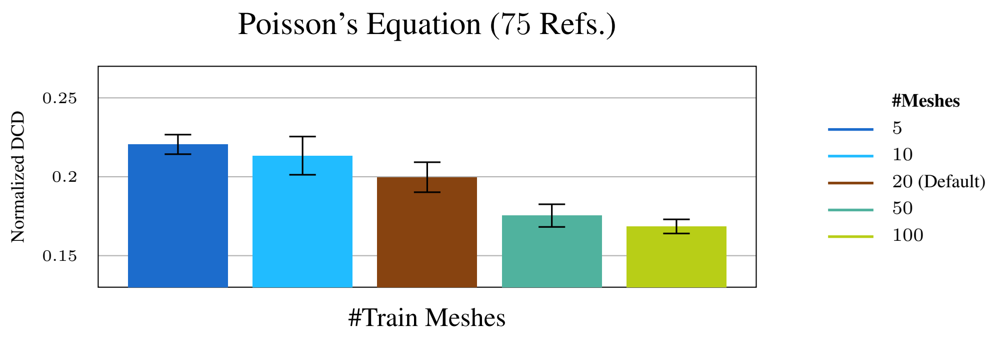

Figure 9: Normalized DCD for AMBER (Max) on Poisson's Equation with 75 refinement steps for different numbers of training meshes. A larger number of training meshes consistently improves performance, yet even a training set size of 5 yields strong generalization performance to unseen meshes.

We explore the data efficiency of AMBER in Figure 9. While the approach continuously improves its predictions when provided with more training data, as few as 5 training meshes and corresponding geometries are sufficient for accurate mesh generation. This generalization capabilities are likely a results of the local, per-node MSE loss and the MPN network architecture. Figure 10 explores design choices of AMBER's training scheme. Both input normalization and transforming the output predictions improve performance. The stratified sampling of new training data for AMBER's replay buffer improves performance, presumably because this ensures an even distribution of training data across mesh generation iterations, whereas randomly adding new meshes eventually under-represents the early meshes in the training data.

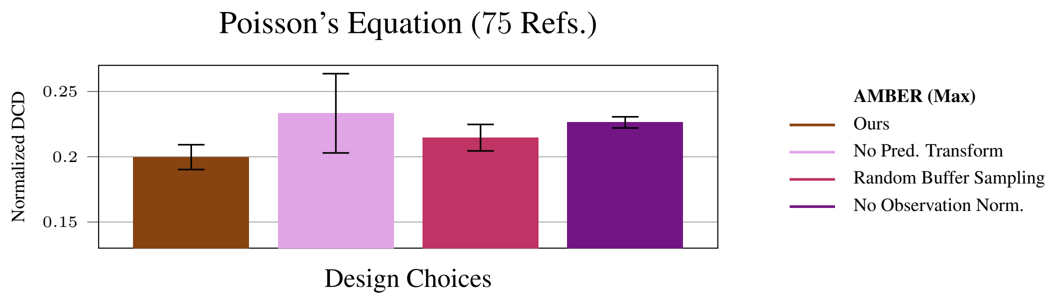

Figure 10: Normalized DCD for AMBER (Max) on Poisson's Equation with 75 refinement steps for different algorithmic design choices. Either Omitting the softplus output transformation for the sizing field prediction or not normalizing the input features yields worse results. Randomly adding intermediate training meshes to the replay buffer instead of adding the same amount of intermediate meshes for each mesh generation step decreases performance.

F.1. Visualizations

We present the qualitative results of all methods applied to all tasks. Figure 11 displays the results for AMBER (Mean), Figure 12 shows the results for AMBER (Max), and Figure 13 showcases the predictions for the CNN baseline.

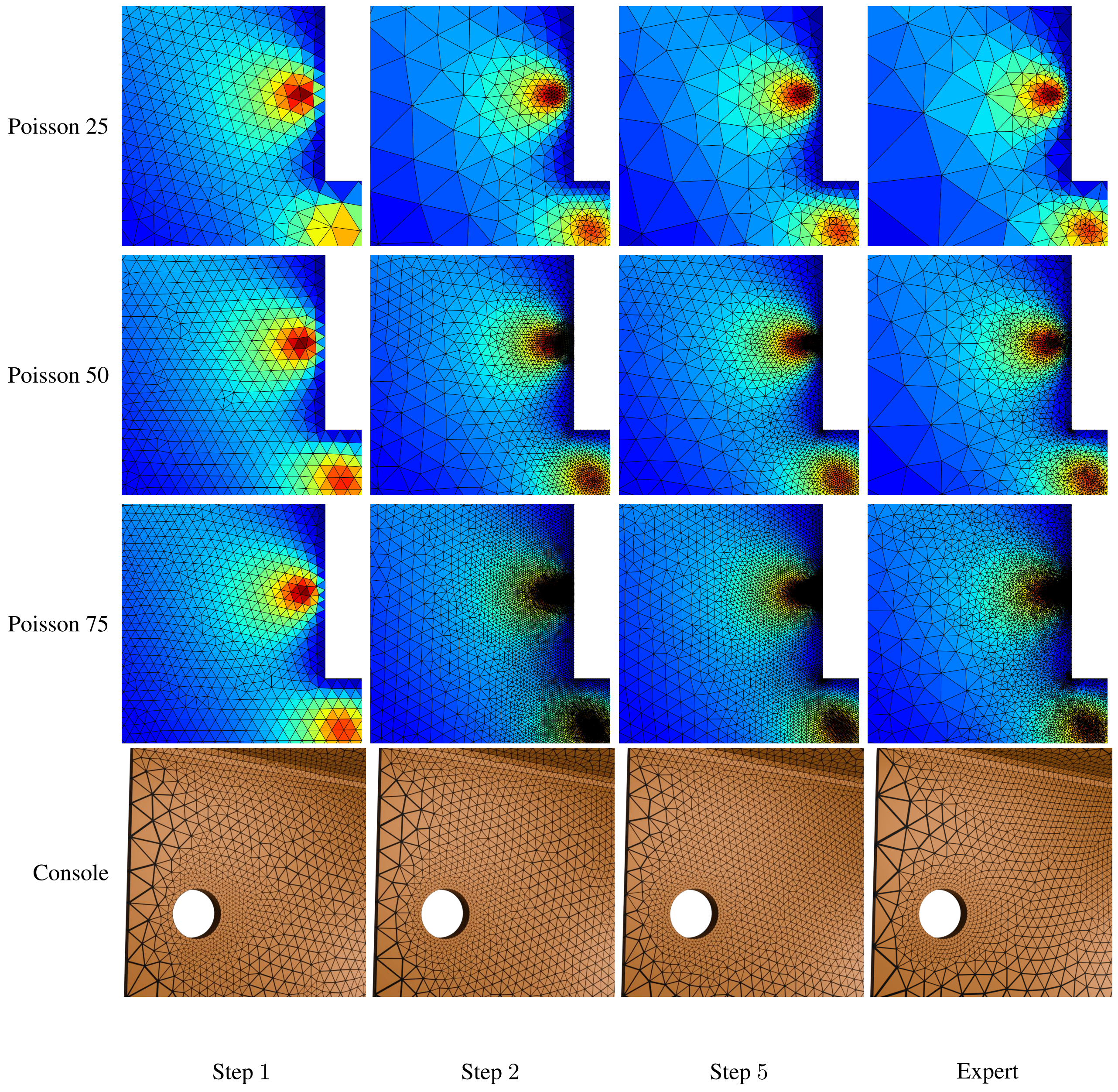

Figure 11: Mesh generation steps of AMBER (Mean) for all tasks. **Row 1 to 3:** Poisson's Equation with 25/50/75 expert refinement steps. **Row 4:** Console task. AMBER (Mean) converges quickly to meshes that match the expert (displayed in the last column).

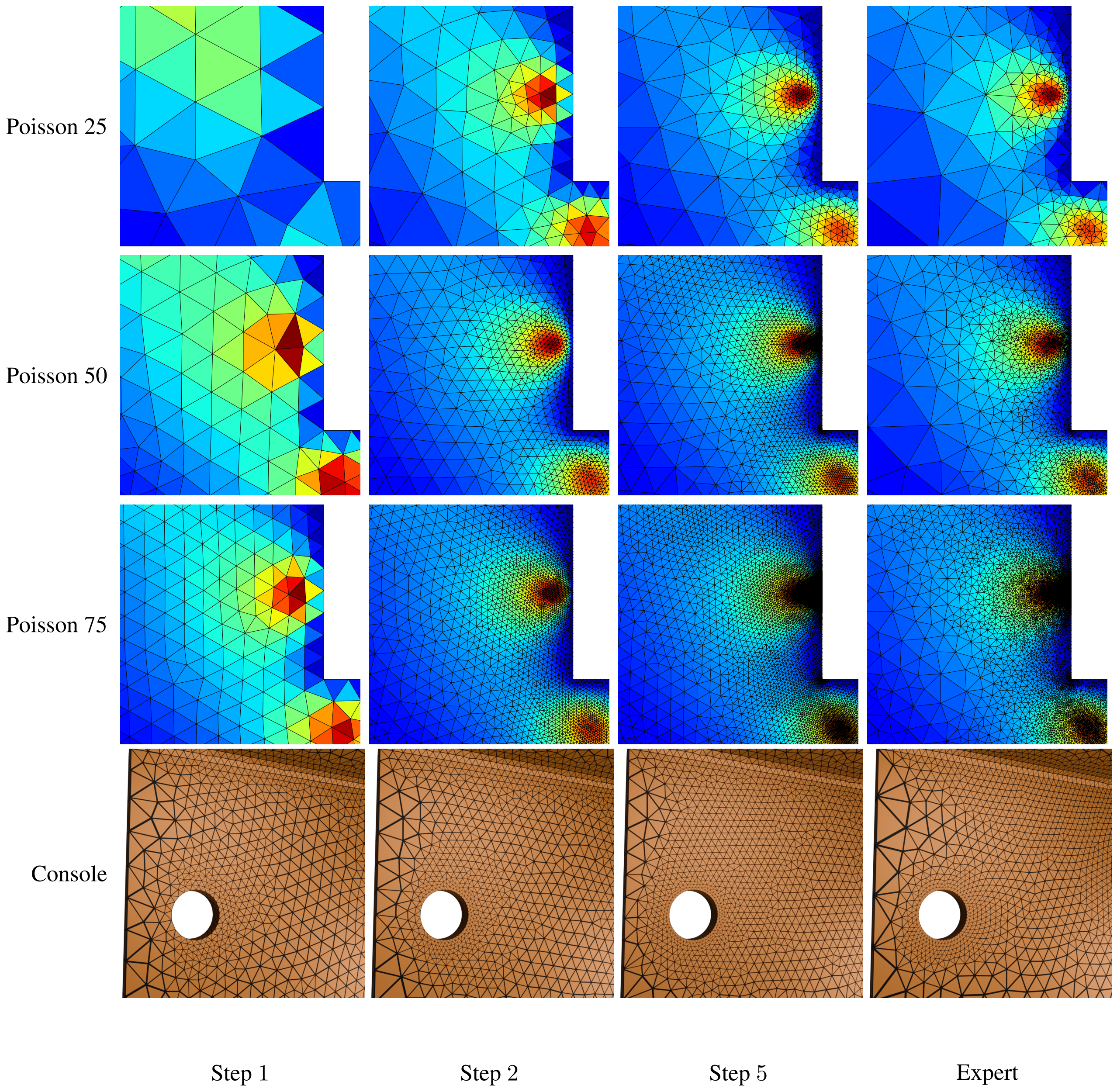

Figure 12: Mesh generation steps of AMBER (Max) for all tasks. **Row 1 to 3:** Poisson's Equation with 25/50/75 expert refinement steps. **Row 4:** Console task. AMBER (Max) creates generates more conservative meshes, yet still closely matches the expert after 5 refinement steps.

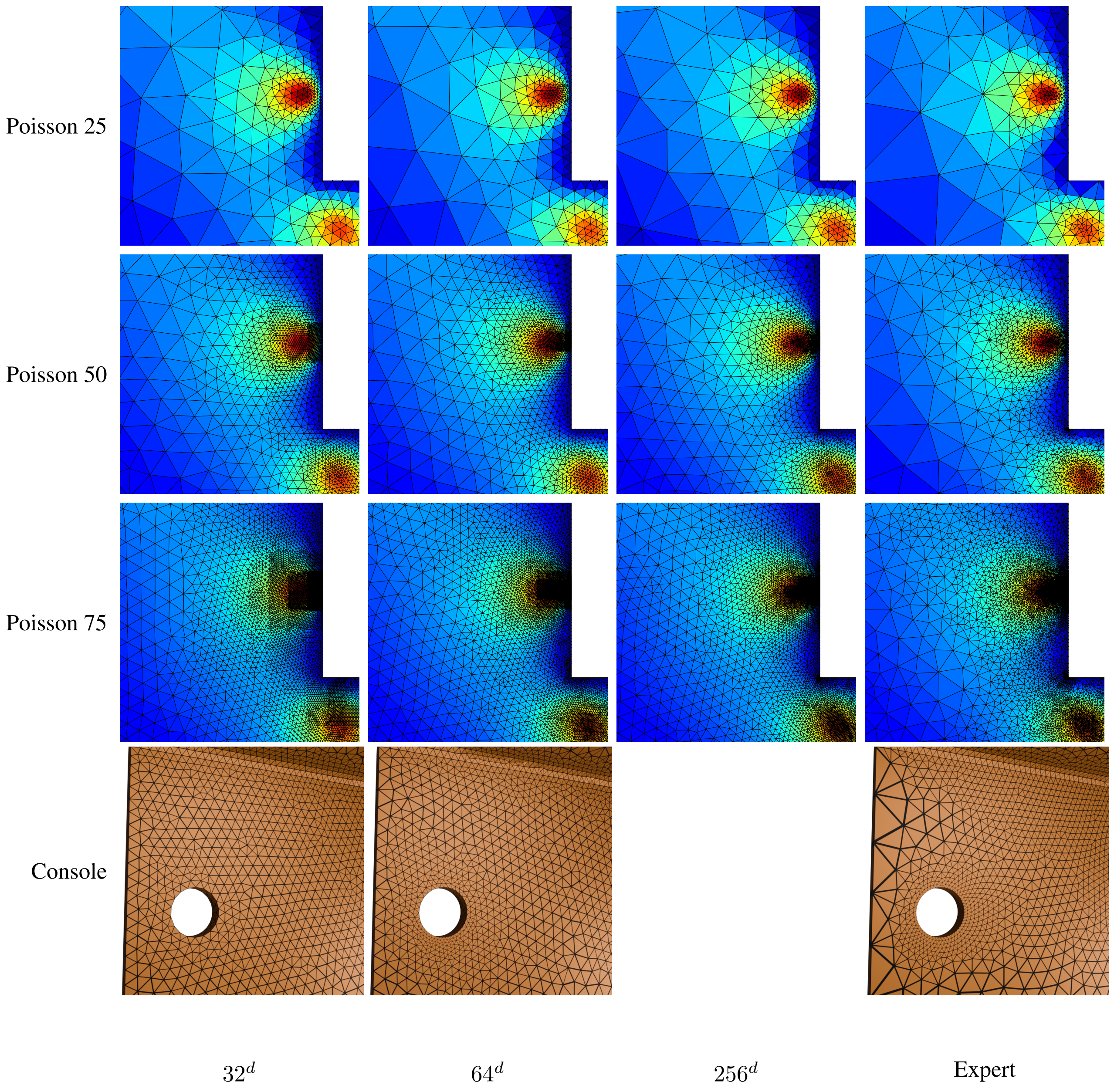

Figure 13: Meshes generated by the CNN baseline with varying input resolution on all environments. **Row 1 to 3:** Poisson's Equation with 25/50/75 expert refinement steps. **Row 4:** Console task. Lower image resolutions lead to square-like artifacts in the output mesh which are not present in the expert meshes. The $256 \times 256 \times 256$ CNN baseline on the console task could not be computed due to excessive memory usage.