Multi-Agent Reinforcement Learning for Adaptive Mesh Refinement

Jiachen Yang
LLNL
Livermore, CA, USA
yang40@llnl.gov

Ketan Mittal
LLNL
Livermore, CA, USA
mittal3@llnl.gov

Tarik Dzanic
Texas A&M University
College Station, TX, USA
tdzanic@tamu.edu

Socratis Petrides
LLNL
Livermore, CA, USA
petrides1@llnl.gov

Brendan Keith
Brown University
Providence, RI, USA
brendan_keith@brown.edu

Brenden Petersen
LLNL
Livermore, CA, USA
petersen33@llnl.gov

Daniel Faissol
LLNL
Livermore, CA, USA
faissol1@llnl.gov

Robert Anderson
LLNL
Livermore, CA, USA
anderson110@llnl.gov

ABSTRACT

Adaptive mesh refinement (AMR) is necessary for efficient finite element simulations of complex physical phenomenon, as it allocates limited computational budget based on the need for higher or lower resolution, which varies over space and time. We present a novel formulation of AMR as a fully-cooperative Markov game, in which each element is an independent agent who makes refinement and de-refinement choices based on local information. We design a novel deep multi-agent reinforcement learning (MARL) algorithm called Value Decomposition Graph Network (VDGN), which solves the two core challenges that AMR poses for MARL: posthumous credit assignment due to agent creation and deletion, and unstructured observations due to the diversity of mesh geometries. For the first time, we show that MARL enables anticipatory refinement of regions that will encounter complex features at future times, thereby unlocking entirely new regions of the error-cost objective landscape that are inaccessible by traditional methods based on local error estimators. Comprehensive experiments show that VDGN policies significantly outperform error threshold-based policies in global error and cost metrics. We show that learned policies generalize to test problems with physical features, mesh geometries, and longer simulation times that were not seen in training. We also extend VDGN with multi-objective optimization capabilities to find the Pareto front of the tradeoff between cost and error.

KEYWORDS

multi-agent reinforcement learning; adaptive mesh refinement; numerical analysis; graph neural network

ACM Reference Format:
Jiachen Yang, Ketan Mittal, Tarik Dzanic, Socratis Petrides, Brendan Keith, Brenden Petersen, Daniel Faissol, and Robert Anderson. 2023. Multi-Agent Reinforcement Learning for Adaptive Mesh Refinement. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 17 pages.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom. © 2023 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

1 INTRODUCTION

The finite element method (FEM) [6] is instrumental to numerical simulation of partial differential equations (PDEs) in computational science and engineering [21, 27]. For multi-scale systems with large variations in local features, such as combinations of regions with large gradients that require high resolution and regions with flat solutions where coarse resolution is sufficient, an efficient trade-off between solution accuracy and computational cost requires the use of adaptive mesh refinement (AMR). The goal of AMR is to adjust the finite element mesh resolution dynamically during a simulation, by refining regions that can contribute the most to improvement in accuracy relative to computational cost.

For evolutionary (i.e., time-dependent) PDEs in particular, a long-standing challenge is to find anticipatory refinement strategies that optimize a long-term objective, such as an efficient tradeoff between final solution accuracy and cumulative degrees of freedom (DoF). Anticipatory refinement strategies would preemptively refine regions of the mesh that will contain solution features (e.g., large gradients) right before these features actually occur. This is hard for existing approaches to achieve. Traditional methods for AMR rely on estimating local refinement indicators (e.g., local error [43]) and heuristic marking strategies (e.g., greedy error-based marking) [4, 7]. Recent data-driven methods for mesh refinement apply supervised learning to learn a fast neural network estimator of the solution from a fixed dataset of pre-generated high-resolution solutions [23, 37]. However, greedy strategies based on local information cannot produce an optimal sequence of anticipatory refinement decisions in general, as they do not have sufficient information about features that may occur at subsequent time steps, while supervised methods do not directly optimize a given long-term objective. These challenges can be addressed by formulating AMR as a sequential decision-making problem and using reinforcement learning (RL) [32] to optimize a given objective directly. However, current single-agent RL approaches for AMR either do not easily

support refinement of multiple elements per solver step [41]; faces different definitions of the environment transition at training and test time [10]; or work by selecting a single global error threshold [11], which poses difficulties for anticipatory refinement.

In this work, we present the first formulation of AMR as a Markov game [18, 25] and propose a novel fully-cooperative deep multi-agent reinforcement learning (MARL) algorithm [9, 16, 40] called Value Decomposition Graph Network (VDGN) to train a team of independently and simultaneously acting agents, each of which is a decision-maker for an element, to optimize a global performance metric and find anticipatory refinement policies. Because refinement and de-refinement actions at each step of the AMR Markov game leads to the creation and deletion of agents, we face the posthumous credit assignment problem [8]: agents who contributed to a future reward are not necessarily present at the future time to experience it. We show that VDGN, by virtue of centralized training with decentralized execution [24], addresses this key challenge. By leveraging graph networks as the inductive bias [5], VDGN supports meshes with varying number of elements at each time step and can be applied to meshes of arbitrary size, depth, and geometry. Moreover, graph attention layers [36] in VDGN enable each element to receive information within a large local neighborhood, so as to anticipate incoming or outgoing solution features and learn to take preemptive actions. Experimentally, using the advection equation as a pedagogical benchmark problem, we show for the first time that MARL: (a) displays anticipatory refinement behavior; (b) generalizes to different initial conditions, initial mesh resolutions, simulation durations, and mesh geometries; (c) significantly improves error and cost metrics compared to local error threshold-based policies, and unlocks new regions of the error-cost optimization landscape. Furthermore, we augment VDGN with a multi-objective optimization method to train a single policy that discovers the Pareto front of error and cost.

2 PRELIMINARIES

2.1 Finite Element Method

The finite element method [6] models the domain of a PDE with a mesh that consists of nonoverlapping geometric elements. Using the weak formulation and choosing basis functions to represent the PDE solution on these elements, one obtains a system of linear equations that can be numerically solved. The shape and sizes of elements determine solution error, while the computational cost is primarily determined by the number of elements. Improving the trade-off between error and cost is the objective of AMR. This work focuses purely on h-refinement, in which refinement of a coarse element produces multiple new fine elements (e.g., isotropic refinement of a 2D quadrilateral element produces four new elements), whereas de-refinement of a group of fine elements produces a new coarse element. In both cases, the original coarse element or fine elements are removed from the FEM discretization.

2.2 Notation

Each element is identified with an agent, denoted by $i \in \{1, \dots, n_t\}$, where n_t is variable across time step $t = 0, 1, \dots, T_{\max}$ within each episode of length $T_{\max}$, and $n_t \in [N_{\max}]$, where $N_{\max}$ is constrained by $\text{depth}_{\max}$ to be $n_x \cdot n_y \cdot 4^{\text{depth}_{\max}}$ in the case of quadrilateral elements. Let each element i have its own individual observation space $\mathcal{S}^i$ and action space $\mathcal{A}^i$. Let s denote the global state, o^i and a^i denote agent i's individual observation and action, respectively, and $a := (a^1, \dots, a^{n_t})$ denote the joint action by n_t agents. In the case of AMR, all agents have the same observation and action spaces. Let R denote a single global reward for all agents and let P denote the environment transition function, both of which are well-defined for any number of agents $n_t \in [N_{\max}]$. Let $\gamma \in (0, 1]$ be the discount factor. The FEM solver time at episode step t is denoted by τ (we omit the dependency $\tau(t)$ for brevity), which advances by τ_{step} in increments of $d\tau = 0.002$ during each discrete step $t \to t + 1$ up to final simulation time τ_f.

For element i at time t, let u_t^i and $\hat{u}_t^i$ denote the true and numerical solution, respectively, and let $c_t^i := \|u_t^i - \hat{u}_t^i\|_2$ denote the error. Let $c_t := \sqrt{\sum_{i=1}^{n_t} (c_t^i)^2}$ denote the global error of a mesh with n_t elements. Let $\text{depth}(i) = 0, 1, \dots, \text{depth}_{\max}$ denote the refinement depth of element i. Let $d_t := \sum_{s=0}^{t} \text{DoF}_s$ denote the cumulative degrees of freedom (DoF) of the mesh up to step t, which is a measure of computational cost, and let d_{thres} be a threshold (i.e., constraint) on the cumulative DoF seen during training.

2.3 Value Decomposition Network

In the paradigm of centralized training with decentralized execution (CTDE) [24], global state and reward information is used in centralized optimization of a team objective at training time, while decentralized execution allows each agent to take actions conditioned only on their own local observations, independently of other agents, both at training time and at test time. One simple yet effective way to implement this in value-based MARL is Value Decomposition Networks (VDN) [31]. VDN learns within the class of global action-value functions $Q(s, a)$ that decompose additively:

$$Q(s, a) := \sum_{i=1}^{n} Q^i(s^i, a^i), \tag{1}$$

where Q^i is an individual utility function representing agent i's contribution to the joint expected return.

This decomposition is amenable for use in Q-learning [39], as it satisfies the *individual-global-max* (IGM) condition:

$$\underset{\mathbf{a} \in \Pi_{i=1}^{n} \mathcal{A}^i}{\text{argmax}}\ Q(s, a) = \left[\underset{a^1 \in \mathcal{A}^1}{\text{argmax}}\, Q^1(s^1, a^1), \dots, \underset{a^n \in \mathcal{A}^n}{\text{argmax}}\, Q^n(s^n, a^n) \right]. \tag{2}$$

This means the individual maxima of Q^i provide the global maximum of the joint Q function for the Q-learning update step [20, 39], which scales linearly rather than exponentially in the number of agents. Using function approximation for Q_θ^i with parameter θ, the VDN update equations using replay buffer $\mathcal{B}$ are:

$$\theta \leftarrow \theta - \nabla_\theta \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{B}} \left[(y_{t+1} - Q_\theta(s_t, a_t))^2 \right] \tag{3}$$

$$y_{t+1} := R_t + \perp \gamma Q_\theta(s_{t+1}, a)|_{a = [\text{argmax}_{a^i} Q_\theta^i(s_{t+1}^i, a^i)]_{i=1}^n}, \tag{4}$$

where $\perp$ is the stop-gradient operator.

3 AGENT CREATION AND DELETION

Each agent's refinement and de-refinement action has long-term impact on the global error (e.g., refining before arrival of a feature

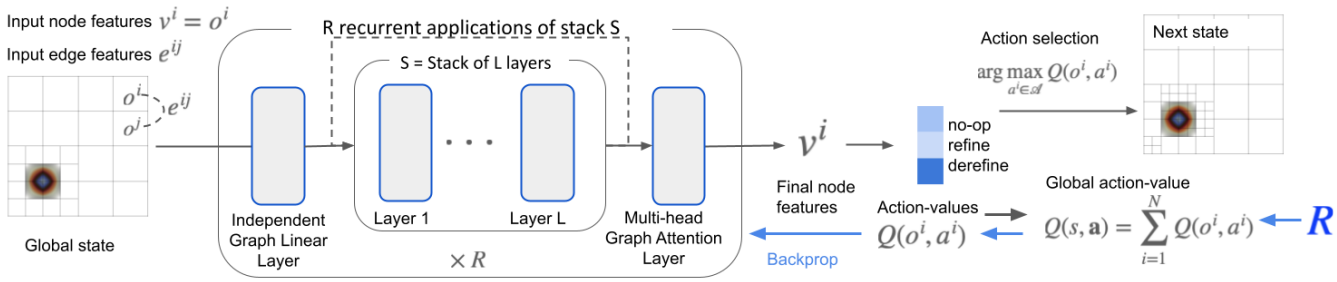


Figure 1: A state-action-next state transition in the Markov game for AMR. Shown in detail are the action selection (rightward) and learning (leftward) processes in the Value Decomposition Graph Network.

would reduce error upon its arrival). However, since all elements that refine or de-refine are removed immediately and their individual trajectories terminate, they do not observe future states and future rewards. This is the posthumous multi-agent credit assignment problem [8], which we propose to address using centralized training. First, we show that an environment with variable but bounded number of agents can be written as a Markov game [25] (see proof in Appendix B).

Proposition 1. *Let $\mathcal{M}$ denote a multi-agent environment where the number of agents $n_t = 1, \ldots, N_{max}$ can change at every time step $t = 0, 1, \ldots, T_{max}$ due to agent-creation and agent-deletion actions in each agent's action space. At each time t, the environment is defined by the tuple $\left(\{\mathcal{S}^i\}_{i=1}^{n_t}, \{\mathcal{A}^i\}_{i=1}^{n_t}, R, P, \gamma, n_t, N_{\max}\right)$. $\mathcal{M}$ can be written as a Markov game with a stationary global state space and joint action space that do not depend on the number of currently existing agents.*

Centralized training for posthumous multi-agent credit assignment. Our key insight for addressing the posthumous credit assignment problem stems from Proposition 1: because the environment is Markov and stationary, we can use centralized training with a global reward to train a global state-action value function $Q(s, a)$ that (a) persists across time and (b) evaluates the expected future return of any (s_t, a_t). Crucially, these two properties enable $Q(s, a)$ to sidestep the issue of posthumous credit assignment, since the value estimate of a global state will be updated by future rewards via temporal difference learning regardless of agent deletion and creation. To arrive at a truly multi-agent approach, we factorize the global action space so that each element uses its individual utility function $Q^i(o^i, a^i)$ to choose its own action from {no-op, refine, de-refine}. This immediately leads to the paradigm of centralized training with decentralized execution [24], of which VDN (eq. (1)) is an example. We discuss the pros and cons of other formulations in Appendix E.

Effective space. Agent creation and deletion means that the accessible region of the global state-action space changes over time during each episode. While this is not a new phenomenon, AMR is special in that the sizes of the informative subset of the global state and the available action set depend directly on the current number of existing agents. Hence, a key observation for algorithm development is that a model-free multi-agent reinforcement learning algorithm only needs to account for the accessible state-action space $\prod_{i=1}^{n_t} \mathcal{S}^i \times \mathcal{A}^i$ at each time step, since the expansion or contraction of that space is part of the environment dynamics that are accounted implicitly by model-free MARL methods. In practice, this means all dummy states $s_\nexists$ do not need to be input to policies, and policies do not need to output the (mandatory) no-op actions for the $N_{\max} - n_t$ nonexistent elements at time t. This informs our concrete definition of the Markov game for AMR in Section 4.

4 AMR AS A MARKOV GAME

State. The global state is defined by the collection of all individual element observations and pairwise relational features. The individual observation o^i of element i consists of:

- $\log(c_t^i)$: logarithm of error at element i at time t.
- 1-hot representation of element refinement depth.

Relational features e^{ij} are defined for each pair of spatially-adjacent elements i, j that form edge $e = (i, j)$ (directed from j to i) in the graph representation of the mesh (see Section 5), as a 1-dimensional vector concatenation of the following:

- $\mathbf{1}[\text{depth}(i) - \text{depth}(j)]$: 1-hot vector indicator of the difference in refinement depth between i and j.
- $\langle u_j, \frac{\Delta x}{\|\Delta x\|_2^2} \rangle \cdot \tau_{\text{step}}$: Dimensionless inner product of velocity u_j at element j with the displacement vector between i and j. Here $\Delta x := (x_i - x_j)$, where x_i is the center of element i.

We use the velocity u_j at the sender element so that the receiver element is informed about incoming or outgoing PDE features and can act preemptively.

Action. All elements have the same action space:

$$\mathcal{A} := \{\texttt{no-op}, \texttt{refine}, \texttt{de-refine}\},$$

where `no-op` means the element persists to the next decision step; `refine` means the element is equipartitioned into four smaller elements; `de-refine` means that the element opts to coalesce into a larger coarse element, subject to feasibility constraints specified by the transition function (see below).

Transition. Given the current state and agents' joint action, which is chosen simultaneously by all agents, the transition $P\colon \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is defined by these steps:

(1) Apply de-refinement rules to each element i whose action is `de-refine`: (a) if it is at the coarsest level, i.e., $\text{depth}(i) = 0$, or it belongs to a group of sibling elements in which *any* element chose to refine, then its choice is overridden to be `no-op`; (b) if, within its group of sibling elements, a majority (or tie) of elements chose `de-refine`, then the whole group is de-refined.
(2) Apply refinement to each agent who chose `refine`.

(3) Step the FEM simulation forward in time by τ_{step} and compute a new solution on the updated mesh. An episode terminates when $\tau + d\tau > \tau_f$ or $d_t > d_{\text{thres}}$. This follows standard procedure in FEM [2] and knowledge of the transition dynamics is not used by the proposed model-free MARL approach.

Reward. We carefully design a shaped reward [22] that encourages agents to minimize the final global error. Let $c_0 = 1.0$ be a dummy initial global error. The reward at step $t = 1, 2, \ldots$ is

$$R_t = \begin{cases} p \cdot (\tau - \tau_f) + \log(c_{t-1}), & \text{if } d_t > d_{\text{thres}} \wedge \tau + d\tau < \tau_f \\ \log(c_{t-1}/c_t) - p \cdot \max(0, \frac{d_t}{d_{\text{thres}}} - 1), & \text{if } \tau + d\tau \geq \tau_f \\ \log(c_{t-1}/c_t), & \text{else} \end{cases} \tag{5}$$

The first case applies a penalty p when the cumulative DoF exceeds the DoF threshold before reaching the simulation final time. The second case applies a penalty based on the amount by which the DoF threshold is exceeded when the final time is reached. The last case provides the agents with a dense learning signal based on potential-based reward shaping [22], so that the episode return (i.e., cumulative reward at the end of the episode) is $R = -\log(c_{T_{\max}})$ in the absence of any of the other penalties.

Objective. We consider a fully-cooperative Markov game in which all agents aim to find a shared parameterized policy $\pi_\theta \colon \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ to maximize the objective

$$\max_\theta J(\theta) := \mathbb{E}_{s_{t+1} \sim P(\cdot|a,s_t), a \sim \pi_\theta(\cdot|s)} \left[\sum_{t=0}^{T} \gamma^t R_t \right] \tag{6}$$

Remark 1. *By using time limit τ_f and DoF threshold d_{thres} in the reward at training time, while not letting agents observe absolute time and cumulative DoF, agents must learn to make optimal decisions based on local observations, which enables generalization to longer simulation duration and DoF budgets at test time.*

Remark 2. *For problems without an easily computable analytical solution to compute the true error, one may use an error estimator for the element observations. The reward, which is only needed at training time and not at test time, can still be based on error with respect to a highly resolved reference simulation. Empirical results on this configuration are provided in Appendix D.*

5 VALUE DECOMPOSITION GRAPH NETWORK

To enable anticipatory refinement in the time-dependent case, an element must observe a large enough neighborhood around itself. However, it is difficult to define a fixed local observation window for general mesh geometries, element shapes, and boundaries. Instead, we use Graph Networks [5, 28] as a general inductive bias to learn representations of element interactions on arbitrary meshes.

Specifically, we construct a policy based on Graph Attention Networks [36], which incorporates self-attention [35] into graph networks. At each step t, the mesh is represented as a graph $\mathcal{G} = (V_t, E_t)$. Each node v^i in $V = \{v^i\}_{i=1:n_t}$ corresponds to element i and its feature is initialized to be the element observation o^i. $E = \{e^k = (r^k, s^k)\}_{k=1:N^e}$ is a set of edges, where an edge e^k exists between sender node s^k and receiver node r^k if and only if they are spatially adjacent (i.e., sharing either a face or a vertex in the mesh). Its feature is initialized to be the relational feature $e^{r^k s^k}$.

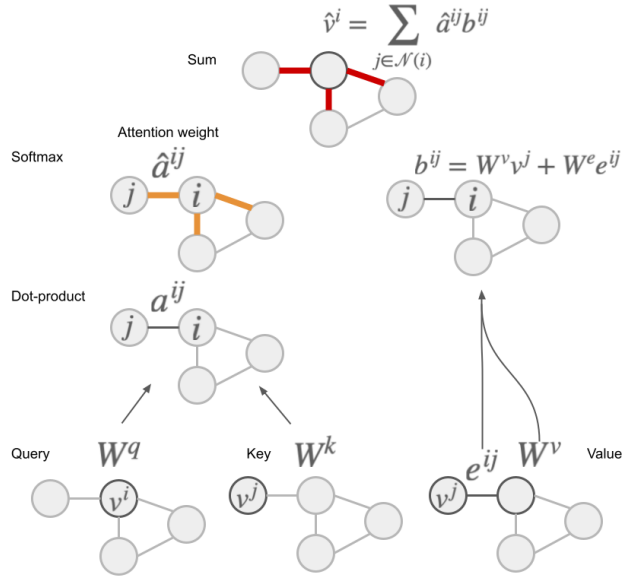


Figure 2: Graph Attention Layer. A softmax over all edges connected to node i produces attention weights $\hat{a}^{ij}$ for edge (i, j) (eq. (8)). A weighted sum over values b^{ij} with weight $\hat{a}^{ij}$ produces the updated node feature $\hat{v}^i$ (eq. (10)).

5.1 Graph Attention Layer

In a graph attention layer, each node is updated by a weighted aggregation over its neighbors: weights are computed by self-attention using node features as queries and keys, then applied to values that are computed from node and edge features.

Self-attention weights $\hat{a}^{ij}$ for each node i are computed as follows (see Figure 2): 1) we define queries, keys, and values as linear projections of node features, via weight matrices W^q, W^k, and W^v (all $\in \mathbb{R}^{d \times \dim(v)}$) shared for all nodes; 2) for each edge (i, j), we compute a scalar pairwise interaction term a^{ij} using the dot-product of queries and keys; 3) for each receiver node i with sender node $j \in \mathcal{N}_i$, we define the attention weight as the j-th component of a $\texttt{softmax}$ over all neighbors $k \in \mathcal{N}_i$:

$$a^{ij} := W^q v^i \cdot W^k v^j \qquad \text{for } (i, j) \in E, \tag{7}$$

$$\hat{a}^{ij} := \texttt{softmax}_j(\{a^{ik}\}_k) = \frac{\exp(a^{ij})}{\sum_{k \in \mathcal{N}_i} \exp(a^{ik})} \qquad \text{for } j \in \mathcal{N}_i\,. \tag{8}$$

We use these attention weights to compute the new feature for each node i as a linear combination over its neighbors $j \in \mathcal{N}_i$ of projected values $W^v v^j$. Edge features e^{ij}, with linear projection using $W^e \in \mathbb{R}^{d \times \dim(e)}$, capture the relational part of the observation:

$$b^{ij} := W^v v^j + W^e e^{ij} \qquad \text{for } (i, j) \in E\,, \tag{9}$$

$$\hat{v}^i := \sum_{j \in \mathcal{N}_i} \hat{a}^{ij} b^{ij} \qquad \text{for } i \in V\,. \tag{10}$$

Despite being a special case of the most general message-passing flavor of graph networks [5], graph attention networks separate the learning of $\hat{a}^{ij}$, the scalar importance of interaction between i and j relative to other neighbors, from the learning of b^{ij}, the vector determining how j affects i. This additional inductive bias

reduces the functional search space and can improve learning with large receptive fields around each node, just as attention is useful for long-range interactions in sequence data [35].

Multi-head graph attention layer. We extend graph attention by building on the mechanism of multi-head attention [35], which uses H independent linear projections $(W^{q,h}, W^{k,h}, W^{v,h}, W^{e,h})$ for queries, keys, values and edges (all projected to dimension d/H), and results in H independent sets of attention weights $\hat{a}^{ij,h}$, with $h = 1, 2, \ldots, H$. This enables attention to different learned representation spaces and was found to stabilize learning [36]. The new node representation with multi-head attention is the concatenation of all output heads, with an output linear projection $W^o \in \mathbb{R}^{d \times d}$. Appendix A.1 shows the multi-head versions of eqs. (7) to (10).

5.2 Value Decomposition Graph Network

We use the graph attention layer to define the Value Decomposition Graph Network (VDGN), shown in Figure 1. Firstly, an independent graph network block [5] linearly projects nodes and edges independently to $\mathbb{R}^d$.

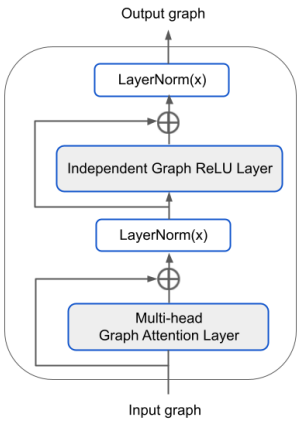


Figure 3: VDGN layer

Each layer F of VDGN involves two sub-layers: a multi-head graph attention layer followed by a fully-connected independent graph network block with ReLU nonlinearity. For each of these sub-layers, we use residual connections [15] and layer normalization [3], so that the transformation of input graph g is $\text{LayerNorm}(g + \text{SubLayer}(g))$. This is visualized in Figure 3. We define a VDGN *stack* as $S := F_L(F_{L-1}(\cdots F_1(g)\cdots))$ with L unique layers without weight sharing, then define a single forward pass by r recurrent applications of S: $P(g) = S(S(\cdots S(g)\cdots))$ with r instances of S. Finally, to produce $|\mathcal{A}|$ action-values for each node (i.e., element) i, we apply a final graph attention layer whose output linear projection is $W^o_{\text{out}} \in \mathbb{R}^{|\mathcal{A}| \times d}$, so that each final node representation is $\hat{v}^i \in \mathbb{R}^{|\mathcal{A}|}$, interpreted as the individual element utility $Q(o^i, a^i)$ for all possible actions $a^i \in \mathcal{A}$.

VDGN is trained using eq. (1) in a standard Q-learning algorithm [20], along a number of value-function based algorithmic improvements detailed in Section 5.4.

5.3 Symmetries

Methods for FEM and AMR should respect symmetries of the physics being simulated. For the simulations of interest in this work, we require a refinement policy to satisfy two properties: a) spatial equivariance: given a spatial rotation or translation of the PDE, the mesh refinement decisions should also rotate or translate in the same way; b) time invariance: the same global PDE state at different absolute times should result in the same refinement decisions. By construction of node and edge features, and the fact that graph neural networks operate on individual nodes and edges independently, we have the following result, proved in Appendix B.

Proposition 2. *VDGN is equivariant to global rotations and translations of the error and velocity field, and it is time invariant.*

5.4 Improvements to VDGN

A number of independent improvements [17] to single-agent Deep Q Network [20] can provide complementary gains in learning speed and performance. Since VDGN fundamentally is a Q function-based method, we can extend it with double Q-learning [34], dueling networks [38], and prioritized replay [29]. Details are provided in Appendix A.2. We did not employ the other improvements contained in Rainbow [17] (noisy networks, distributional Q-learning, and multi-step returns) because the environment dynamics are deterministic, and the episode horizon at train time is short.

5.5 Multi-objective VDGN

In applications where a user's preference between minimizing error and reducing computational cost is not known until test time, one cannot *a priori* combine error and cost into a single scalar reward at training time. Instead, one must take a multi-objective optimization viewpoint [14] and treat cost and error as separate components of a vector reward function $\mathbf{R} = (R^c, R^e)$. The components encourage lower DoFs and lower error, respectively, and are defined by

$$R^c_t := \frac{d_{t-1} - d_t}{d_{\text{thres}}}; \qquad R^e_t := \log(c_{t-1}) - \log(c_t). \tag{11}$$

The objective is $\mathbb{R}^2 \ni \mathbf{J}(\theta) := \mathbb{E}_{s_{t+1} \sim P(\cdot|a,s_t), a \sim \pi_\theta(\cdot|s)}[\sum_{t=0}^{T} \gamma^t \mathbf{R}_t]$, which is vector-valued. We focus on the widely-applicable setting of linear preferences, whereby a user's scalar utility based on preference vector ω is $\omega^T \mathbf{R}$ (e.g., $\omega = [0.5, 0.5]$ implies the user cares equally about cost and error). At training time, we randomly sample $\omega \in \Omega$ in each episode and aim to find an optimal action-value function $\mathbf{Q}^*(s, a, \omega) := \arg_Q \sup_{\pi \in \Pi} \omega^T \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^T \mathbf{R}_t]$, where $\arg_Q$ extracts the vector $E_\pi[\ldots]$ corresponding to the supremum. We extend VDGN with Envelop Q-learning [42], a multi-objective RL method that efficiently finds the convex envelope of the Pareto front in multi-objective MDPs; see Appendix A.3 for details. Once trained, $\mathbf{Q}^*$ induces the optimal policy for any preference ω according to the greedy policy $a^* = \text{argmax}_a\, \omega^T \mathbf{Q}^*(s, a, \omega)$.

6 EXPERIMENTAL SETUP

We designed experiments to test the ability of VDGN to find generalizable AMR strategies that display anticipatory refinement behavior, and benchmark these policies against standard baselines on error and DoF metrics. We define the FEM environment in Section 6.1, and the implementation of our method and baselines in Section 6.2 and Appendix C. Results are analyzed in Section 7.

6.1 AMR Environment

We use MFEM [2, 19] and PyMFEM [19], a modular open-source library for FEM, to implement the Markov game for AMR. We ran experiments on the linear advection equation $\frac{\partial u}{\partial t} + \nu \nabla \cdot u = 0$ with random initial conditions (ICs) for velocity ν and solution $u(0)$, solving it using the FEM framework on a two-dimensional L^2 finite

element space with periodic boundary conditions. Each discrete step of the Markov game is a mesh re-grid step, with τ_{step} FEM simulation time elapsing between each consecutive step. The solution is represented using discontinuous first-order Lagrange polynomials, and the initial mesh is partitioned into $n_x \times n_y$ quadrilateral elements. Appendix C contains further FEM details on the mesh partition and the construction of element observations.

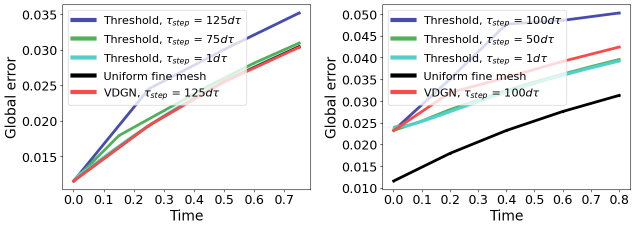
Linear advection is a useful benchmark for AMR despite its seeming simplicity because the challenge of anticipatory refinement can be made arbitrarily hard by increasing the τ_{step} of simulation time that elapses between two consecutive steps in the Markov game (i.e., between each mesh update step). Intuitively, an optimal refinement strategy must refine the entire connected region that covers the path of propagation of solution features with large solution gradients (i.e., high error on a coarse mesh), and maintain coarse elements everywhere else. Hence, the larger the τ_{step}, the harder it is for distant elements, which currently have low error but will experience large solution gradients later, to refine preemptively. But such long-distance preemptive refinement capability is exactly the key for future applications in which one prefers to have few re-meshing steps during a simulation due to its computational cost. Moreover, the existence of an analytic solution enables us to benchmark against error threshold-based baselines under the ideal condition of having access to perfect error estimator.

Metric. Besides analyzing performance via error c and cumulative DoFs d individually, we define an efficiency metric as $\eta = 1 - \sqrt{\tilde{c}^2 + \tilde{d}^2} \in [0, 1]$, where a higher value means higher efficiency. Here, $\tilde{c} := \frac{c - c_{\text{fine}}}{c_{\text{coarse}} - c_{\text{fine}}}$ is a normalized solution error and $\tilde{d} := \frac{d - d_{\text{coarse}}}{d_{\text{fine}} - d_{\text{coarse}}}$. is normalized cumulative degrees of freedom (a measure of computational cost). Here, the subscripts "fine" and "coarse" indicate that the quantity is computed on a constant uniformly fine and coarse mesh, respectively, that are held static (not refined/de-refined) over time. The uniformly fine and coarse meshes themselves attain efficiency $\eta = 0$. Efficiency $\eta = 1$ is unattainable in principle, since non-trivial problems require $d > d_{\text{coarse}}$.

6.2 Implementation and Baselines

The graph attention layer of VDGN was constructed using the Graph Nets library [5]. We used hidden dimension 64 for all VDGN layers (except output layer of size $|\mathcal{A}|$), and $H = 2$ attention heads. For $\text{depth}_{\max} = 1$, with initial $n_x = n_y = 16$, we chose $L = 2$ internal layers, with $R = 3$ recurrent passes. Each Markov game step has $\tau_{\text{step}} = 0.25$, $\tau_f = 0.75$ (hence $T_{\max} = 3$). For $\text{depth}_{\max} = 2$, with $n_x = n_y = 8$, we used $L = R = 2$. Each Markov game step has $\tau_{\text{step}} = 0.2$, $\tau_f = 0.8$ (hence $T_{\max} = 4$). For each training episode, we uniformly sampled the starting position and velocity of a 2D isotropic Gaussian wave as the initial condition. The FEM solver time discretization was $d\tau = 0.002$ throughout. See Appendix C for further architectural details and hyperparameters.

We compare with the class of local error-based **Threshold** policies, each member of which is defined by a tuple $(\theta_r, \theta_d, \tau_{\text{step}})$ as follows: every τ_{step} of simulation time, all elements whose true error exceed θ_r are refined, while those with true error below θ_d are de-refined. These policies represent the ideal behavior of widely-used AMR methods based on local error estimation, in the limit of perfectly accurate error estimation.



(a) $\text{depth}_{\max} = 1$ **(b) $\text{depth}_{\max} = 2$**

Figure 4: Global error versus simulation time of VDGN, compared with Thresholld policies with different τ_{step} elapsed between each mesh update step. (a) VDGN with the longest duration $\tau_{\text{step}} = 125d\tau$ has error growth comparable to Threshold with the shortest duration $\tau_{\text{step}} = 1d\tau$. (b) VDGN significantly outperforms its Threshold counterpart with $\tau_{\text{step}} = 100d\tau$.

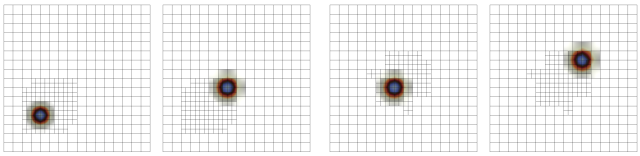


Figure 5: At each RL step, VDGN refines the full path segment that will be traversed by the wave over many solver steps into the future. Here, and for all subsequent mesh visualizations, we show the process: refinements, solution after τ_{step}, refinements, and so on.

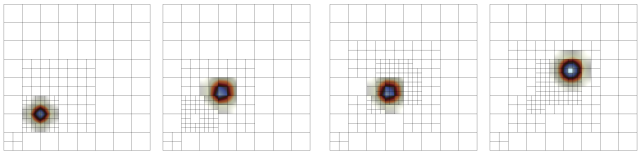
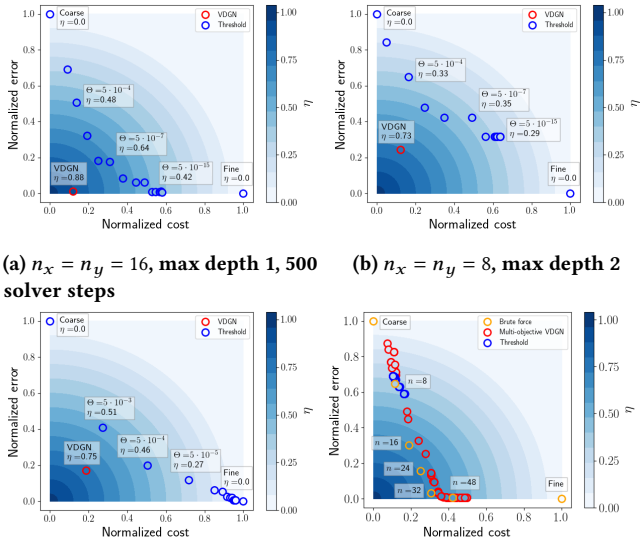


Figure 6: VDGN chooses more level-1 refinement than necessary for the wave's location at $t+1$, so that level-2 refinement is possible for the wave's location at $t+2$.

Remark 3. *Crucially, note that the Threshold policy class does not necessarily contain the global optimal policy for all AMR problems because such policies are incapable of anticipatory refinement and cannot access the full error-cost objective landscape. Suppose an element i with flat features and negligible error $c_t^i \ll 1$ at time t needs to refine before the arrival of complex PDE features at $t+1$. If $\theta_r > c_t^i$, then element i is not refined preemptively and large error is incurred at $t+1$. If $\theta_r < c_t^i$, then many other elements j with error $c_t^j > \theta_r$ are also refined at t but they may not contain complex features at $t+1$, so DoF cost is unnecessarily increased.*

7 EXPERIMENTAL RESULTS

Overall, we find that VDGN policies display anticipatory refinement, generalize to different initial conditions, mesh resolutions and simulation durations, thereby uncovering Pareto-efficient regions of the error-cost trade-off that were previously inaccessible by traditional error-estimator-based methods. VDGN policy runtimes are comparable to Threshold policies (see Table 4)



(a) $n_x = n_y = 16$**, max depth 1, 500 solver steps**

(b) $n_x = n_y = 8$**, max depth 2**

(c) $n_x = n_y = 16$**, max depth 1, 2500 solver steps.**

(d) Pareto front of multi-objective VDGN.

Figure 7: Trade-off between error and computational cost. VDGN unlocks regions inaccessible by threshold policies.

7.1 Anticipatory Refinement

As discussed in Section 6.1, each mesh update incurs a computational cost, which means AMR practitioners prefer to have a long duration of simulation time between each mesh update step. Figure 4 shows the growth of global error versus simulation time of VDGN and Threshold policies with different τ_{step} between each mesh update step. In the case of $\text{depth}_{\text{max}} = 1$, VDGN was trained and tested using the longest duration $\tau_{\text{step}} = 125d\tau$ (i.e., it has the fewest mesh updates), but it matches the error of the most expensive threshold policy that updates the mesh after each $\tau_{\text{step}} = 1d\tau$ (see Figure 4a). This is possible only because VDGN preemptively refines the contiguous region that will be traversed by the wave within $125d\tau$ (e.g., see Figure 5). In contrast, Threshold must update the mesh every $1d\tau$ to achieve this performance, since coarse elements that currently have negligible error due to their distance from the incoming feature do not refine before the feature's arrival.

Moreover, in the case of $\text{depth}_{\text{max}} = 2$, the agents learned to choose level-1 refinement at $t = 1$ for a region much larger than the feature's periphery, so that these level-1 elements can preemptively refine to level 2 at $t = 2$ before the feature passes over them. This is clearly seen in Figure 6. This enabled VDGN with $\tau_{\text{step}} = 100d\tau$ (fewest update steps) to have error growth rate close to that of Threshold with $\tau_{\text{step}} = 1d\tau$ (see Figure 4b).

Symmetry. Comparing Figure 12 with Figure 5, we see that VDGN policies are equivariant to rotation of initial conditions. Reflection equivariance is also visible for the opposite moving waves in Figure 10. Translation equivariance can be seen in Figure 13 Note that perfect symmetry holds only for rotation by integer multiples of $\pi/2$ and translation by integer multiples of the width of a level-0 element. Symmetry violation from mesh discretization is unavoidable for other values.

7.2 Pareto Optimality

Figure 7 shows that VDGN unlocks regions of the error-cost landscape that are inaccessible to the class of Threshold policies in all of the mesh configurations that were tested. We ran a sweep over refinement threshold $\theta_r \in [5\times10^{-3}, \ldots, 5\times10^{-8}, 5\times10^{-15}]$ with de-refinement threshold $\theta_d = 4 \times 10^{-15}$. In the case of $\text{depth}_{\text{max}} = 1, 2$ with 500 solver steps, and $\text{depth}_{\text{max}} = 1$ with 2500 solver steps, Figures 7a to 7c show that VDGN lies outside the empirical Pareto front formed by threshold-based policies, and that VDGN Pareto-dominates those policies for almost every value of θ_r: given a desired error (cost), VDGN has much lower cost (error). Table 1 shows that VDGN has significantly higher efficiency than Threshold policies for all tested threshold values, for $\text{depth}_{\text{max}} = 1, 2$.

To understand the optimality of VDGN policies, we further compared multi-objective VDGN to brute-force search for the best sequence of refinement actions in an anisotropic 1D advection problem with $n_x = 64, n_y = 1$ and two mesh update steps. To make brute-force search tractable, we imposed the constraint that a contiguous region of n elements are refined at each step (while all elements outside the region are de-refined). We searched for the starting locations of the region that resulted in lowest final global error. By varying n, this procedure produces an empirical Pareto front of such brute-force policies in the error-cost landscape, which we plot in Figure 7d. For multi-objective VDGN, we trained a single policy and evaluated it with 100 randomly sampled preferences $\omega = [\alpha, 1 - \alpha]$ where $\alpha \sim \text{Unif}[0, 1]$. Figure 7d shows that a single multi-objective VDGN policy produces a Pareto front (o) that approaches the Pareto front formed by brute force policies (o). Moreover, we see that Threshold policies with various refinement thresholds (o) are limited to a small section of the objective landscape, whereas VDGN unlocks previously-inaccessible regions.

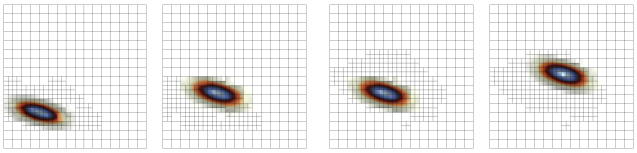


Figure 8: Policy trained on isotropic 2D Gaussian can be applied to anisotropic 2D Gaussian.

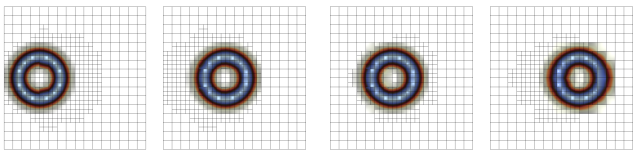


Figure 9: Policy trained on isotropic 2D Gaussian can be applied to ring functions.

7.3 Generalization

Longer time. At training time for VDGN, each episode consisted of approximately 400-500 FEM solver steps. We tested these policies on episodes with 2500 solver steps, which presents the agents with features outside of its training distribution due to accumulation of numerical error over time. Table 1 shows that: a) VDGN maintain highest top performance; b) the performance ratio between VDGN

Table 1: Mean and standard error of efficiency of VDGN versus local error threshold-based policies, over 100 runs with uniform random ICs. Policies used in generalization tests were trained on isotropic Gaussian features on a square mesh with quadrilateral elements for less than $500d\tau$ per episode. Last row shows the efficiency ratio of VDGN to the best threshold policy.

	In-distribution			Generalization					
θ_r	Depth 1 375 steps	Depth 2 400 steps	Triangular 1250 steps	Depth 1 2500 steps	Depth 2 2500 steps	Anisotropic 2500 steps	Ring 2500 steps	Opposite 2500 steps	Star 750 steps
5×10^{-3}	0.35 (0.01)	0.35 (0.01)	0.69 (0.02)	0.52 (0.02)	0.41 (0.01)	0.62 (0.02)	0.61 (0.02)	0.56 (0.002)	0.85 (0.01)
5×10^{-4}	0.74 (0.02)	0.51 (0.01)	0.76 (0.01)	0.57 (0.02)	0.43 (0.02)	0.64 (0.02)	0.56 (0.02)	0.61 (0.01)	0.92 (0.01)
5×10^{-5}	0.84 (0.01)	0.56 (0.01)	0.66 (0.02)	0.46 (0.02)	0.34 (0.02)	0.53 (0.02)	0.48 (0.02)	0.50 (0.01)	0.87 (0.01)
5×10^{-6}	0.82 (0.01)	0.55 (0.01)	0.56 (0.02)	0.37 (0.02)	0.26 (0.01)	0.42 (0.02)	0.39 (0.02)	0.42 (0.01)	0.83 (0.01)
5×10^{-7}	0.77 (0.01)	0.50 (0.01)	0.47 (0.02)	0.30 (0.02)	0.20 (0.01)	0.33 (0.02)	0.31 (0.02)	0.32 (0.01)	0.79 (0.01)
5×10^{-8}	0.70 (0.01)	0.44 (0.01)	0.38 (0.02)	0.24 (0.02)	0.15 (0.01)	0.26 (0.02)	0.25 (0.02)	0.26 (0.01)	0.74 (0.01)
5×10^{-15}	0.34 (0.01)	0.27 (0.01)	0.10 (0.01)	0.08 (0.01)	0.05 (0.003)	0.07 (0.01)	0.07 (0.01)	0.07 (0.01)	0.45 (0.02)
VDGN	**0.92** (0.01)	**0.66** (0.01)	**0.80** (0.01)	**0.84** (0.004)	**0.73** (0.01)	**0.78** (0.02)	**0.82** (0.01)	**0.63** (0.03)	**0.93** (0.01)
VDGN/best θ_r	1.10	1.18	1.05	1.47	1.70	1.22	1.34	1.03	1.13

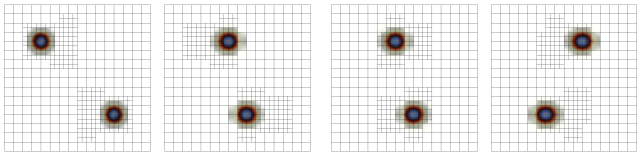


Figure 10: Policy trained on one bump with one velocity can be applied to two bumps with opposite velocities.

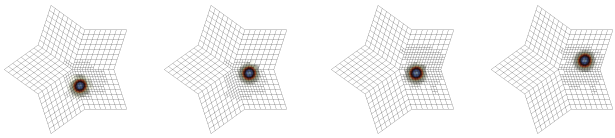


Figure 11: Policy trained on square mesh, run on a star mesh.

and the best Threshold policy actually increases in comparison to the case with shorter time (e.g., 1.47 in the "Depth 1 2500 steps" column, as opposed to 1.10 in the "Depth 1" column). This is because the error of threshold-based policies accumulates quickly over time due to the lack of anticipatory refinement, whereas VDGN mitigates the effect. Figure 16 shows that VDGN sustains anticipatory refinement behavior in test episodes longer than training episodes.

Out-of-distribution test problems. Even though VDGN policies were trained on square meshes with 2D isotropic Gaussian waves, we find that they generalize well to initial conditions and mesh geometries that are completely out of the training distribution. On anisotropic Gaussian waves (Figure 8), ring-shaped features (Figure 9), opposite-moving waves (Figure 10), star-shaped meshes (Figure 11), VDGN significantly outperforms Threshold policies without any additional fine-tuning or training (see the "Generalization" group in Table 1). Figure 15 shows qualitatively that a policy trained on quadrilateral elements shows rational refinement decisions when deployed on triangular elements.

Scaling. Since VDGN is defined by individual node- and edge-level computations with parameter-sharing across nodes and edges, it is a local model that is agnostic to size and scale of the global mesh. Figures 17 and 18 shows that a policy trained on $n_x = n_y = 16$ can be run with rational refinement behavior on an $n_x = n_y = 64$ mesh.

8 RELATED WORK

A growing body of work leverage machine learning and deep neural networks [12] to improve the trade-off between computational cost and accuracy of numerical methods: e.g., reinforcement learning for generating a fixed (non-adaptive) mesh [26], unsupervised clustering for marking and p-refinement [33], and supervised learning for target resolution prediction [23], error estimation [37], and mesh movement [30]. The following three are the closest work to ours. Yang et al. [41] proposed a global single-agent RL approach for h-adaptive AMR. It does not naturally support refining multiple elements per mesh update step, and anticipatory refinement was not conclusively demonstrated. Gillette et al. [11] work within the class of marking policies parameterized by an error threshold and showed that single-agent RL finds robust policies that dynamically choose the error threshold and outperform fixed-threshold policies in elliptic problems. However, threshold-based policies may not contain the optimal policy for time-dependent problems that require anticipatory refinement. Foucart et al. [10] proposed a local single-agent RL approach whereby the agent makes a decision for one randomly-selected element at each step. At training time, the global solution is updated every time a single element action occurs; at test time, the agent faces a different environment transition since the global solution is updated only after it has acted for all elements. Our multi-agent approach enables the definition of the environment transition to be the same at training and test time.

9 CONCLUSION

We have formulated a Markov game for adaptive mesh refinement, shown that centralized training addresses the posthumous credit assignment problem, and proposed a novel multi-agent reinforcement learning method called Value Decomposition Graph Network (VDGN) to train AMR policies directly from simulation. VDGN displays anticipatory refinement behavior, enabling it to unlock new regions of the error-cost objective landscape that were inaccessible by previous threshold-based AMR methods. We verified that trained policies work well on out-of-distribution test problems with PDE features, mesh geometries, and simulation duration not seen in training. Our work serves as a stepping stone to apply multi-agent reinforcement learning to more complex problems in AMR.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 and the LLNL-LDRD Program with project tracking #21-SI-001. The authors thank Andrew Gillette for helpful discussions in the project overall. LLNL-PROC-841328.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. 2021. MFEM: A modular finite element methods library. *Computers & Mathematics with Applications* 81 (2021), 42 – 74. Development and Application of Open-source Software for Problems with Numerical PDEs.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[4] Wolfgang Bangerth and Rolf Rannacher. 2013. *Adaptive finite element methods for differential equations*. Birkhäuser.

[5] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[6] Susanne Brenner and Ridgway Scott. 2007. *The mathematical theory of finite element methods*. Vol. 15. Springer Science & Business Media.

[7] Jakub Červený, Veselin Dobrev, and Tzanio Kolev. 2019. Nonconforming mesh refinement for high-order finite elements. *SIAM Journal on Scientific Computing* 41, 4 (Jan. 2019), C367–C392. https://doi.org/10.1137/18m1193992

[8] Andrew Cohen, Ervin Teng, Vincent-Pierre Berges, Ruo-Ping Dong, Hunter Henry, Marwan Mattar, Alexander Zook, and Sujoy Ganguly. 2021. On the use and misuse of absorbing states in multi-agent reinforcement learning. *arXiv preprint arXiv:2111.05992* (2021).

[9] Jakob N Foerster. 2018. *Deep multi-agent reinforcement learning*. Ph.D. Dissertation. University of Oxford.

[10] Corbin Foucart, Aaron Charous, and Pierre FJ Lermusiaux. 2022. Deep Reinforcement Learning for Adaptive Mesh Refinement. *arXiv preprint arXiv:2209.12351* (2022).

[11] Andrew Gillette, Brendan Keith, and Socratis Petrides. 2022. Learning robust marking policies for adaptive mesh refinement. *arXiv preprint arXiv:2207.06339* (2022).

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[13] Hado Hasselt. 2010. Double Q-learning. *Advances in neural information processing systems* 23 (2010).

[14] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. 2022. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022), 1–59.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[16] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. 2019. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 33, 6 (2019), 750–797.

[17] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*.

[18] Michael L Littman. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*. Elsevier, 157–163.

[19] mfem [n.d.]. MFEM: Modular Finite Element Methods [Software]. mfem.org. https://doi.org/10.11578/dc.20171025.1248

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[21] Peter Monk et al. 2003. *Finite element methods for Maxwell's equations*. Oxford University Press.

[22] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, Vol. 99. 278–287.

[23] Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowlishwaran. 2022. NUNet: Deep Learning for Non-Uniform Super-Resolution of Turbulent Flows. *arXiv preprint arXiv:2203.14154* (2022).

[24] Frans A Oliehoek and Christopher Amato. 2016. *A concise introduction to decentralized POMDPs*. Springer.

[25] Guillermo Owen. 1982. *Game Theory: Second Edition*. Academic Press, Orlando, Florida.

[26] Jie Pan, Jingwei Huang, Gengdong Cheng, and Yong Zeng. 2022. Reinforcement learning for automatic quadrilateral mesh generation: a soft actor-critic approach. *arXiv preprint arXiv:2203.11203* (2022).

[27] Junuthula Narasimha Reddy and David K Gartling. 2010. *The finite element method in heat transfer and fluid dynamics*. CRC press.

[28] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[29] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. In *International Conference on Learning Representations*.

[30] Wenbin Song, Mingrui Zhang, Joseph G Wallwork, Junpeng Gao, Zheng Tian, Fanglei Sun, Matthew D Piggott, Junqing Chen, Zuoqiang Shi, Xiang Chen, et al. 2022. M2N: Mesh Movement Networks for PDE Solvers. *arXiv preprint arXiv:2204.11188* (2022).

[31] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2085–2087.

[32] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[33] Kenza Tlales, Kheir-Eddine Otmani, Gerasimos Ntoukas, Gonzalo Rubio, and Esteban Ferrer. 2022. Machine learning adaptation for laminar and turbulent flows: applications to high order discontinuous Galerkin solvers. *arXiv preprint arXiv:2209.02401* (2022).

[34] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[37] Joseph G Wallwork, Jingyi Lu, Mingrui Zhang, and Matthew D Piggott. 2022. E2N: Error Estimation Networks for Goal-Oriented Mesh Adaptation. *arXiv preprint arXiv:2207.11233* (2022).

[38] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1995–2003.

[39] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[40] Jiachen Yang. 2021. *Cooperation in Multi-Agent Reinforcement Learning*. Ph.D. Dissertation. Georgia Institute of Technology.

[41] Jiachen Yang, Tarik Dzanic, Brenden Petersen, Jun Kudo, Ketan Mittal, Vladimir Tomov, Jean-Sylvain Camier, Tuo Zhao, Hongyuan Zha, Tzanio Kolev, et al. 2021. Reinforcement learning for adaptive mesh refinement. *arXiv preprint arXiv:2103.01342* (2021).

[42] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. 2019. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. *Advances in Neural Information Processing Systems* 32 (2019).

[43] Olgierd Cecil Zienkiewicz and Jian Zhong Zhu. 1992. The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique. *Internat. J. Numer. Methods Engrg.* 33, 7 (1992), 1331–1364.

A ADDITIONAL DETAILS ON METHODS

A.1 Multi-head attention

Multi-head graph attention layer. We extend graph attention by building on the mechanism of multi-head attention [35], which uses H independent linear projections $(W^{q,h}, W^{k,h}, W^{v,h}, W^{e,h})$ for queries, keys, values and edges (all projected to dimension d/H), and results in H independent sets of attention weights $\hat{a}^{ij,h}$, with $h = 1, 2, \ldots, H$. This enables the model to attend to different learned representation spaces and was previously found to stabilize learning [36]. The new node representation with multi-head attention is the concatenation of all output heads, with an output linear projection $W^o \in \mathbb{R}^{d \times d}$. Hence, eqs. (7) to (10) are extended to be:

$$\hat{a}^{ij,h} := \frac{\exp(W^{q,h}v^i \cdot W^{k,h}v^j)}{\sum_{l \in \mathcal{N}_i} \exp(W^{q,h}v^i \cdot W^k v^l)} \quad \text{for } (i,j) \in E, h \in [H], \tag{12}$$

$$b^{ij,h} := W^{v,h}v^j + W^{e,h}e^{ij} \quad \text{for } (i,j) \in E, h \in [H], \tag{13}$$

$$\hat{v}^i := W^o \left(\Big\|_{h=1}^{H} \sum_{j \in \mathcal{N}_i} \hat{a}^{ij,h} b^{ij,h} \right) \quad \text{for } i \in V. \tag{14}$$

A.2 Improvements to value-based learning

Standard Deep Q Network (DQN) [20] can be improved with double Q-learning, dueling network, and prioritized replay. Similarly, these improvements can be applied to VDN and VDGN as well.

Double Q-learning. DQN maintains a main network Q_θ with trainable parameters θ and a target network $Q_{\theta'}$ with separate trainable parameters θ', which is used in the TD-target

$$y_{t+1} := R_t + \gamma \max_a Q_{\theta'}(s_{t+1}, a). \tag{15}$$

After every update of the main parameters θ, the target parameters θ' are slowly updated via $\theta' \leftarrow \lambda\theta + (1-\lambda)\theta'$. In contrast, Double DQN [13, 34] uses the main network to compute the greedy policy, then uses the target network to evaluate the value of the greedy policy:

$$y_{t+1} := R_t + \gamma Q_{\theta'}(s_{t+1}, \operatorname*{argmax}_a Q_\theta(s_{t+1}, a)). \tag{16}$$

This alleviates the overestimation issue caused by using a single network for both selecting and evaluating the greedy policy. In the context of VDN and VDGN, we compute the TD target by using the target network $Q^i_{\theta'}$ to evaluate the greedy joint actions, which are selected by the main network Q^i_θ:

$$y_{t+1} := R_t + \gamma Q_{\theta'}(s_{t+1}, a)|_{a=[\operatorname{argmax}_{a^i} Q^i_\theta(s^i_{t+1}, a^i)]} \tag{17}$$

$$Q_\theta(s,a) := \sum_{i=1}^{n} Q^i_\theta(s^i, a^i) \quad Q_{\theta'}(s,a) := \sum_{i=1}^{n} Q^i_{\theta'}(s^i, a^i). \tag{18}$$

Dueling Network. In DQN, the neural network takes the state s as input and has $i = 1, \ldots, |\mathcal{A}|$ output nodes, each of which represents the scalar $Q(s, a = i)$ for $|\mathcal{A}|$ discrete actions. Dueling networks [38] use the fact that sometimes it suffices to estimate the value of a state without estimating the value of each action at the state, which implies that learning a value function $V^\pi(s,a)$ is enough. Practically, this separation of state-action values from state values is achieved by parameterizing $Q_{\phi,\alpha,\beta}(s,a)$ as a sum of value function $V_{\phi,\beta}(s)$ and advantage function $A_{\phi,\alpha}(s,a)$:

$$Q_{\phi,\alpha,\beta}(s,a) = V_{\phi,\beta}(s) + A_{\phi,\alpha}(s,a), \tag{19}$$

where ϕ are shared parameters while α and β are separate parameters specific to A and V, respectively. In practice, the expression used is

$$Q_{\phi,\alpha,\beta}(s,a) = V_{\phi,\beta}(s) + \left(A_{\phi,\alpha}(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A_{\phi,\alpha}(s,a') \right), \tag{20}$$

for stability and identifiability reasons. Equation (20) is used as the form for both the main network with parameters $\theta := (\phi, \alpha, \beta)$ and target network with parameters $\theta' := (\phi', \alpha', \beta')$. Equation (20) is directly compatible with Double DQN Equation (17). It is also directly compatible with VDGN, where we use two separate final graph attention layers to output $V(s^i)$ and $A(s^i, a^i)$ for each node i.

Prioritized replay. DQN stores the online experiences into a finite rolling replay buffer $\mathcal{B}$ and periodically trains on batches of transitions that are sampled uniformly at random from $\mathcal{B}$. However, the frequency of experiencing a transition (which determines the count of occurrence in $\mathcal{B}$) should not necessarily determine the probability of sampling and training on that sample. This is because some transitions may have high occurrence frequency but low TD-error (if the Q values of the state are already well approximated), whereas other important transitions with high TD-error (due to insufficient training) may have low occurrence frequency in $\mathcal{B}$. Prioritized replay [29] defines the priority p_i of a transition $i = (s_i, a_i, r_i, s'_i)$ based on its TD-error $\delta_i := y_i - Q(s_i, a_i)$, and defines the probability of sampling transition i as

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{21}$$

where α is a hyperparameter that anneals between uniform sampling ($\alpha = 0$) and always sampling the highest priority transition ($\alpha \to \infty$). We use the rank-based variant of prioritized replay, whereby $p_i := \frac{1}{\operatorname{rank}(i)}$ and $\operatorname{rank}(i)$ is the rank of i when buffer $\mathcal{B}$ is sorted according to $|\delta_i|$. We also use importance sampling correction weights $w_i := (1/N1/P(i))^\beta$, so that the agents learn with $w_i\delta_i$ rather than δ_i. This is an orthogonal improvement to any replay-based off-policy reinforcement learning method, and hence is directly compatible with both Double DQN and dueling network.

A.3 Multi-objective VDGN

Working in the context of linear preferences, the goal is to find a set of policies Π^* corresponding to the convex coverage set of the Pareto front, i.e., $\Pi^* := \{\pi \in \Pi \colon \exists \omega \in \Omega, \forall \pi' \in \Pi, \omega^T J^\pi \geq \omega^T J^{\pi'}\}$.

$$\Pi^* := \{\pi \in \Pi \colon \exists \omega \in \Omega, \forall \pi' \in \Pi, \omega^T J^\pi \geq \omega^T J^{\pi'}\} \tag{22}$$

To do so, we extend VDGN with Envelop Q-learning [42], a multi-objective RL method that efficiently finds the convex envelope of the Pareto front in multi-objective MDPs. Envelop Q-learning searches over the space of already-learned preferences $\omega' \in \Omega$ to find the

best TD target $\mathbf{y}(\omega)$ for a given preference ω. The single-objective VDN update equations eqs. (3) and (4) become

$$\theta \leftarrow \theta - \nabla_\theta \mathbb{E}_{s_t, a_t, s_{t+1}} \left[\|\mathbf{y}_{t+1} - \mathbf{Q}_\theta(s_t, a_t)\|_2^2 \right] \tag{23}$$

$$\mathbf{y}_{t+1}(\omega) := \mathbf{R}_t + \gamma \arg_Q \max_{a,\omega'} \omega^T \mathbf{Q}(s_{t+1}, a, \omega') \tag{24}$$

$$\max_{a,\omega'} \omega^T \mathbf{Q}(s, a, \omega') = \omega^T \mathbf{Q}(s, a)|_{a=[\operatorname{argmax}_{a^i,\omega'} \omega^T \mathbf{Q}^i(s^i, a^i, \omega')]_{i=1}^n} \tag{25}$$

where $\arg_Q \max_{a,\omega'} \omega^T \mathbf{Q}(s_{t+1}, a, \omega')$ extracts the vector $\mathbf{Q}(s_{t+1}, a^*, \omega^*)$ such that (a^*, ω^*) achieves the argmax. Note that the global argmax is still efficiently computable via value decomposition, since

$$\max_a \omega^T Q(s, a) == \sum_{i=1}^n \max_{a^i} \omega^T Q^i(s, a^i) \tag{26}$$

$$\Rightarrow \operatorname{argmax}_a \omega^T Q = \left[\operatorname{argmax}_{a^i} \omega^T Q^i(s, a^i) \right]_{i=1}^n \tag{27}$$

B PROOFS

Proposition 1. *Let $\mathcal{M}$ denote a multi-agent environment where the number of agents $n_t = 1, \ldots, N_{max}$ can change at every time step $t = 0, 1, \ldots, T_{max}$ due to agent-creation and agent-deletion actions in each agent's action space. At each time t, the environment is defined by the tuple $\left(\{\mathcal{S}^i\}_{i=1}^{n_t}, \{\mathcal{A}^i\}_{i=1}^{n_t}, R, P, \gamma, n_t, N_{\max}\right)$. $\mathcal{M}$ can be written as a Markov game with a stationary global state space and joint action space that do not depend on the number of currently existing agents.*

Proof. We prove this by construction. First define a dummy individual agent state denoted $s_\nexists$ that is interpreted semantically as "nonexistence". Define the global state space $\mathcal{S}$ to be $\mathcal{S} := \prod_{i=1}^{N_{\max}} (\mathcal{S}^i \cup \{s_\nexists\})$. Hence, any arbitrary state s_t, with n_t number of agents, belongs to $\mathcal{S}$. More specifically, it belongs to the subspace $\left(\prod_{i=1}^{n_t} \mathcal{S}^i\right) \times \left(\prod_{i=n_t+1}^{N_{\max}} \{s_\nexists\}\right) \subset \mathcal{S}$.

Next, define the joint action space $\mathcal{A}$ to be $\mathcal{A} := \prod_{i=1}^{N_{\max}} \mathcal{A}^i$. At a time step during an episode when there are n_t elements, the effectively available action space is just $\prod_{i=1}^{n_t} \mathcal{A}^i$, meaning that the joint action is written as $a = (a^1, \ldots, a^{n_t}, \underbrace{\text{no-op}, \ldots, \text{no-op}}_{N_{\max}-n_t})$, where no-op is ignored by the reward and transition function. The reward for the joint state-action space is defined by

$$\begin{aligned} &R\big((s^1, \ldots, s^{n_t}, \underbrace{s_\nexists, \ldots, s_\nexists}_{N_{\max}-n_t}), (a^1, \ldots, a^{n_t}, \underbrace{\text{no-op}, \ldots, \text{no-op}}_{N_{\max}-n_t})\big) \\ &= R\big((s^1, \ldots, s^{n_t}), (a^1, \ldots, a^{n_t})\big), \end{aligned}$$

where we overload the use of notation R since R is already well-defined for variable number of agents. Similarly, the transition function P that was already well-defined for variable number of agents induces a transition function for the new state-action spaces $\mathcal{S}$ and $\mathcal{A}$. Hence the stationary Markov game is defined by the tuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma, N_{\max})$. This completes the construction. □

Proposition 2. *VDGN is equivariant to global rotations and translations of the error and velocity field, and it is time invariant.*

Proof. This is proved by induction on the number of graph attention layers.

Base case. In the original PDE, let $X = (x^1, \ldots, x^n)$ denote the coordinates of elements $i = 1, \ldots, n$, let $v = (v^1, \ldots, v^n)$ denote the input node features, and let $e = \{e^{ij}\}_{(i,j)\in E}$ denote the input edge features. Let element i with coordinate x^i have node feature v^i and edge feature set $\{e^{ij}\}_{j\in\mathcal{N}(i)}$. Suppose that the first layer of VDGN produces updated node features $\hat{v} = (\hat{v}^1, \ldots, \hat{v}^n)$. Any rotation can be represented by matrix multiplication with an orthogonal matrix $Q \in \mathbb{R}^{k\times k}$, where k is the number of spatial dimensions. Upon rotation of the PDE solution and velocity field, we check that the element at coordinate Qx^i produces the same node value as the value $\hat{v}^i$ that element i would have produced without the rotation. First, note that the displacement vector for any edge e and the velocity vector transform as $\hat{d}^e = Qd^e$ and $\hat{q} = Qq$, respectively. The node feature at Qx^i is v^i, since the PDE solution has been rotated. The edge feature set at Qx^i is $\{e^{ij}\}_{j\in\mathcal{N}(i)}$ because: a) by definition of rotation of the PDE, initial conditions are also rotated and hence the one-hot relative depth observation $\mathbf{1}[\text{depth}(i) - \text{depth}(j)]$ is unchanged; b) the inner product part of the edge feature is unchanged because $\langle \hat{d}^e, \hat{q}\rangle = \langle Qd^e, Qq\rangle = \langle d^e, q\rangle$ because orthogonality of Q implies $Q^TQ = 1$. Hence the input node and edge features at Qx^i in the rotated case are equal to those for element i in the original case, so the updated node feature at Qx^i is still $\hat{v}^i$. For translation, which is represented by addition with some vector $b \in \mathbb{R}^k$, the same reasoning applies by noting that displacements and velocities are invariance to addition, so the element at $x^i + b$ in the translated case has the same node and edge observations as the element x^i in the original case, so the updated node feature at $x^i + b$ is still $\hat{v}^i$.

Inductive step. Suppose the claim holds for layer $l-1$. This means that the node representation $\hat{v}^{l-1}$ (used as input to layer l) at element Qx^i in the transformed PDE is equal to the input node representation at element x^i in the original PDE. The input edge representation is still $\{e^{ij}\}_{j\in\mathcal{N}(i)}$ since edge representations are not updated, and the reasoning in the base case for edges applies. Hence, layer l's inputs in the transformed case are the same as its inputs in the original case, so the outputs $\hat{v}^l$ are also the same. This completes the proof for equivariance.

Time invariance. This follows immediately from the fact that absolute time is not used as an observable, and the fact that VDGN learns a Markov policy. □

C ADDITIONAL IMPLEMENTATION DETAILS

Ancillary algorithmic details. Every t_{train} environment steps, we sample batch size M transitions from the replay buffer $\mathcal{B}$ and conduct one training step. We used the Adam optimizer with learning rate η in Tensorflow [1]. Each agent has its own independent ϵ-greedy exploration with ϵ linearly decaying from ϵ_{start} to ϵ_{end} within n_ϵ episodes. Along with prioritized replay, we used a lower bound of p_{unif} on the probability of sampling each transition from the replay buffer. Hyperparameters for all algorithmic settings are listed in Table 2. We ran a simple population-based elimination search with 128 starting samples on the final exploration rate $\epsilon_{textend}$, learning rate η, and target update rate τ on one 16×16 mesh, then used those hyperparameters for all experiments. We used the same prioritized replay hyperparameters as the original paper. Graph attention hyperparameters L and R should be based

on the spatial domain of influence (e.g., number of elements along a direction) that the user wants to include to inform a refinement decision.

Table 2: Algorithm hyperparameters

Name	Symbol	Value
Batch size	M	16
Replay buffer size	$\|\mathcal{B}\|$	10000
Exploration decay episodes	n_ϵ	150000
Exploration end	ϵ_{end}	0.01
Exploration start	ϵ_{start}	0.5
Learning rate	η	5×10^{-4}
Prioritized replay exponent	α	0.5
Prioritized replay importance exponent	β	0.4
Env steps per train	t_{train}	4
Target update rate	λ	0.0112
Uniform sampling probability	p_{unif}	0.001
Attention layer size	d	64
Number of attention heads	H	2
Number of attention layers	L	2
Number of recurrent passes	R	depth 1: 3 depth 2: 2

Table 3: Environment settings

Name	Symbol	Value
DoF threshold	d_{thres}	5620
Initial error threshold	c_{thres}	5×10^{-4}
Solver time increment	$d\tau$	2×10^{-3}
Initial mesh partition	(n_x, n_y)	depth 1: $(16, 16)$ depth 2: $(8, 8)$
Training mesh dimension	$[s_x, s_y]$	$[0, 2.0]^2$
Solver time per mesh update	τ_{step}	depth 1: 0.25 depth 2: 0.2
Training max simulation time	τ_f	depth 1: 0.75 depth 2: 0.8

FEM settings. Hyperparameters for all environment settings are listed in Table 3. All simulations begin by applying level-1 refinement to elements whose initial error, which is available in general due to the known initial condition, is larger than c_{thres}. The training mesh length and width are $[s_x, s_y]$. We specified velocity by magnitude $u \in \mathbb{R}$ and angle $\theta \in [0, 1.0]$, so that velocity components are $v_x = u\cos(2\pi\theta)$ and $v_y = u\sin(2\pi\theta)$. We trained on 2D Gaussian functions

$$f(x, y, t) = 1 + \exp\left(-w((x - (x_0 + v_x t))^2 + (y - (y_0 + v_y t))^2)\right), \tag{28}$$

with initial conditions sampled from uniform distributions: $u \sim U[0, 1.5]$, $\theta \sim U[0, 1.0]$, $x_0, y_0 \sim U[0.5, 1.5]$, and $w = 100$. At test time, anisotropic Gaussians are specified by

$$f(x, y, t) = 1 + \exp\Big(-(w_x(x - dx)^2 + w_y(y - dy)^2 \tag{29}$$

$$+ w_{xy}(x - dx)(y - dy))\Big) \tag{30}$$

$$dx = x_0 + v_x t \tag{31}$$

$$dy = y_0 + v_y t \tag{32}$$

with $\theta \sim U[0, 1.0]$, $u \sim U[0, 1.5]$, $w_x, w_y, w_{xy} \sim U[20, 100]$, $x_0, y_0 \sim U[0.5, 1.5]$. Ring functions are

$$f(x, y, t) = 1 + \exp\left(-w(\sqrt{\Delta r^2} - r)^2\right) \tag{33}$$

$$\Delta r^2 = (x - (x_0 + v_x t))^2 + (y - (y_0 + v_y t))^2 \tag{34}$$

with $r \sim U[0.1, 0.3]$, $\theta \sim U[0, 1.0]$, $u \sim U[0, 1.5]$, $x_0, y_0 \sim U[0.5, 1.5]$. Opposite traveling Gaussians have the same form as above, with $u \sim U[0, 1.5]$, $w = 100$, the leftward moving Gaussian with $\theta = 0.5$, $x_0 \sim U[0.5, 1.5]$ and $y_0 \sim U[0.3, 0.7]$, while the rightward moving Gaussian has $\theta = 0.0$, $x_0 \sim U[0.5, 1.5]$ and $y_0 \sim U[1.3, 1.7]$. The star mesh test case uses a mesh from MFEM [2, Example 8], with $\theta \sim U[0.05, 0.25]$, $u \sim U[0, 6.0]$, $w = 5$, $x_0 = -1$ and $y_0 = -4$.

D ADDITIONAL RESULTS

Symmetry. In Figure 5, the initial conditions are $x_0 = y_0 = 0.5$ and $v_x = v_y = 1.5$. We rotated the initial conditions by π, so that $x_0 = y_0 = 1.5$ and $v_x = v_y = -1.5$, and we can see from Figure 12 that the global refinement choices are similarly rotated by π. Rotational equivariance can be also seen in Figure 10, where the refinement choices for the leftward and rightward moving waves are reflections of each other. Translational equivariance along the vertical and horizontal axes can be seen in Figure 13.

Runtime. Table 4 shows that the runtime of trained VDGN policies at test time is on the same order of magnitude as Threshold policies, for both average time per action (for all elements) and average time per episode.

Scaling. Figure 16 shows a policy trained with only three steps per episode (375 solver steps), running on 20 steps at test time (2500 solver steps). Figures 17 and 18 shows a policy trained on a 16×16 mesh with only three steps per episode, tested on a 64×64 mesh.

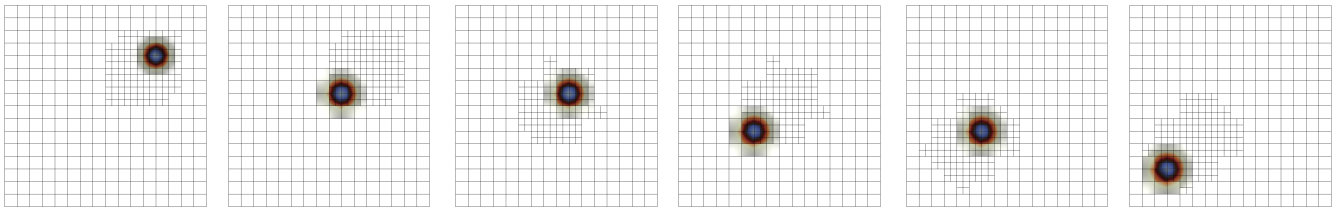
Using error estimator as observation. Table 5 shows that VDGN retains better performance than Threshold policies when the true error in agents' observations is replaced by the estimates given by the ZZ error estimator [43].

E OTHER FORMULATIONS AND THEIR DIFFICULTIES

E.1 Single-agent Markov decision process

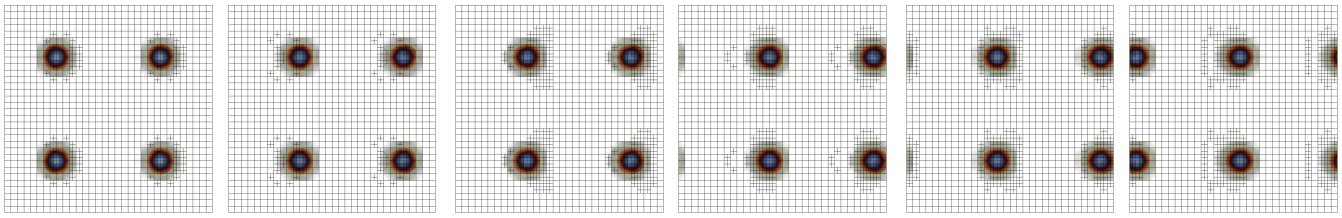
As mentioned in Section 8, previous work that treat AMR as a sequential decision-making problem took the single-agent viewpoint. This poses issues with producing multiple element refinements at each mesh update step.

In the global approach [41], a single agent takes the entire mesh state as input, chooses one element out of all available elements to refine (or potentially de-refine), and then the global solution is updated. To avoid computing the global solution after every refinement/de-refinement of a single element, which would be prohibitively expensive, an extension of this approach would have to use either a fixed hyperparameter or a fixed rule to determine the number of rounds of single-element selection between each



$t = 1$ $t = 2$ $t = 3$

Figure 12: Global refinements by VDGN are equivariant to rotation of the propagating feature. Compare to Figure 5

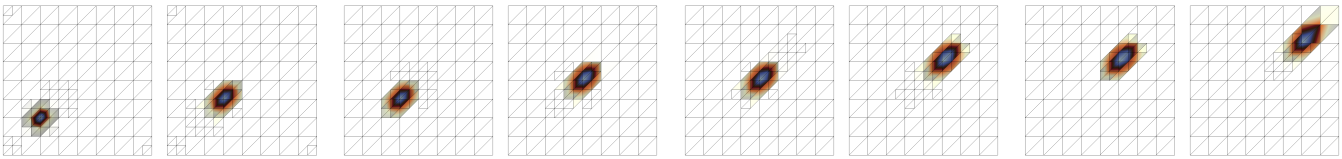


$t = 1$ $t = 2$ $t = 3$

Figure 13: Translational equivariance.

Table 4: Runtime in seconds of VDGN is comparable to local error threshold-based policies with best θ_r selected from Table 1. Average over 100 test episodes.

	In-distribution			Generalization					
	Depth 1 375 steps	Depth 2 400 steps	Triangular 1250 steps	Depth 1 2500 steps	Depth 2 2500 steps	Anisotropic 2500 steps	Ring 2500 steps	Opposite 2500 steps	Star 750 steps
Threshold (action)	0.009	0.005	0.005	0.015	0.012	0.014	0.014	0.021	0.005
Threshold (total)	1.18	0.84	2.15	12.9	11.0	11.9	12.2	18.1	1.30
VDGN (action)	0.024	0.011	0.005	0.026	0.011	0.008	0.008	0.008	0.008
VDGN (total)	1.96	1.18	2.26	15.0	8.23	17.3	18.5	21.6	3.43



$t = 1$ $t = 2$ $t = 3$ $t = 4$

Figure 14: Policy trained on triangular elements.

Table 5: Mean and standard error of efficiency of VDGN using ZZ error estimator values as observation, versus threshold-based policies, over 100 runs with uniform random ICs.

θ_r	Depth 1 375 steps	Depth 2 400 steps
5×10^{-3}	0.916 (0.007)	0.523 (0.021)
5×10^{-4}	0.899 (0.002)	0.529 (0.022)
5×10^{-5}	0.857 (0.003)	0.509 (0.021)
5×10^{-6}	0.813 (0.004)	0.474 (0.019)
5×10^{-7}	0.767 (0.004)	0.431 (0.018)
5×10^{-8}	0.718 (0.005)	0.390 (0.016)
5×10^{-15}	0.473 (0.006)	0.260 (0.014)
VDGN	**0.948** (0.002)	**0.543** (0.022)
VDGN/best θ_r	1.03	1.03

solution update. Specifying this hyperparameter or rule is hard to do in advance and may erroneously exclude the true optimal policy from the policy search space.

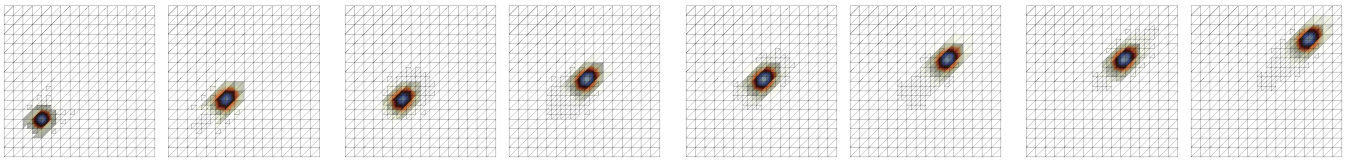


Figure 15: Policy trained on quadrilateral elements can be run on triangular elements.

In the local approach [10], a single agent is "centered" on an element and makes a refine/no-op/de-refine decision for that element. At training time, because each refine/de-refine action impacts the global solution, which directly impacts the reward, the environment transition involves a global solution update every time the agent chooses an action for an element. Since this is prohibitively expensive to do for larger number of elements at test time, [10] redefines the environment transition at test time so that the global solution is updated only after the agent chooses an action for all elements. This presents the agent with different definitions of the environment transition function at train and test time.

E.2 Fully-decentralized training and execution

In contrast to the approach of centralized training with decentralized execution used in this work, one may consider fully-decentralized training and execution. A reason for full decentralization is the fact that the governing equations of most physical systems in finite element simulations are local in nature, so one may hypothesize that an element only needs to respond to local observations and learn from local rewards. However, this poses two main challenges: 1) the class of policies accessible by fully-decentralized training with local rewards may not contain the global optimum joint policy; 2) for h-adaptive mesh refinement with maximum depth greater than one, one must confront the posthumous credit assignment problem: a refinement or de-refinement action may have long-term impact on the global error but the agent responsible for that action immediately disappears upon taking the action and hence does not receive future rewards.

For the second challenge in particular, we can easily construct a scenario in which an agent who only receives an immediate reward and disappears upon refinement (i.e., does not persist) cannot learn anticipatory refinement. We can construct a scenario where an agent cannot make an optimal refinement action: 1) use a moving feature such that error is minimized by the sequence of actions: $a_t^i = 1$ at time t, and $a_{t+1}^j = 1, \forall j \in \text{Desc}(i)$ at time $t+1$; 2) construct the feature so that action $a_t^i = 1$ has no instantaneous benefit for error reduction, meaning that the instantaneous reward r_t^i for the state-action pair $(o_t^i, a_t^i = 1)$ is the same as the reward for the pair $(o_t^i, a_t^i = 0)$. Since agent i's trajectory terminates upon receiving reward r_t^i, this means agent i cannot distinguish the value of refinement versus no-op. This problem may be overcome by letting all agents persist after refinement, receive dummy observations and local reward but take no action, so that they can learn the past action's impact on future local reward. Another way is to let the agent disappear upon refinement/de-refinement at transition (s_t, a_t, r_t) but use the final reward r_T (e.g., upon episode termination at time $t = T$) as the reward for the transition (s_t, a_t, r_T).

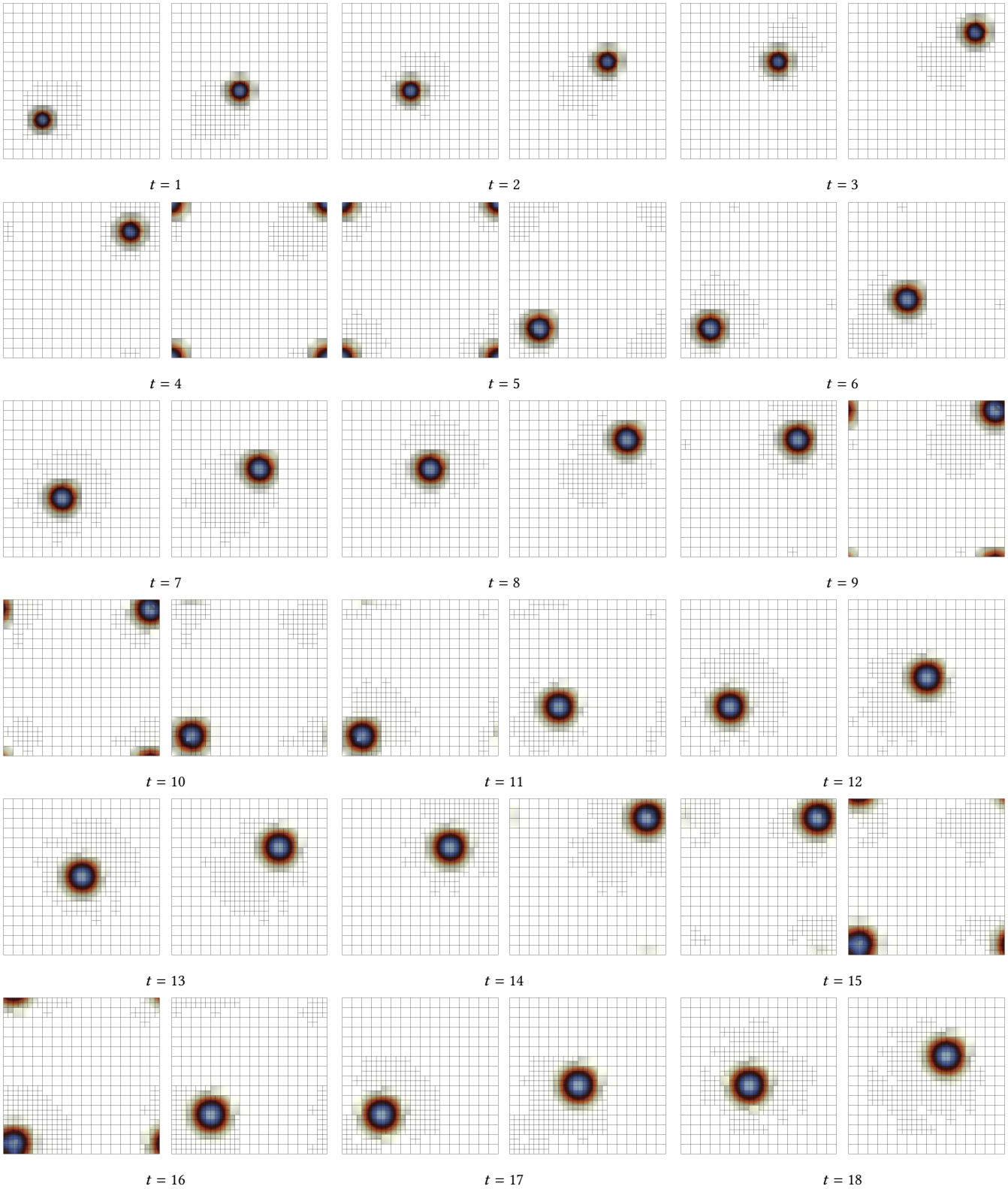
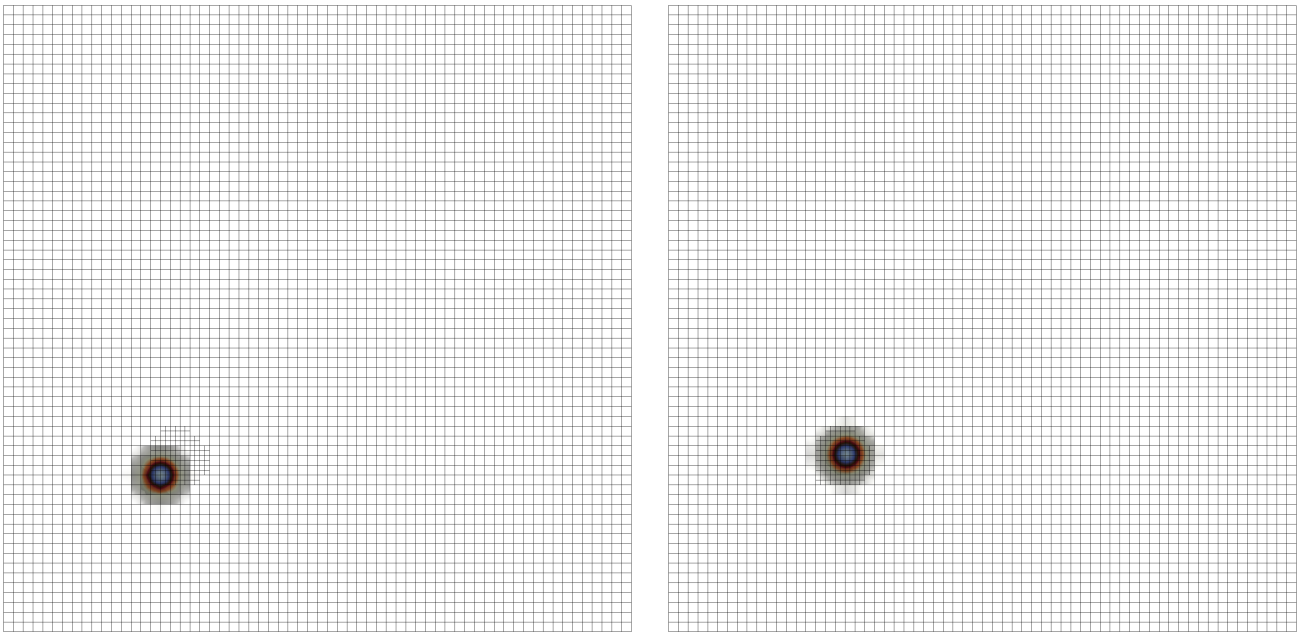
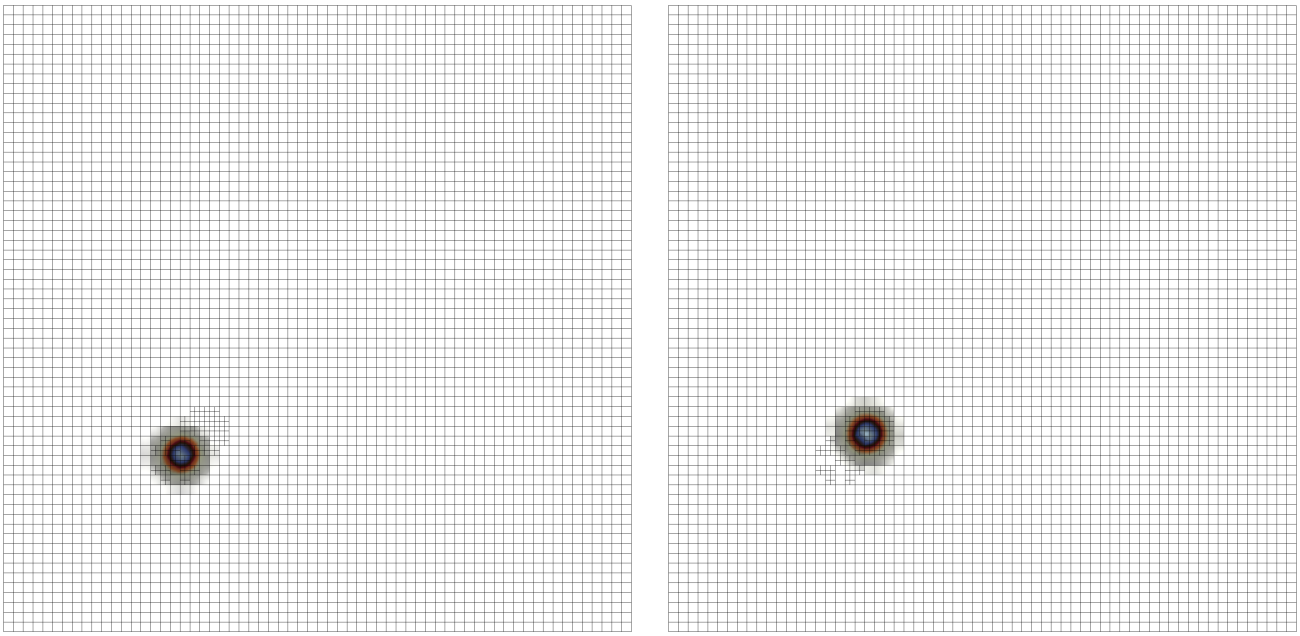


Figure 16: VDGN generalizes to longer simulation durations, despite seeing only up to $t = 3$ during training episodes.



$t = 1$



$t = 2$

Figure 17: VDGN generalizes to larger meshes not seen in training. See Figure 18 for subsequent time steps.

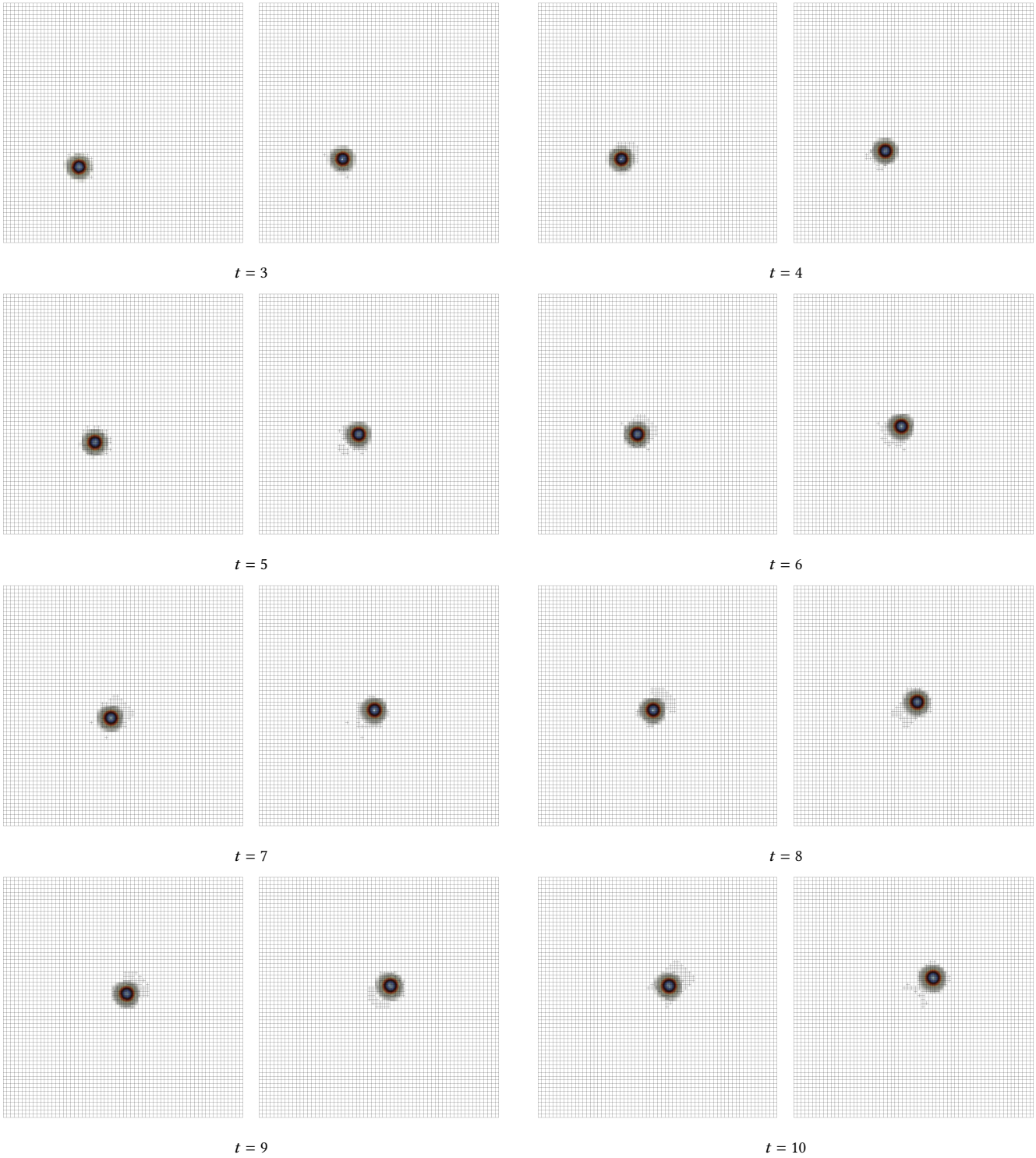


Figure 18: Continued from Figure 17. VDGN generalizes to larger meshes not seen in training.