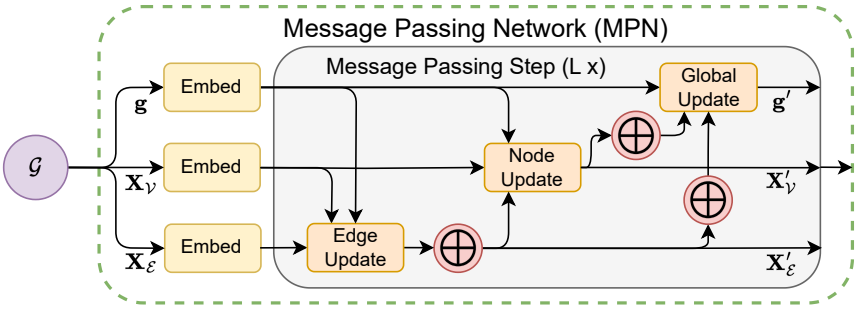

Figure 7: Detailed schematic of the MPN architecture employed by ASMR. Given a graph $\mathcal{G}$, the global features $\mathbf{g}$, node features $\mathbf{X}_\mathcal{V}$, and edge features $\mathbf{X}_\mathcal{E}$ are linearly embedded into separate latent spaces. From here, L message passing steps are performed. The resulting latent features per node are interpreted as a local observation encoding and can be given to an RL policy or value function.

A Message Passing Network Architecture

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}_\mathcal{V}, \mathbf{X}_\mathcal{E}, \mathbf{g})$, Message Passing Networks (MPN) [32, 37, 38] are GNN consisting of L *Message Passing Steps*. Each step l receives the output of the previous step and updates the features $\mathbf{X}_\mathcal{V}$, $\mathbf{X}_\mathcal{E}$ for all nodes $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$, as well as globals $\mathbf{g}$. Using linear embeddings $\mathbf{x}_v^0$, $\mathbf{x}_e^0$, and $\mathbf{g}^0$ of the initial node, edge, and global features, the l-th step is given as

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

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

The operator $\oplus$ is a permutation-invariant aggregation such as a sum, max, or mean operator. Each $f_\cdot^l$ is a learned function that we generally parameterize as a simple MLP. The network's final output is a learned representation $\mathbf{x}_v^L$ for each node $v \in \mathcal{V}$. Figure 7 provides a schematic overview of the full MPN architecture.

B Systems of Equations

In its most general form, the FEM is used to approximate the solution $\boldsymbol{u}(\boldsymbol{x})$ for the set of test functions $\boldsymbol{v}$, which satisfies the weak formulation $\forall \boldsymbol{x} : \forall \boldsymbol{v}(\boldsymbol{x}) : a(\boldsymbol{u}(\boldsymbol{x}), \boldsymbol{v}(\boldsymbol{x})) = l(\boldsymbol{v}(\boldsymbol{x}))$ of the underlying system of equations. In the following, we describe the specific equations and boundary conditions used for our experiments.

B.1 Laplace's Equation

Let Ω be a domain with an inner boundary $\partial\Omega_0$ and an outer boundary $\partial\Omega_1$. We seek a solution $u(\boldsymbol{x})$ that satisfies the weak formulation of the Laplace Equation

$$\int_\Omega \nabla u(\boldsymbol{x}) \cdot \nabla v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0$$

for all test functions $v(\boldsymbol{x})$. Additionally, the solution has to satisfy the Dirichlet boundary conditions

$$u(\boldsymbol{x}) = 0, \boldsymbol{x} \in \partial\Omega_0 \quad \text{and} \quad u(\boldsymbol{x}) = 1, \boldsymbol{x} \in \partial\Omega_1.$$

We use a unit square $(0, 1)^2$ for the outer boundary $\partial\Omega_0$ of the domain and add a randomly sampled square hole, whose borders are considered to be the inner boundary $\partial\Omega_1$. The size of the hole is sampled from the uniform distribution $U(0.05, 0.25)^2$, and its mean position is sampled from $U(0.2, 0.8)^2$. We add the closest distance to the inner boundary as an additional node feature.

B.2 Poisson's Equation

The weak formulation of the considered Poisson problem is given as

$$\int_\Omega \nabla u(\boldsymbol{x}) \cdot \nabla v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \int_\Omega f(\boldsymbol{x}) v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \quad \forall v.$$

Here, $f(\boldsymbol{x}) : \Omega \to \mathbb{R}$ denotes the load function and $v(\boldsymbol{x})$ the test function. In addition to the weak formulation, the solution must be zero on the boundary $\partial\Omega$ of the domain Ω. We model Poisson's Equation on L-shaped domains Ω, using a rectangular cutoff whose lower left corner is sampled from $p_0 \sim U(0.2, 0.95)^2$, resulting in a domain $\Omega = (0,1)^2 \backslash (p_0 \times (1,1))$. On this domain, we sample a Gaussian Mixture Model with 3 components. The mean of each component is sampled from $U(0.1, 0.9)^2$, using rejection sampling to ensure that all means lie within the domain. The components' covariances are determined by first drawing diagonal covariances, where each dimension is drawn independently from a log-uniform distribution $\exp(U(\log(0.0003, 0.003)))$. The diagonal covariances are then rotated by a random angle in $U(0, 180)$ to produce Gaussians with a full covariance matrix. The component weights are drawn from the distribution $\exp(N(0,1)) + 1$ and subsequently normalized, where the 1 in the end is used to ensure that all components have relevant weight. Here, the evaluation of the load function f at the respective face midpoint is added as a node feature.

B.3 Stokes flow

Let $\boldsymbol{u}(\boldsymbol{x})$ be the velocity field and $p(\boldsymbol{x})$ the pressure field. We consider a Stokes flow of a fluid through a channel. Therefore, we seek a solution u and p, which satisfy the weak formulation of the Stokes flow without a forcing term

$$\nu \int_\Omega \nabla \boldsymbol{v} \cdot \nabla \boldsymbol{u} \, \mathrm{d}\boldsymbol{x} - \int_\Omega (\nabla \cdot \boldsymbol{v}) p \, \mathrm{d}\boldsymbol{x} = 0 \quad \forall \boldsymbol{v}$$

$$\int_\Omega (\nabla \cdot \boldsymbol{u}) q \, \mathrm{d}\boldsymbol{x} = 0 \quad \forall q,$$

wherein $\boldsymbol{v}(\boldsymbol{x})$ and $q(\boldsymbol{x})$ denote the test functions [88]. In addition, we assume a no-slip condition $\boldsymbol{u} = \boldsymbol{0}$ at both the top and bottom of the channel, and an inlet-profile defined as

$$\boldsymbol{u}(x = 0, y) = u_\mathrm{P} y(1 - y) + \sin(\varphi + 2\pi y).$$

At the outlet, the gradient of velocity $\nabla \boldsymbol{u} = \boldsymbol{0}$ is set to zero. For stability purposes, we use P_1/P_2 Taylor-Hood-elements, i.e., quadratic shape functions for the velocity and linear shape functions for the pressure [87]. We sample the quadratic part u_P of the velocity inlet from a log-uniform distribution $\exp(U(\log(0.5, 2)))$. The domains are a class of trapezoids that we derive from the unit square by randomly choosing 2 of the 4 vertices and adding a random inward-facing y offset drawn from $U(0, 0.45)$ to these points. The resulting trapezoid is subsequently normalized to lie in $(0,1)^2$. We add the parabolic part of the inlet velocity of the current PDE as a global feature. We optimize the meshes for the prediction of the velocity norm and use linear shape functions for the numerical error approximation in 1 for simplicity.

B.4 Linear Elasticity

We are looking for the steady-state deformation of a solid under stress, due to displacements at the boundary of the part $\partial\Omega$. Here, we are interested in both the norm of the deformation and the norm of the stress. The weak formulation of the considered problem on the domain Ω without body forces is given as [89]

$$\int_\Omega \boldsymbol{\sigma}(\boldsymbol{\varepsilon}(\boldsymbol{u})) : \boldsymbol{\varepsilon}(\boldsymbol{v}) \, \mathrm{d}\boldsymbol{x} = 0.$$

Here, $\boldsymbol{u}(\boldsymbol{x})$ is the displacement field, $\boldsymbol{v}(\boldsymbol{x})$ is the test function, and $\boldsymbol{\varepsilon}(\boldsymbol{u}) = \frac{1}{2}(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^\top)$ is the strain tensor. $\boldsymbol{\sigma}(\boldsymbol{\varepsilon})$ is the stress tensor, which is given as $\boldsymbol{\sigma}(\boldsymbol{\varepsilon}) = 2\mu\boldsymbol{\varepsilon} + \lambda \mathrm{tr}(\boldsymbol{\varepsilon})\boldsymbol{I}$ in a linear-elastic and isotropic case. The Lamé parameters $\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}$ and $\mu = \frac{E}{2(1+\nu)}$ can be calculated with the problem specific Young's modulus $E = 1$ and the Poisson ratio $\nu = 0.3$. The displacement

$\boldsymbol{u}(x = 0, y) = \boldsymbol{u}_0$ on the left side of the boundary is specified by a task-dependent parameter $\boldsymbol{u}_0$, whereas the displacement $\boldsymbol{u}(x = L, y) = 0$ is set to zero on the right boundary. The stress $\boldsymbol{\sigma} \cdot \boldsymbol{n} = \mathbf{0}$ is zero normal to the boundary at both the top and bottom of the part. We use the same class of L-shaped domains as in the Poisson problem in Section B.2 and set u_P by drawing a random angle from $U[0, \pi]$ to pull on the domain from different angles, and add random magnitude from $U(0.2, 0.8)$. We add the task-dependent displacement u_P as a global feature. We are interested in the norm of the displacement field u and the resulting Von-Mises stress, giving us a 2-dimensional objective. We weight both dimensions equally in the reward.

B.5 Non-stationary Heat Diffusion

We consider a non-stationary thermal diffusion problem defined by the weak formulation

$$\int_\Omega \frac{\partial u}{\partial t} \, \mathrm{d}\boldsymbol{x} + \int_\Omega a \nabla u \cdot \nabla v \, \mathrm{d}\boldsymbol{x} = \int_\Omega f v \, \mathrm{d}\boldsymbol{x} \quad \forall v,$$

wherein u denotes the temperature, v the test function, a the thermal diffusivity and f a heat distribution, given as

$$f = q \exp\left(-100 \left((x - x_p(\tau)) + (y - y_p(\tau))\right)\right).$$

The position of the maximum heat entry $\boldsymbol{p}_\tau(\tau) = (x_p(\tau), y_p(\tau))$ is changing over time, while its magnitude is scaled by a factor q. The temperature $u \in \partial\Omega$ is set to zero on all boundaries. For the time-integration, the implicit euler method is applied. We use a total of $\tau_{\text{max}} = 20$ time steps in $\{0.5, \ldots, 10\}$, a scaling factor of $q = 1000$ and a diffusivity $a = 0.001$. The position of the heat source at step τ is linearly interpolated as $\boldsymbol{p}_\tau = \boldsymbol{p}_0 + \frac{\tau}{\tau_{\text{max}}}(\boldsymbol{p}_{\tau_{\text{max}}} - \boldsymbol{p}_0)$, where the start and goal positions $\boldsymbol{p}_0$ and $\boldsymbol{p}_{\tau_{\text{max}}}$ are randomly drawn from the domain. To create our domains, we start with 10 points that are equidistantly placed on a circle with center $(0.5, 0.5)$ and radius 0.4. Each point is distorted by a random value drawn from $U(-0.2, 0.2)^2$. We then normalize the resulting points to be in $(0, 1)^2$ and calculate the convex hull. The result is a family of convex polygons with up to 10 vertices. We measure the error of the final simulation step, and provide the distance to the start and end position of the heat source as additional node features.

C Further Experiments

C.1 Experiment Details

All experiments are repeated for $n = 10$ random seeds with randomized PDEs and network parameters. All PDEs are normalized to be in $(0, 1)^2$. We train all policies on 100 training PDEs and evaluate the resulting final policies on 100 different evaluation PDEs that we keep consistent across random seeds for better comparability. All experiments are run for up to 3 days on 8 cores of an Intel Xeon Platinum 8358 CPU.

In terms of total compute, we train 4 different learned methods, namely the 3 RL baselines and our method, on 5 separate tasks. Each experiment is repeated for 10 different target mesh resolutions and 10 repetitions, resulting in $5 \cdot 4 \cdot 10 \cdot 10 = 2000$ main experiments, each of which is run for up to 3 days. Additionally, we use a similar amount of compute for the combined ablations, preliminary experiments and error-based heuristics.

For practical purposes, we add an element threshold β_{max} in our environments, and terminate an episode with a large negative reward when this threshold is exceeded.

C.2 Maximum Reward

Equation 2 scales the reduction in error of each element by its area. This modification encourages the policy to focus on smaller elements, effectively shifting the objective from an reduction in average error across the mesh to a minimization of error densities.

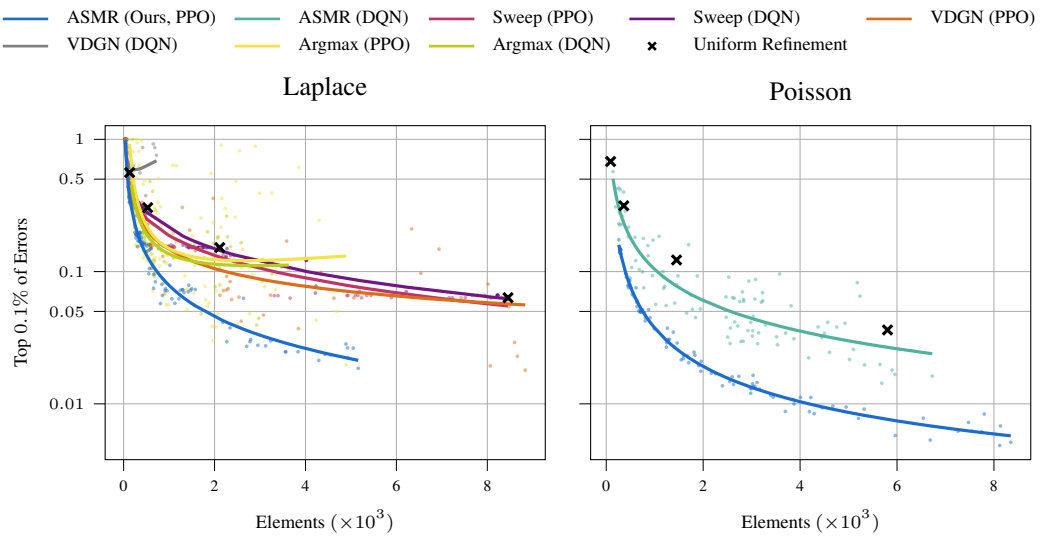

Figure 8: Pareto plot of normalized Top $0.1\,\%$ of errors and number of final mesh elements for (Left) PPO and DQN for all RL baselines on the Laplace equation and (Right) PPO and DQN for ASMR on the Poisson task. PPO generally results in better performance, with the exception of *Argmax*, which is slightly better when using DQN. The performance of *Argmax* is likely a result of the comparatively simple action space of only refining a single element per environment step.

An alternate way to phrase this objective is to make the reward depend on the reduction in maximum error per element. For this, we modify the error estimate per element of Equation 1 to read

$$\hat{\text{err}}(\Omega_i^t) \approx \max_{\Omega_m^* \subseteq \Omega_i^t} \left| u_{\Omega^*}(p_{\Omega_m^*}) - u_{\Omega^t}(p_{\Omega_m^*}) \right|,$$

and subsequently drop the area scaling and replace the sum in Equation 2 with a maximum, i.e.,

$$\mathbf{r}'(\Omega_i^t) := \left(\text{err}(\Omega_i^t) - \max_j \mathbf{M}_{ij}^t \text{err}(\Omega_j^{t+1}) \right) - \alpha \left(\sum_j \mathbf{M}_{ij}^t - 1 \right).$$

While conceptually simpler than our reward formulation, evaluating the decrease in maximum error optimizes only this objective, and may result in degenerate meshes when looking at the overall error. Figure 6 compares this alternate reward formulation to that of ASMR. We find that it performs well when considering the Top $0.1\,\%$ of errors, but that it fails to generate meshes with a low overall error.

D Extended Results

D.1 Baseline Algorithms

The left of Figure 8 shows results for the RL baseline methods for both PPO and DQN as the RL backbone. We find that PPO performs better for *VDGN* and *Sweep*, while DQN seems to be better for *Argmax*. For the PPO variant of *VDGN*, we factorize the value function instead of the Q-function, i.e., we define the value function of the full mesh as the sum of value functions of the individual mesh elements.

D.2 Ablations.

Proximal Policy Optimization and Deep Q-Networks. The right of Figure 8 compares ASMR with PPO and DQN as the RL algorithms. We find that a using a mean instead of a sum for the agent mapping of the targets of the Q-values increases training stability, and thus use it for the DQN experiments with ASMR. However, the PPO variant still performs significantly better than the DQN one. As such, we use PPO as the RL algorithm for ASMR for all other experiments.

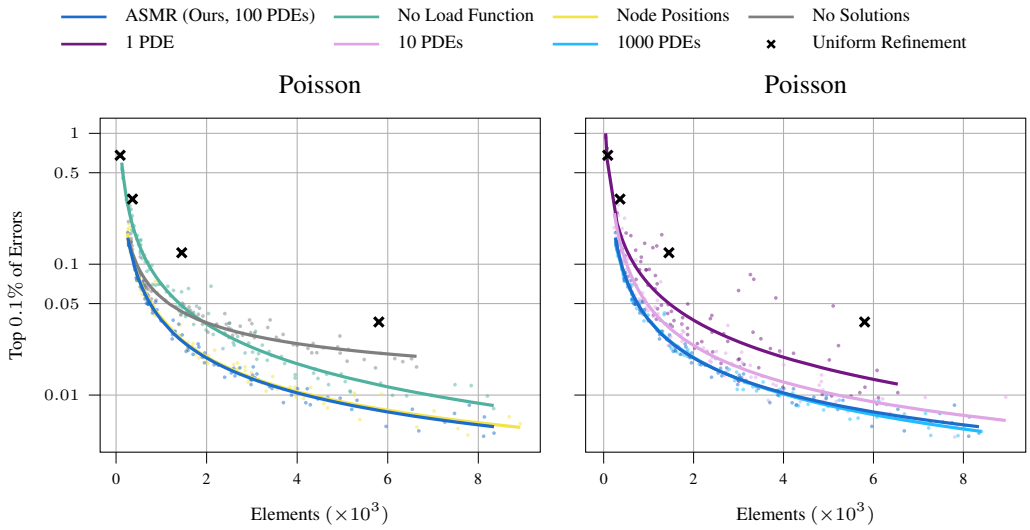

Figure 9: Pareto plot of normalized Top $0.1\,\%$ of errors and number of final mesh elements for Poisson's Equation. (Left) Omitting either the solution or evaluation of the load function per element significantly decreases the performance of our approach. Adding explicit node positions is slightly detrimental to the performance, likely because it causes the observation to no longer be equivariant to e.g., reflection. (Right) Performance is reduced for fewer training PDEs, but stabilizes around 100 PDEs. Interestingly, ASMR achieves acceptable performance when using a single training PDE, which is likely a result of our spatial problem formulation.

Node Features. ASMR utilizes both task-dependent information, such as the evaluation of the load function for Poisson's Equation, and the local solution $u(x)$ per mesh element as part of its observation graph. Here, we experiment how the performance is affected if either of these features is omitted. Additionally, we consider a variant where we include explicit (x, y) positions of each element midpoint as node features. The results are shown on the left of Figure 9 We find that both the task-dependent features and the solution are important for the performance of our approach. *Not* adding positional features slightly improves performance, presumably because the features assign a fixed position to each mesh element, causing the observation graph to no longer be equivariant to rotation, translation and reflection. Interestingly, ASMR provides reasonable refinements even without solution information, suggesting that the RL algorithm is able to detect relevant regions of the PDE from just an encoding of the domain and the boundary conditions.

Number of Training PDEs Since calculating the fine-grained reference Ω^* is slow for large meshes and complex tasks, we want to minimize the number of unique PDEs that we need during training. We use 100 PDEs in our other experiments, and additionally visualize results for 1, 10 and 1000 training PDEs on the right of Figure 9. We find that fewer than 100 PDEs lead to less stable and reliable results, and that there is only a minor advantage in using 1000 PDEs compared to our 100. Noticeably, a single training PDE results in acceptable refinements in most cases, which hints at significant generalization capabilities that are likely due to our spatial treatment of the underlying task.

D.3 Runtime comparison.

We compare the wallclock-time of our approach with that of directly computing the fine-grained uniform mesh Ω^*. For our approach, we measure the cumulative time of creating an initial coarse mesh, and then iteratively solving the problem on this mesh, computing the resulting observation graph, feeding the observation graph to the policy to obtain actions, and using these actions to refine the mesh a total of $T = 6$ times. For the uniform mesh, we simply measure the time it takes to refine the coarse mesh 6 times and to subsequently solve the problem on the resulting mesh. We use a single 8-Core AMD Ryzen 7 3700X Processor for all measurements. Figure 10 shows the results for all considered tasks. We find that our approach is always significantly faster than computing the fine-grained mesh despite the comparatively large computational overhead. Further, the final

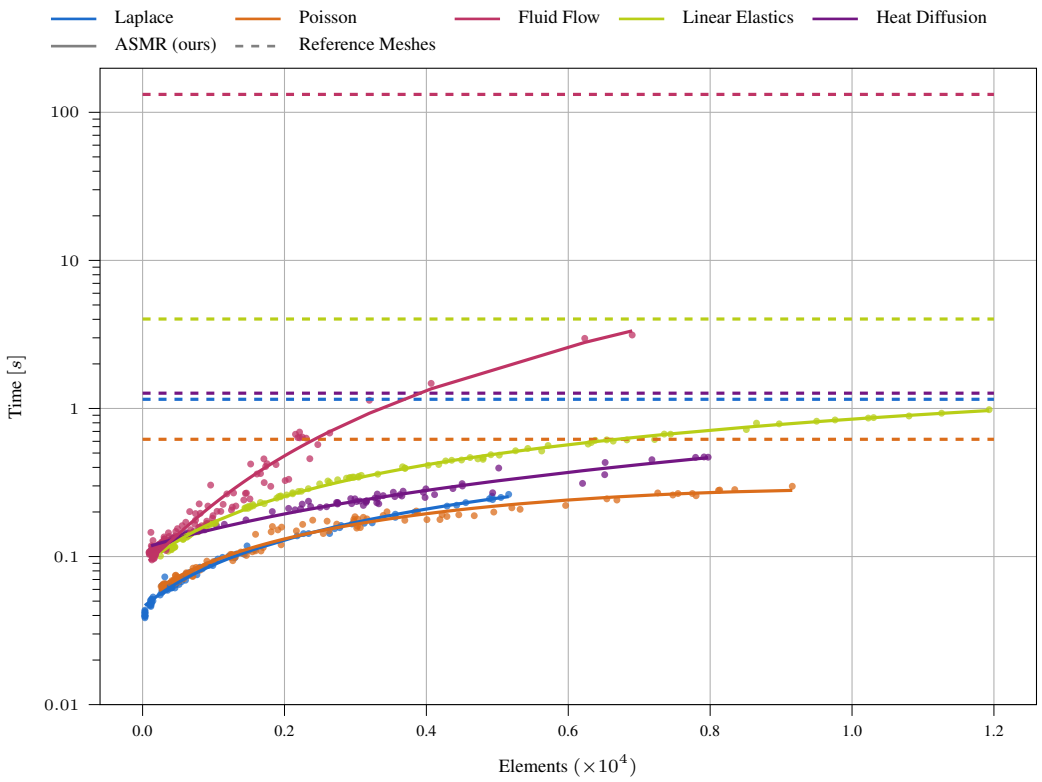

Figure 10: Wallclock-time (in seconds) of ASMR for different numbers of elements compared to the uniform reference Ω^*. The dotted lines represent the wallclock-time of creating and solving Ω^*, while the full lines represent a quadratic regression of the wallclock-time of our method for different numbers of final elements. On average, Ω^* contains about 10^5 elements, with concrete numbers varying depending on the domain. Our approach is significantly faster than the reference for all tasks.

resolution of the computed meshes trades off the wallclock-time of the method, meaning that ASMR can be trained to generate coarser or finer meshes depending on task-specific computational budgets. Notably, for the Navier Stokes equations, which use P_1/P_2 Taylor-Hood-elements, our method is more than 30 times faster than Ω^* even for highly refined final meshes.

D.4 Generalization capabilities.

We qualitatively test the generalization capabilities of our method by employing it for different Poisson tasks. For this, we consider the 4 different domain types used in throughout the main experiments, plus a simple rectangular domain $\Omega = (0, 1)^2$. We sample 3 random domains from each class, and use Gaussian Mixture Model load functions with 1, 3 and 5 components respectively. Figure 11 shows refinements of an ASMR policy with $\alpha = 0.0075$ for the resulting 3×5 problems. We find that ASMR generalizes across all domains and load functions, which is likely a result of the Swarm RL setting, where each mesh element is governed by its own agent.

D.5 Target Mesh Resolutions

All RL methods use some parameter to control the number of target elements of the final refined mesh. ASMR and *VDGN* use an element penalty α, *Sweep* uses a budget N_{max}, and *Argmax* different numbers of rollout steps T. We visualize the effect of different target resolutions in Figure 12. The results indicate that ASMR provides meshes with consistent numbers of elements for a given target resolution, while the other RL methods produce meshes with different numbers of elements for the same or similar target resolutions. The concrete target resolution parameters used for our experiments are found in Table 1.

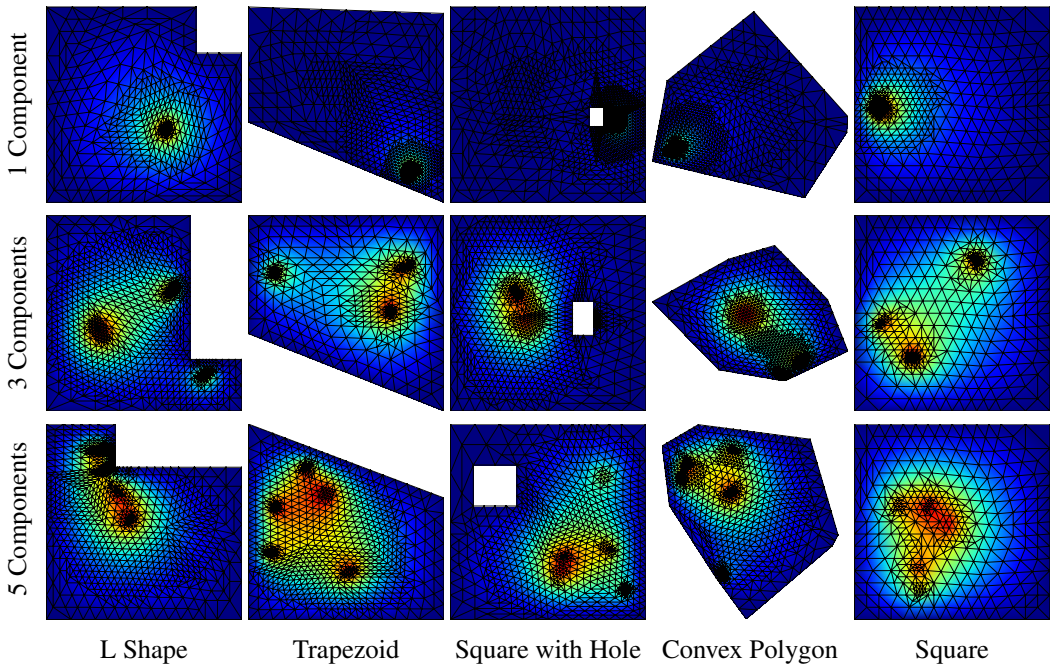

Figure 11: Final refined meshes of ASMR for different domains and Poisson's equation with a Gaussian Mixture Model load with 1, 3 and 5 components. Even though ASMR is only trained on L-shaped domains with 3 components in the load function, it generalizes to different domains as well as more and less complex loads.

D.6 Alternate Error Metrics

Section 5 evaluates all approaches on the normalized average of the Top $0.1\,\%$ of errors of the numerical integration. The left of Figure 13 instead shows the error of the Top $5\,\%$ of integration points for the Poisson task. We find that the results similar to that of Figure 4, i.e., that approaches that perform well on the highest $0.1\,\%$ of errors also generally show good results on a larger percentage of errors.

Using the notation of Section 3, we can also measure the mean remaining error of the final refinement Ω^T compared to the reference Ω^* via its normalized mean error, i.e., as

$$\frac{\sum_r \text{Area}(\Omega^*_m) \left| u_{\Omega^*}(p_{\Omega^*_m}) - u_{\Omega^T}(p_{\Omega^*_m}) \right|}{\sum_r \text{Area}(\Omega^*_m) \left| u_{\Omega^*}(p_{\Omega^*_m}) - u_{\Omega^0}(p_{\Omega^*_m}) \right|}.$$

This metric can be seen as a Top $100\,\%$ error, except that each integration point is additionally scaled with the area of its corresponding reference element. The right of Figure 13 shows results for this linear error for the Poisson equation. Figure 14 displays results for the Laplace equation and the Stokes flow, and results for the linear elasticity and heat diffusion tasks are shown in Figure 15. The general trends for the linear errors are consistent with the Top $0.1\,\%$ errors of Section 5. Since this metric focuses on the mean instead of the maximum error, the *Local Oracle* performs better than on the Top $0.1\,\%$ metric, reaching or surpassing the performance of the *Local Maximum Oracle* on most tasks. Still, both local oracle heuristics seem to perform different and in some cases sub-optimal, likely due to global dependencies in the PDEs and due to the remesher enforcing a conforming solution. ASMR provides accurate refinements that optimize both the Top $0.1\,\%$ and the remaining linear error on most tasks. For e.g., the linear elasticity task, the refinements provided by ASMR significantly outperform both error-based oracle heuristics, likely because this task has global dependencies that can be learned by the RL algorithm, but are ignored by the local heuristics. For the Stokes flow, a simple uniform refinement seems to be sufficient to reduce the mean remaining error, suggesting that the distribution of total error mass is comparatively homogeneous. The behavior of different variants of our reward function on this metric are provided in Figure 6.

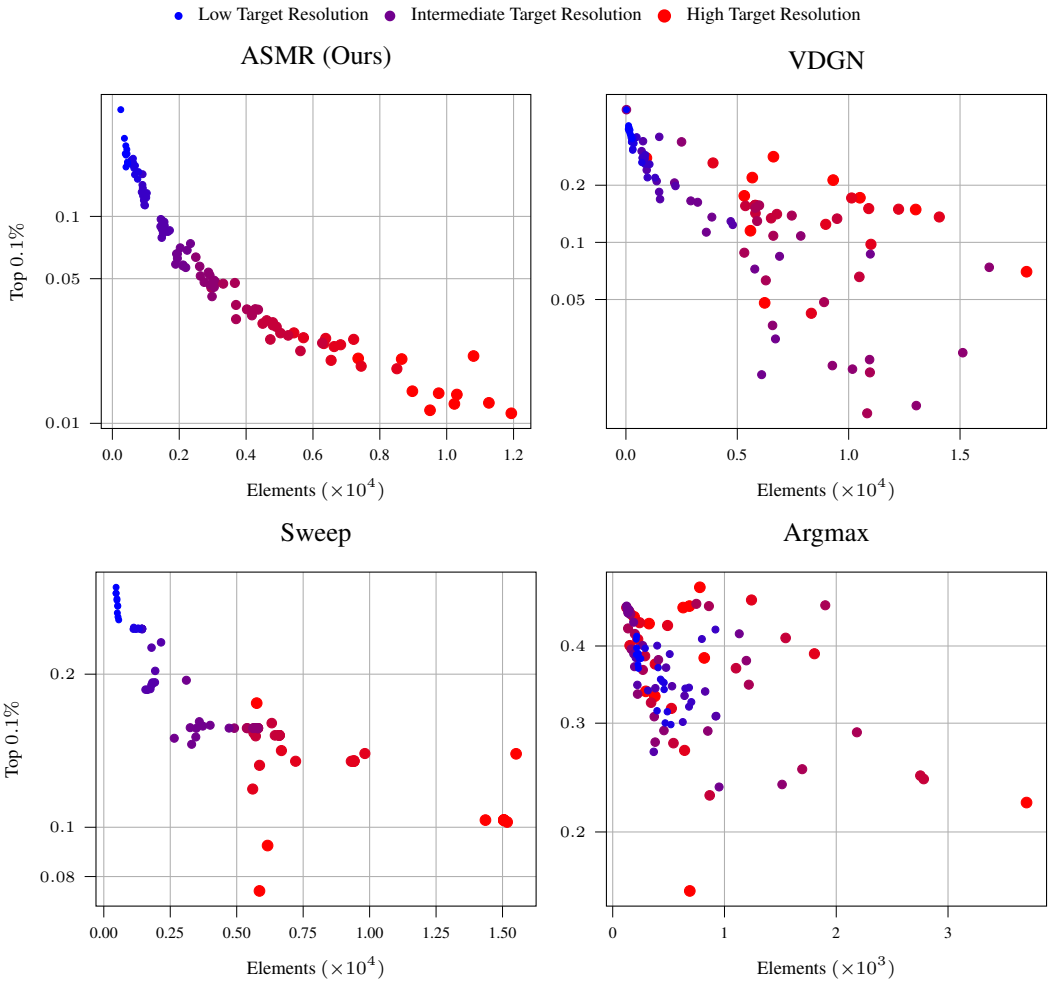

Figure 12: Pareto plot of normalized Top $0.1\,\%$ of errors and number of final mesh elements for the linear elasticity task for all RL methods. Small blue dots indicate a random seed trained on a coarse target mesh resolution, which corresponds to large element penalties α for ASMR and *VDGN*, a small budget N_{max} for *Sweep* and a low number of rollout steps T for *Argmax*. Large red dots correspond to a finer target mesh resolution, and the purple dots interpolate between the two. Details on the target resolution parameters are found in Table 1. We find that ASMR provides high-quality refinements with consistent numbers of final mesh elements for any given target resolution, whereas the other methods yield widely different results for similar target resolutions when trained on different random seeds.

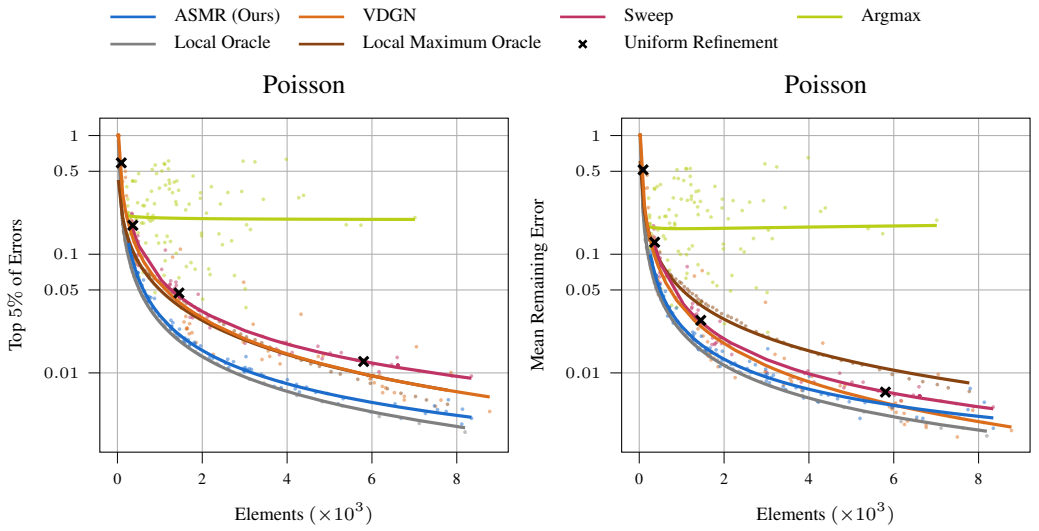

Figure 13: Pareto plot of the normalized (Left) Top $5\,\%$ of errors and (Right) mean remaining error and number of final mesh elements for the Poisson equation. The results are similar to the Top $0.1\,\%$ of errors as seen in Figure 4. The Top $5\,\%$ metric interpolates between the Top $0.1\,\%$ and the mean remaining error. We find that the performance of the *Local Maximum Oracle* degrades when evaluating more integration points, likely because it only focuses on the maximum error of each element by construction. ASMR provides refinements that perform well on all considered metrics.

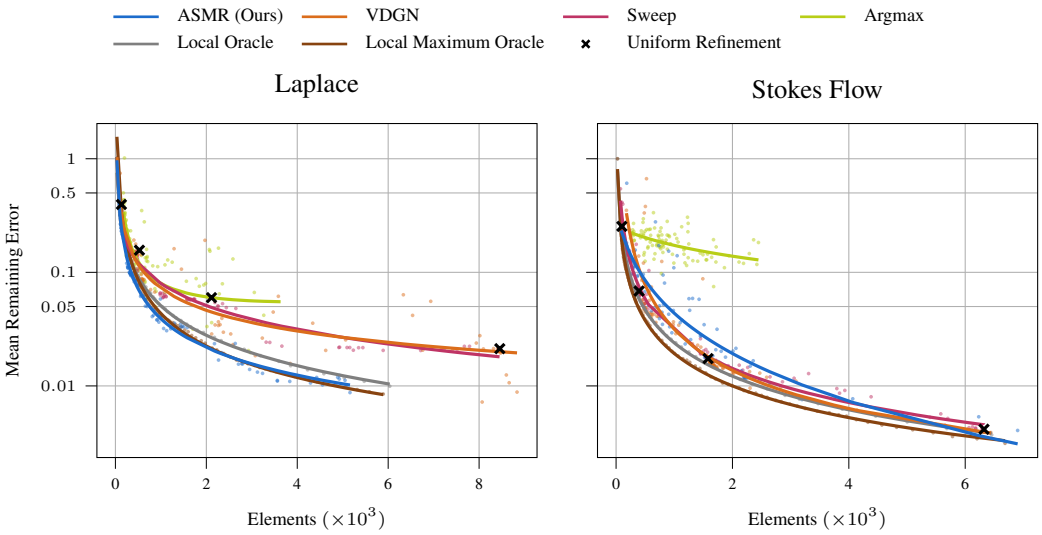

Figure 14: Pareto plot of the normalized mean remaining error for (Left) the Laplace equation and (Right) the Stokes flow. The mean remaining error weights areas with high error less when compared to the Top $0.1\,\%$ of errors. As a result, both the uniform refinement and the methods that produce more uniform meshes, such as *Sweep*, perform better on this metric. For the Stokes flow task, the mesh must be relatively uniform across the full length of the domain to reduce the total error mass, which likely causes the uniform refinements to be comparatively good.

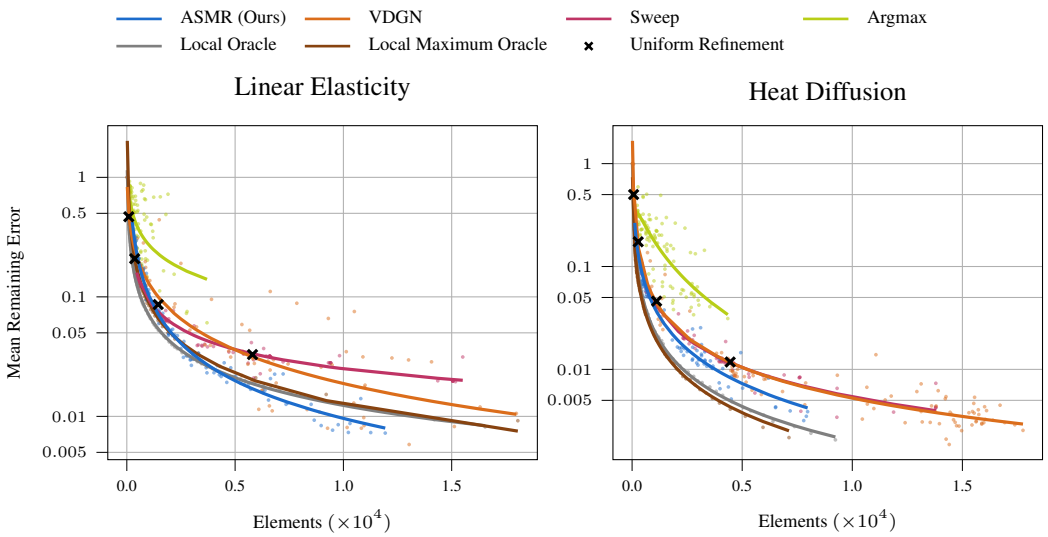

Figure 15: Pareto plot of the normalized mean remaining error for (Left) the linear elasticity and (Right) the heat diffusion task. The mean remaining error weights areas with high error less when compared to the Top $0.1\,\%$ of errors. Thus, uniform refinements and methods that produce uniform refinements generally perform better on this metric. Interestingly, ASMR outperforms both the *Local Oracle* and the *Local Maximum Oracle* on the linear elasticity task, likely because the task has global dependencies that are respected by our RL framework, but are ignored by the local heuristics.

E Hyperparameters

E.1 General Hyperparameters

We use the same hyperparameters across all methods and environments unless mentioned otherwise.

PPO. We largely follow the suggestions of [90] for our PPO parameters. We train each PPO policy for a total of 800 iterations. In each iteration, the algorithm samples 256 environment transitions and then trains on them for 5 epochs with a batch size of 32. The value function loss is multiplied with a factor of 0.5 and we clip the gradient norm to 0.5. The policy and value function clip ranges are chosen to be 0.2. We normalize the observations with a running mean and standard deviation. The discount factor is $\gamma = 0.99$ and advantages are estimated via Generalized Advantage Estimate [91] with $\lambda = 0.95$.

DQN. For DQN-based approaches, we instead train for $24 * 800 = 19200$ steps, where each step consists of executing an environment transition and then drawing a batch of samples 32 samples from the replay buffer for a single gradient update. Since every environment transition describes a full refinement step, including a mesh and its solution, we keep a total of 5000 transitions in the replay buffer. We additionally draw 500 initial random replay buffer samples before the first training step. During training, we draw actions using a Boltzmann distribution over the predicted Q-values per agent, where we linearly decrease the temperature of the distribution from 10 to 0.01 in the first 9600 steps. We find that this action selection strategy leads to more correlated actions when compared to an epsilon greedy action sampling, which stabilizes the training for our iterative mesh refinement problems. We update the target networks using Polyak averaging at a rate of 0.99 per step. Further, we follow previous work [92] and combine a number of common improvements for DQNs, namely double Q-learning [93], dueling Q-networks [94] and prioritized experience replay [95].

Neural Networks. All networks are implemented in PyTorch [96] and trained using the ADAM optimizer [97] with a learning rate of 3.0e-4 unless mentioned otherwise. All MLPs use 2 hidden layers and a latent dimension of 32. We use separate MPNs for the policy and the value function. Each MPN consists of 2 message passing steps, where each update function is represented as an MLP with *LeakyReLU* activation functions. The policy and value function heads are additional MLPs with *tanh* activation functions acting on the final latent node features of the MPN. All message aggregations $\bigoplus$ are mean aggregations. Additionally, we apply Layer Normalization [98] and

Residual Connections [99] independently for the node, edge and global features after each message passing step.

E.2 Baseline-Specific Parameters

To accommodate the step-wise nature of *Argmax* and *Sweep*, we increase the size of their replay buffer to 10000 when using DQN. For *Argmax*, we use a maximum refinement depth of 10 refinements per element to avoid numerical instabilities during simulation, skipping actions that try to refine elements that have been refined too often. We consider environment sequences of up to $T = 400$ steps since the method marks only one element at a time. Due to the computational demand of this method, we only evaluate it on 10 instead of 100 evaluation PDEs. For *Sweep*, the agent is placed on a random mesh element for each training step and may decide not to refine this element, resulting in no change in the mesh. Here, we follow the proposed hyperparameters for this approach and train each rollout for 200 steps. As this approach is based on purely local agents, we adapt our input features per element to consist of our regular node features, the global resource budget proposed by the authors, the mean solution and area of the element's neighbors and the average distance to them. The global budget is controlled via a maximum number of elements N_{max}, allowing to get refinements of different granularity. To accommodate for less overall changes in the mesh, we increase the number of environment transitions of PPO to 512, and the number of DQN steps to $96 * 800 = 76800$. Finally, we use a learning rate of 1.0e-5 instead of 3.0e-4 for the DQN variant of *VDGN* to stabilize its training.

E.3 Refinement Hyperparameters

The AMR methods considered in this work use different parameters to control the granularity of the final refined mesh. ASMR and *VDGN* use an element penalty α, while *Sweep* considers an element budget N_{max}. Similarly, *Argmax* varies the number of rollout steps T, and the *Local Oracle* and *Local Maximum Oracle* use different error thresholds θ. For each method and task, we choose 10 different values for the refinement parameter that showcase a wide range of final mesh resolutions. Table 1 lists the different ranges for these parameters for the different tasks. For stability purposes, we set a maximum number of 20000 elements for all experiments except for the *Sweep* baseline, as this baseline uses its own element budget instead. If this number is surpassed, a constant penalty of 1000 is subtracted from the reward and the episode terminates early.

Table 1: Ranges for the different refinement hyperparameters for all tasks. ASMR and *VDGN* apply an element penalty α, but only ASMR scales the area of each element with its area in Equation 3. *Sweep* uses an element budget N_{max}. *Argmax* varies the number of rollout steps T, and the *Local Oracle* and *Local Maximum Oracle* make use of different error thresholds θ.

Method	**Task**				
	Laplace	Poisson	Stokes Flow	Lin. Elasticity	Diffusion
ASMR (α)	$[0.01, 0.3]$	$[0.002, 0.1]$	$[0.006, 0.15]$	$[0.01, 0.15]$	$[0.003, 0.3]$
VDGN (α)	$[2\text{e}{-5}, 5\text{e}{-2}]$	$[2\text{e}{-5}, 5\text{e}{-2}]$	$[3\text{e}{-4}, 5\text{e}{-3}]$	$[1\text{e}{-5}, 1\text{e}{-2}]$	$[5\text{e}{-6}, 5\text{e}{-3}]$
Sweep (N_{max})	$[200, 3000]$	$[400, 5000]$	$[200, 3000]$	$[500, 6000]$	$[400, 5000]$
Argmax (T)	$[25, 400]$	$[25, 400]$	$[25, 400]$	$[25, 400]$	$[25, 400]$
Local Oracle (θ)	$[0.25, 1.00]$	$[0.1, 1.0]$	$[0.16, 1.0]$	$[0.02, 1.0]$	$[0.03, 1.0]$
L. Max. Oracle (θ)	$[0.20, 1.0]$	$[0.2, 1.0]$	$[0.1, 1.0]$	$[0.01, 1.0]$	$[0.02, 1.0]$

F Visualizations

We provide additional visualizations for our method on all tasks, and for all methods on the Poisson task. All visualizations show the final refined mesh of the respective method for 5 different refinement levels on 3 randomly selected PDEs. For the RL methods, all policies are taken from the first repetition of the 10 random seeds conducted for the respective experiment.

F.1 ASMR Refinements

We visualize exemplary refinements of ASMR policies for all considered tasks. The visualizations are given in Figure 16 (Laplace's equation), Figure 17 (Poisson's equation), Figure 18 (Stokes equation), Figure 19 (Linear Elasticity), and Figure 20 (Heat Diffusion). Across all tasks, ASMR is able to provide highly accurate refinements for different numbers of total elements.

F.2 Baseline Comparisons

Figure 21 shows refinements for *Argmax* for different total timesteps T, Figure 22 presents *VDGN* with different α values. Figure 23 visualizes refinements of *Sweep* for a varying number of maximum elements N_{max}, and Figures 24 and 25 show refinements of the *Local Oracle* and *Local Maximum Oracle* for different values of the threshold θ.

The visualizations show that the RL baselines struggle to provide consistent high-quality refinements for different mesh resolutions. The *Argmax* baseline sometimes focuses on uninteresting regions of the mesh or refines the same area too often. Generally, *VDGN* performs well, but it tends not to focus enough on the interesting regions of the domain. *Sweep* provides almost uniform refinements for most PDEs and element budgets, likely as a result of the misalignment in the environment transitions between training and inference. Finally, the error-based heuristics greedily refine the elements with the largest errors of their respective metric. For the *Local Maximum Oracle*, only the areas with the highest error are repeatedly refined every step, while the *Local Oracle* refines more uniformly as it considers the total error mass per mesh element. Both behaviors lead to locally optimal refinements on their respective metric, but may cause issues for PDEs with global dependencies [86] and conforming refinements.

F.3 Element Markings

Figure 26 visualizes a full rollout of our method, including the markings of the elements after every step.

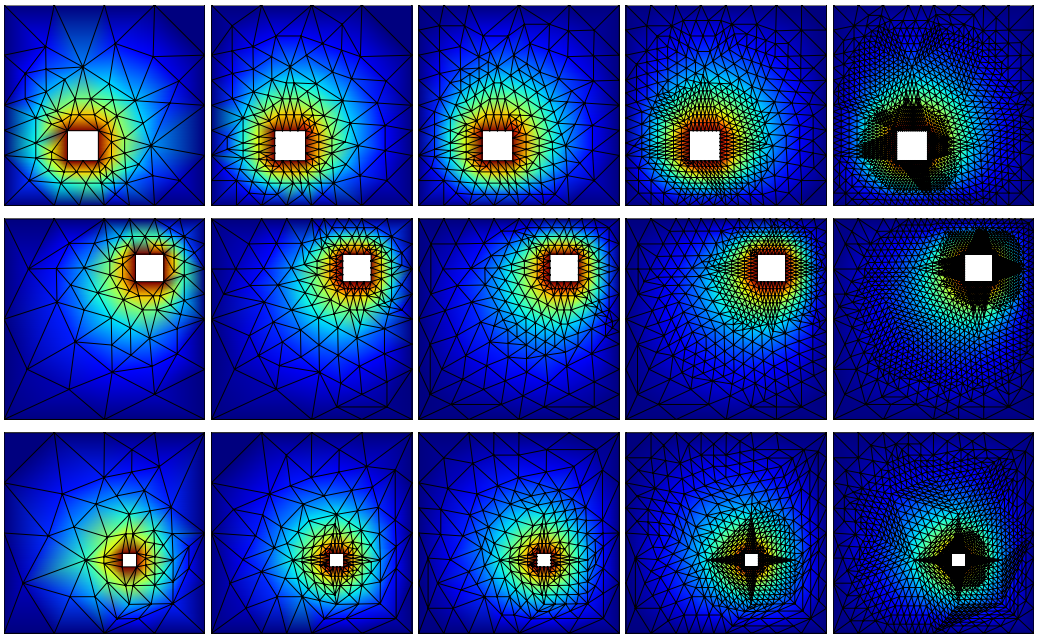

Figure 16: Final refined meshes of ASMR for randomly sampled PDEs for the Laplace equation for different element penalties α.

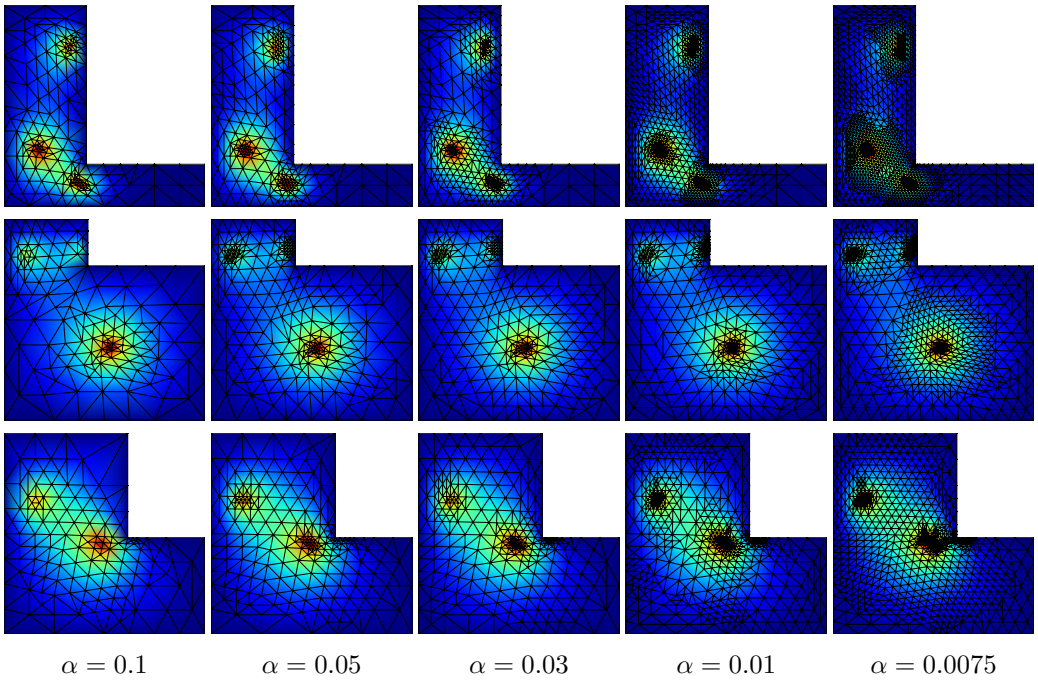

Figure 17: Final refined meshes of ASMR for randomly sampled PDEs for the Poisson equation for different element penalties α.

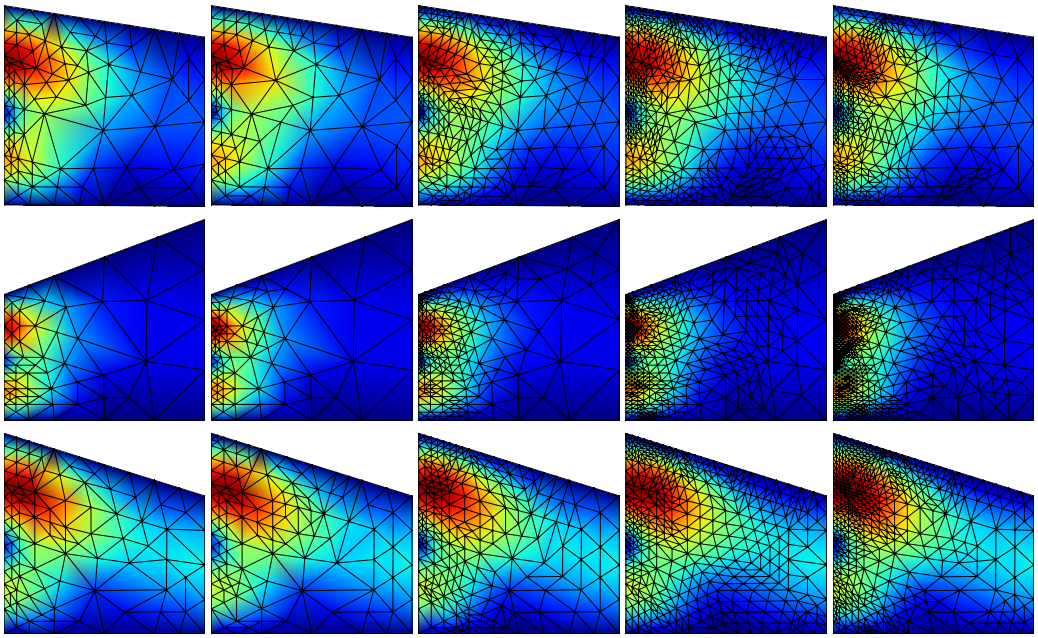

Figure 18: Final refined meshes of ASMR for randomly sampled PDEs for the Stokes flow task for different element penalties α.

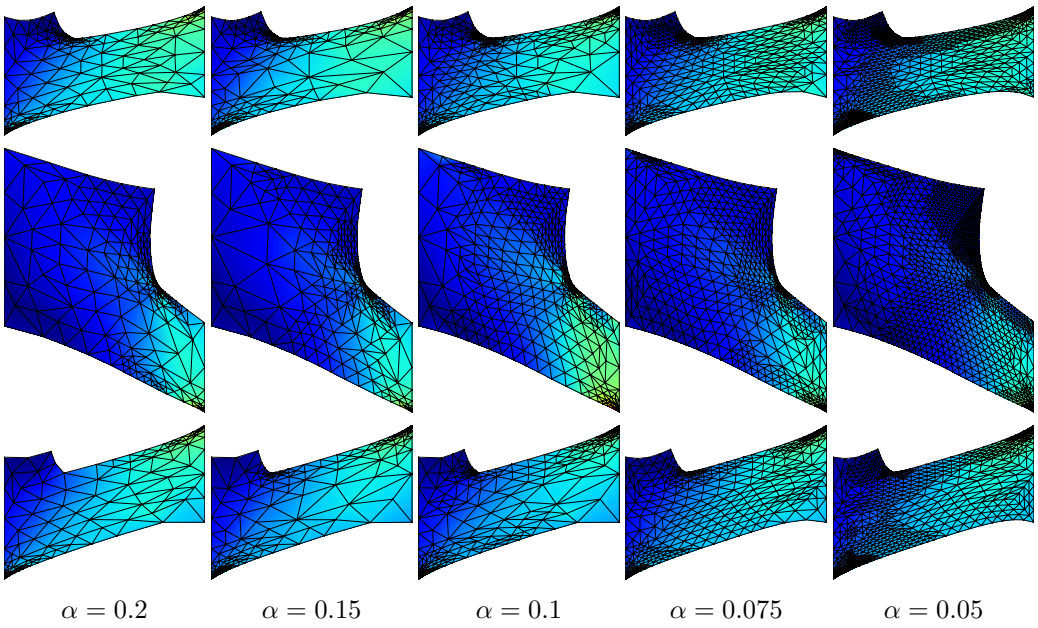

Figure 19: Final refined meshes of ASMR for randomly sampled PDEs for the linear elasticity task for different element penalties α. The visualizations show the deformed meshes, which are originally L-shaped.

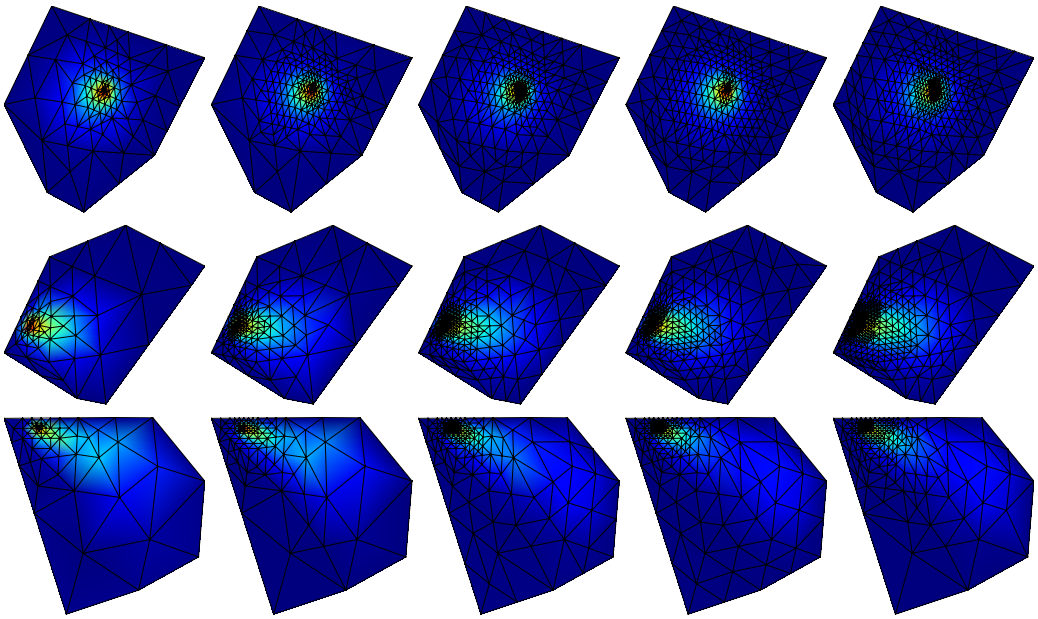

Figure 20: Final refined meshes of ASMR for randomly sampled PDEs for the non-stationary heat diffusion task for different element penalties α.

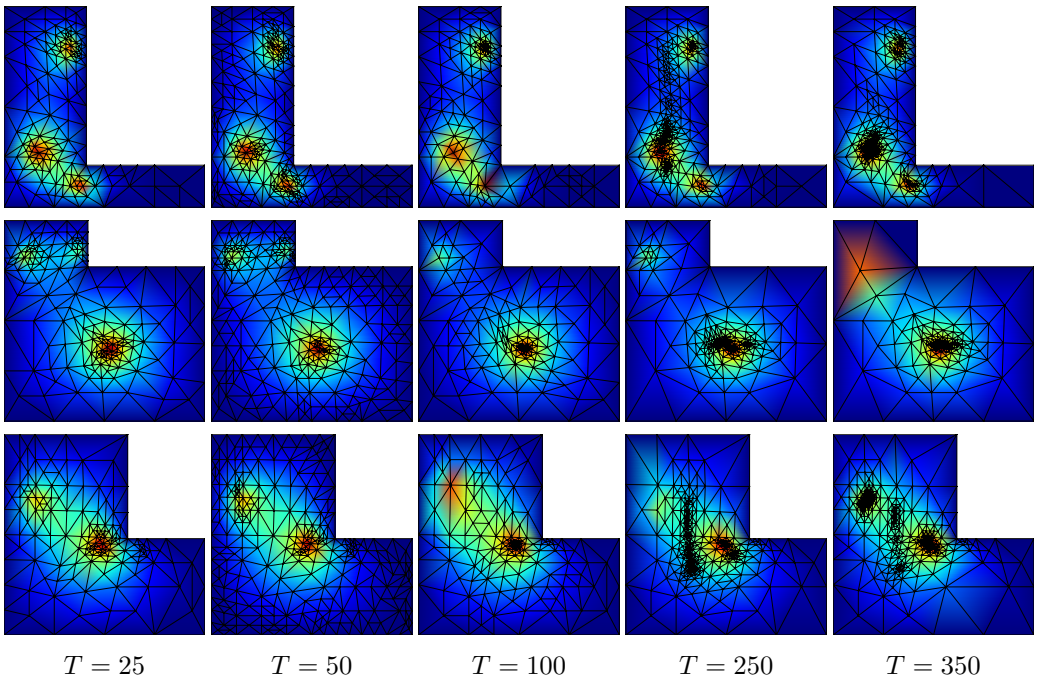

Figure 21: Final refined meshes of the *Argmax* baseline for the Poisson equation on randomly sampled PDEs for different environment rollout lengths T.

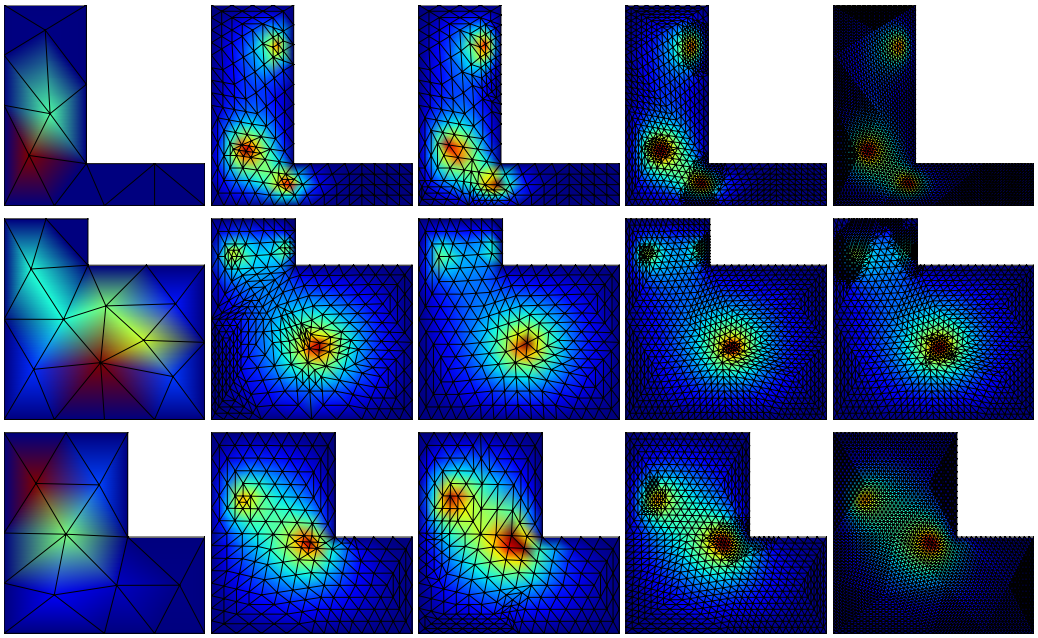

Figure 22: Final refined meshes of the *VDGN* baseline for the Poisson equation on randomly sampled PDEs for different element penalties α.

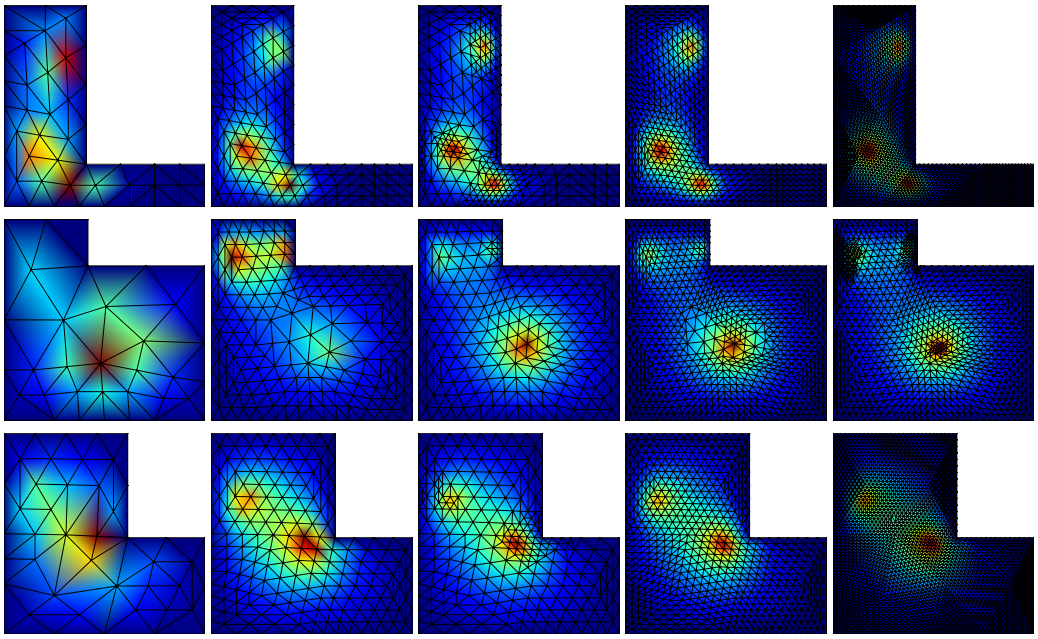

Figure 23: Final refined meshes of the *Sweep* baseline for the Poisson equation on randomly sampled PDEs for different maximum numbers of elements N_{max}.

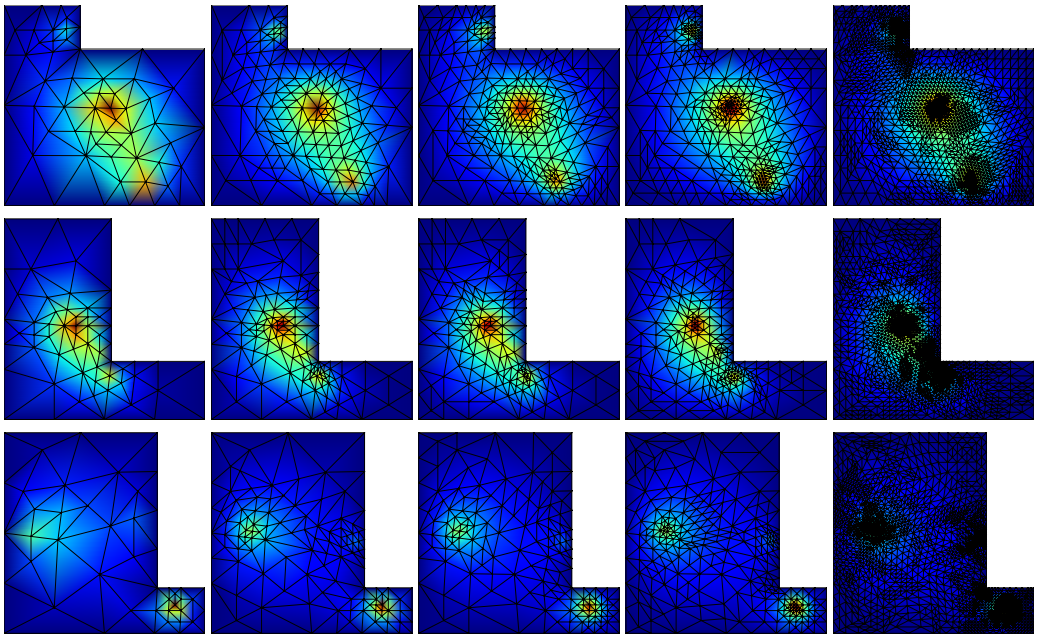

Figure 24: Final refined meshes of the *Local Oracle* baseline for the Poisson equation on randomly sampled PDEs for different error thresholds θ.

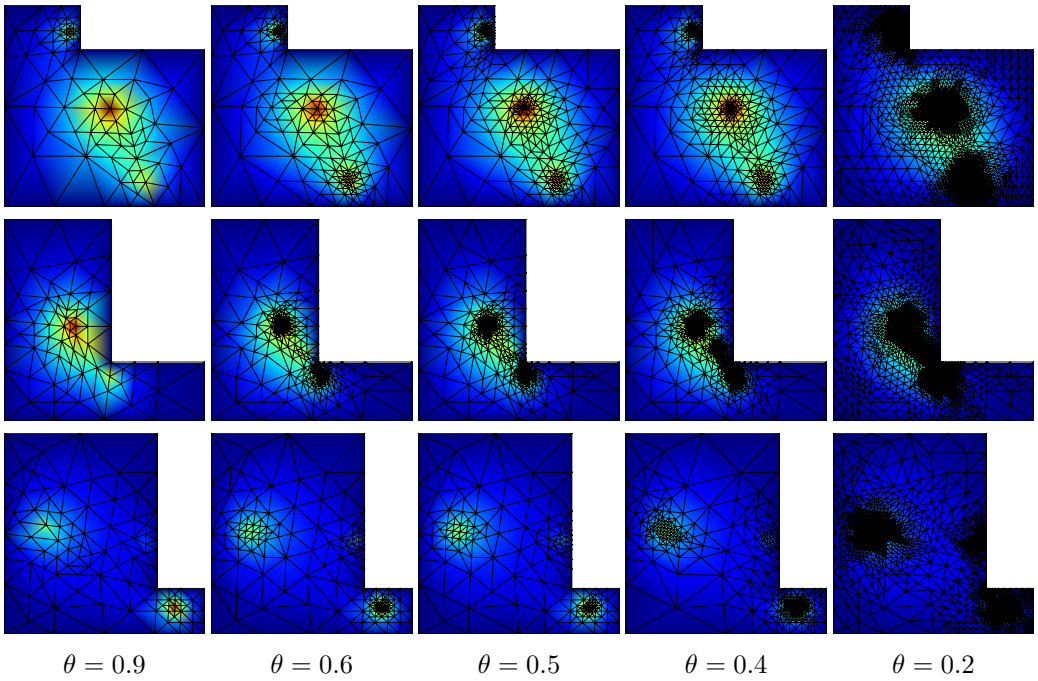

Figure 25: Final refined meshes of the *Local Maximum Oracle* baseline for the Poisson equation on randomly sampled PDEs for different error thresholds θ.

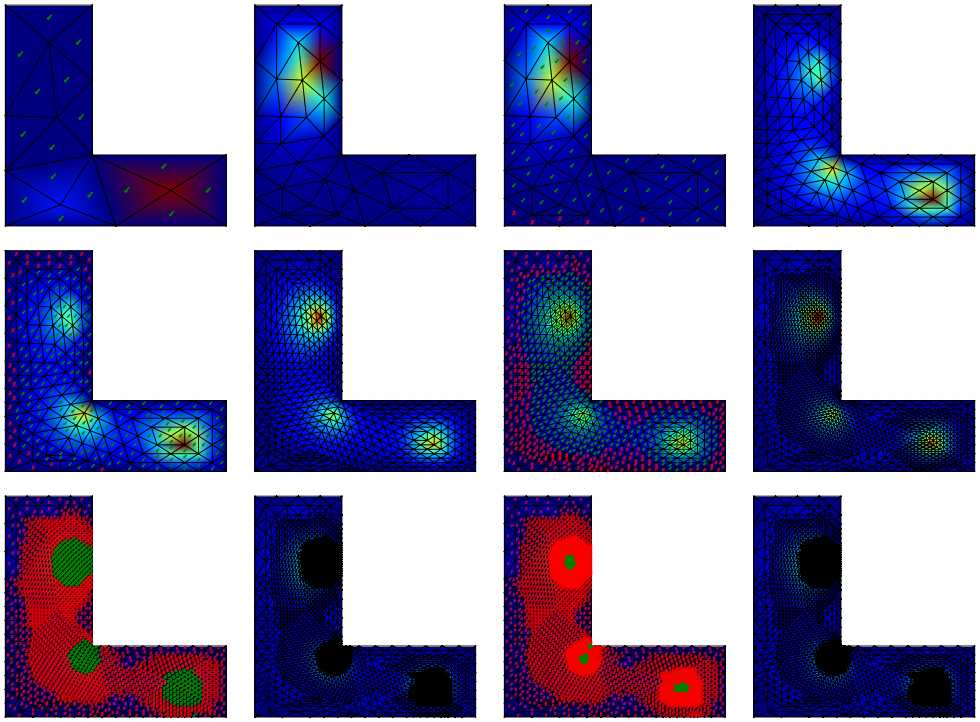

Figure 26: Visualization of a full rollout of our method on a Poisson task, including the markings of the elements after every step. The figures in the first and third row show the markings, and the second and fourth row show the resulting refined meshes.