Journal of Computational Physics 510 (2024) 113061

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

GrainGNN: A dynamic graph neural network for predicting 3D grain microstructure ☆

Yigong Qin [a,*], Stephen DeWitt [b], Balasubramaniam Radhakrishnan [b], George Biros [c]

[a] Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712, USA
[b] Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[c] Oden Institute, The University of Texas at Austin, Austin, TX 78712, USA

ARTICLE INFO

Keywords:
Grain microstructure evolution
Deep learning
Graph neural network
Phase field simulations
Additive manufacturing

ABSTRACT

We propose GrainGNN, a surrogate model for the evolution of polycrystalline grain structure under rapid solidification conditions in metal additive manufacturing. High fidelity simulations of solidification microstructures are typically performed using multicomponent partial differential equations (PDEs) with moving interfaces. The inherent randomness of the PDE initial conditions (grain seeds) necessitates ensemble simulations to predict microstructure statistics, e.g., grain size, aspect ratio, and crystallographic orientation. Currently such ensemble simulations are prohibitively expensive and surrogates are necessary.

In GrainGNN, we use a dynamic graph to represent interface motion and topological changes due to grain coarsening. We use a reduced representation of the microstructure using hand-crafted features; we combine pattern finding and altering graph algorithms with two neural networks, a classifier (for topological changes) and a regressor (for interface motion). Both networks have an encoder-decoder architecture; the encoder has a multi-layer transformer long-short-term-memory architecture; the decoder is a single layer perceptron.

We evaluate GrainGNN by comparing it to high-fidelity phase field simulations for in-distribution and out-of-distribution grain configurations for solidification under laser power bed fusion conditions. GrainGNN results in 80%–90% pointwise accuracy; and nearly identical distributions of scalar quantities of interest (QoI) between phase field and GrainGNN simulations compared using Kolmogorov-Smirnov test. GrainGNN's inference speedup (PyTorch on single x86 CPU) over a high-fidelity phase field simulation (CUDA on a single NVIDIA A100 GPU) is 150×–2000× for 100-initial grain problem. Further, using GrainGNN, we model the formation of 11,600 grains in 220 seconds on a single CPU core.

1. Introduction

Let $\theta(x, y, z, t)$ be the crystal orientation vector as a function of space (x, y, z) and time t. We assume we are given a high fidelity model $\mathcal{F}$ such that $\theta(x, y, z, t) = \mathcal{F}(\boldsymbol{p}, \boldsymbol{\xi}, L)$. Here $\boldsymbol{p}$ encodes the meltpool temperature field using two scalars, G and R, where G

* Corresponding author.
E-mail addresses: ygqin@utexas.edu (Y. Qin), dewittsj@ornl.gov (S. DeWitt), radhakrishnb@ornl.gov (B. Radhakrishnan), gbiros@acm.org (G. Biros).

https://doi.org/10.1016/j.jcp.2024.113061
Received 28 January 2024; Accepted 26 April 2024

Available online 7 May 2024

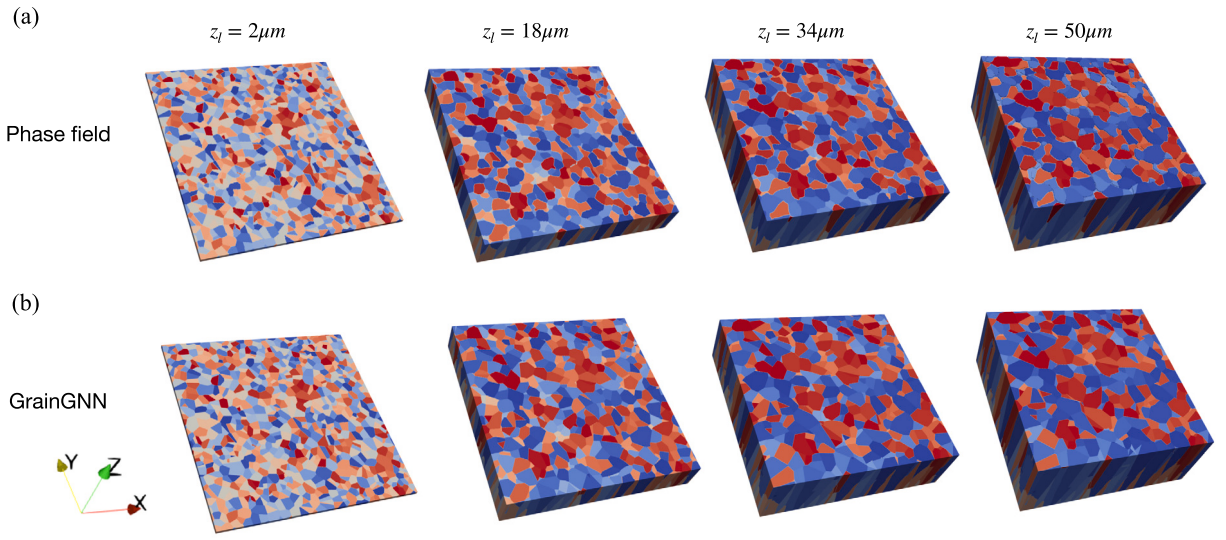

Fig. 1. Time evolution of the solidified microstructure of stainless steel 316L predicted by a phase field model and GrainGNN. (a) Phase field simulation. (b) GrainGNN predictions. z_l is the current height of the solid-liquid interface (SLI). The domain size is (120 μm, 120 μm, 50 μm). The microstructure of 900 grains at $z_l = 2$ μm is the input to both phase field simulation and GrainGNN. We apply a temperature field with $G = 10$ K/μm and $R = 2$ m/s. The grains are growing in the z-direction and stopped at $z_l = 50$ μm. The elapsed simulation time for the phase field model is 48 minutes on one Nvidia A100 GPU. The GrainGNN inference time is 19 seconds on one Intel x86 6-core CPU.

is a scalar metric of the temperature gradient and R is the pulling velocity (cooling rate divided by G); $\boldsymbol{\xi}$ parameterizes the initial microstructure $\boldsymbol{\theta}(x, y, z, t = 0)$ at the solid-liquid interface (SLI); and $L = (L_x, L_y, L_z)$ parameterizes the meltpool size, which we idealize as a 3D rectangular volume with grain epitaxial growth aligned with the z axis (see Fig. 1). In this context, $\boldsymbol{\theta}$ is a piecewise constant function and grains are defined as connected regions with the same $\boldsymbol{\theta}$; $\boldsymbol{\xi}$ is a random field, and quantities of interest (QoIs) that depend on $\boldsymbol{\theta}$ can only be statistically characterized and require expectations over $\boldsymbol{\xi}$.

Evaluating $\mathscr{F}$ can be extremely expensive as it involves multicomponent, multiscale PDEs with moving interfaces. Furthermore, additive manufacturing (AM) involves a large number of meltpool simulations: the length scale of an AM component can be of the order of meters while the meltpool dimensions are $\mathcal{O}(100$ μm). *In this context, we seek to find a surrogate model* $\widetilde{\boldsymbol{\theta}} = \widetilde{\mathscr{F}}(\boldsymbol{p}, \boldsymbol{\xi}, L)$ *that approximates* $\boldsymbol{\theta}$ *and the statistics of the quantities of interest. We also require that* $\widetilde{\mathscr{F}}$ *is much cheaper to compute than* $\mathscr{F}$.

Background and significance: High fidelity models are important tools for predicting grain structure formation under rapid solidification conditions existing during AM [1]. Common numerical methods include phase field models [2–6], cellular automata [7–9], and kinetic Monte Carlo methods [10]. However, these methods are computationally expensive as they require fine spatial and temporal discretizations. Furthermore, due to the stochastic nature of AM process, ensemble simulations [6,11–13] are required to statistically characterize quantities of interest (QoIs) like grain size, aspect ratio, and misorientation [14–16]. Tasks such as process control and optimization require repeated invocation of high fidelity models under variable heating source conditions, feedstock composition, and component geometry. These high computational costs have restricted the adoption of high fidelity simulations for predicting or controlling the grain structure during AM processes.

Summary of proposed methodology and contributions: Here, following our work in 2D surrogates [17], we focus on epitaxial grain growth from the substrate and ignoring grain nucleation and the evolution of the solute concentration field that contributes to intragranular features of the solidification microstructure including primary and secondary arms, microsegregation, etc. The key phenomena that need to be captured at this modeling level are grain envelope evolution and grain coarsening through competitive growth. The latter can be decomposed into grain face elimination and grain elimination [18,19]. The dynamics of $\mathscr{F}$ are quite challenging to capture due to the evolving interface and the topological changes due to coarsening.

In GrainGNN (Section 2.2), first we reduce the representation of $\boldsymbol{\theta}(x, y, z, t)$ using aggressive coarsening in space and time. (i) We introduce a reduced graph representation of $\boldsymbol{\theta}$ that captures grain boundaries, size, edge length, and other geometric features per grain (Section 2.3). (ii) Following [18], we introduce a categorization of topological events that need to be tracked (Section 2.4). (iii) We use two neural networks to model their dynamics: a regression network to capture the interface motion; and a classification network to estimate probabilities of topology-changing events (Section 2.5). (iv) We introduce graph algorithms that combine the network outputs to evolve the microstructure graph topology (Section 2.6). (v) We introduce an algorithm that reconstructs $\widetilde{\boldsymbol{\theta}}(x, y, z, t)$ from the reduced graph/feature-based representation of the microstructure (Section 2.7).

GrainGNN is calibrated (trained) using pairs $\{\boldsymbol{\theta}_k(x, y, z_{l-1}, t_{l-1}), \boldsymbol{\theta}_k(x, y, z_l, t_l)\}_k$, i.e., cross-sections of the microstructure at SLI heights z_{l-1} and z_l observed at times t_{l-1} and t_l with $t_l > t_{l-1}$; and $\boldsymbol{\theta}_k(x, y, z, t) = \mathscr{F}(\boldsymbol{p}_k, \boldsymbol{\xi}_k, L_k)$, where k indicates sampling of AM conditions and initial grain microstructure. We used $k \approx$40K obtained from ∼1,500 high fidelity simulations.

Using these steps, GrainGNN (Algorithm 1) predicts 3D grain microstructure formation under additive manufacturing conditions. Given the grain orientation at $t = 0, z = 0$ with a liquid region for $z > 0$ (see Fig. 2a) and the prediction is $\widetilde{\boldsymbol{\theta}}(x, y, z, t)$ (see Fig. 2b).

The error is measured by pointwise mismatch and quantities of interest that include the percentage of eliminated grains, grain size distribution, and volume-averaged misorientation.

Our contributions can be summarized as follows.

- We introduce a graph and hand-crafted features that greatly compress the spatial representations of the grain microstructure. We create $\boldsymbol{\theta}$-to-graph and graph-to-$\boldsymbol{\theta}$ mappings.
- We introduce two graph-transformer long-short-term-memory (LSTM) networks to predict graph feature evolution and grain topological events.
- We propose a graph update algorithm with $O(\#\text{grains})$ complexity to reconstruct the next graph with the output of GrainGNN. We generalize the algorithm to predict grain microstructure with unseen G, R, domain size, grain size distribution, and a larger number of grains.

Ground truth data is generated using an established phase-field-based model of epitaxial grain growth (Section 2.1). We remark that GrainGNN is independent of the underlying solver/formulation. We train GrainGNN assuming a probability density distribution for $\boldsymbol{\xi}$, a fixed $L = L^0$, and a grid-sampled $\boldsymbol{p}$ (Section 3.1). Then we evaluate the *in-distribution* generalization of GrainGNN in which we keep $L = L^0$, sample training-unseen values of $\boldsymbol{\xi}$ from the training distribution, and unseen values of $\boldsymbol{p}$ and measure pointwise errors of $\boldsymbol{\theta} - \widetilde{\boldsymbol{\theta}}$ as well errors in the statistics of QoIs (Section 3.2). We also test the *out-of-distribution* generalization GrainGNN with $L \neq L^0$ and $\boldsymbol{\xi}$ sampled from a different distribution (Section 3.3). We discuss the results, extensions, and limitations of GrainGNN in Section 4.

Related work: Surrogate models provide an alternative to high fidelity simulations, with the potential for reduced computational cost. Trained on either high fidelity simulation results or experimental results, such surrogates have been successful in addressing challenges in uncertainty quantification and optimization for a range of applications [20–23]. Specifically in the area of material microstructure prediction, machine learning surrogates have received increasing attention in the past few years. Convolutional neural networks have been used for 2D and 3D microstructure image reconstruction [24,25]. Generative adversarial networks [26] are capable of learning microstructure statistics, and can approximate structure-to-material properties forward and inverse maps [27,28]. Recurrent neural networks (RNNs) or its subclass long short-term memory (LSTM) networks [29] have been used to rapidly predict the time evolution of the microstructure. Existing works consider a representative volume element and use LSTM networks to capture evolution statistics for use cases such as: spinoidal decomposition of binary mixtures [30,31], brittle fracture [32], single dendrite growth [32], and grain formation [17]. Ref. [33] applied the convolutional autoencoder to map the microstructure of a two phase mixture to a latent space and used DeepONet [34] to learn the evolution.

Our previous work [17] introduced GrainNN, a transformer-based LSTM to predict 2D epitaxial grain growth during solidification. GrainNN reduces the computational cost by tracking only the grain boundaries instead of each grid point inside the grains. It evolves manually crafted grain shape descriptors defined on each grain. GrainNN can predict microstructure with low pointwise error while achieving significant speed up over high fidelity simulations. It also utilizes domain decomposition and rectangular-to-curvilinear domain mappings to handle systems of many grains and circular geometry. However, in this 2D setting the grain coupling is only one-dimensional and perpendicular to the temperature gradient, which makes GrainNN hard to model the substantially more complex grain-grain interactions in 3D.

To generalize GrainNN to 3D, we utilize graph representations of the grain structure that consist of both grains and the vertices of junctions between grains. Vertex representations have previously been used in models [19,35–37] to track the motion of the grain junction points in polycrystalline microstructures. In these models, the motion of vertices during grain growth is derived from the minimization of the isotropic or anisotropic grain boundary energy. Although for vertex models the update of a grain network is efficient and schemes for handling topological transitions are proposed [35] for curvature-driven grain coarsening, vertex models cannot currently describe grain evolution driven by a temperature gradient during AM solidification. A grain-centric graph structure has been used to predict grain microstructure evolution during solidification [38]. This physics-embedded graph network (PEGN) approach combines classic phase-field (PF) theory into a graph representation of the grains to accelerate PF simulations. Evolution is dictated by the minimization of a PF-derived free energy functional. PEGN is able to capture grain statistics in various AM setups but loses the accuracy of PF in terms of predicting actual grain shapes, due to the lack of tracking grain boundaries and topological changes.

Graph representations of grains have been used for graph neural networks (GNNs) [39,40] to create static microstructure-to-material-property maps. Graph convolutions on the grain networks are used to capture the spatial variations of microstructure properties and the graph sparsity enables faster evaluations of material behavior than high fidelity models. Beyond applications to material microstructures, GNNs have been successful in a wide range of other applications where the data is amenable to a graph structure [41–47].

Previous GNNs successful for processing dynamic graphs include GC-LSTM [48], EvolveGCN [49], and dyngraph [50], etc. Most use an RNN (LSTM) or autoencoder to encode graph features and a decoder to predict the probability of a node/link formation or destruction. In this paper, we add a transformer [51] operator to GC-LSTM encoder to capture grain interactions more accurately.

2. Methods

In this section we discuss the overall methodology. We start with the grain growth model, then we discuss the reduced parameterization of $\boldsymbol{\theta}$, the $\boldsymbol{\theta}$-to-graph map, the graph dynamics, the LSTM neural networks, the graph update algorithm, and the graph-to-$\boldsymbol{\theta}$ map. We conclude with details about the phase field solver.

Table 1
Notation table.

Symbol	Description (Units)
T	temperature field
ϕ_α	phase field
$\mathcal{I}$	grain index field
$\boldsymbol{\theta}$	grain orientation field (°)
L	domain size, $L = \{L_x, L_y, L_z\}$
G_z	temperature gradient (K/μm)
R_z	pulling velocity (m/s)
$\boldsymbol{p}$	process parameters, $p = \{G_z, R_z\}$
$\boldsymbol{\xi}$	initial state of the substrate
l	layer index
n_l	number of layers
I_l	grain index image at height z_l
G_l	graph extracted from I_l
F_l	features of graph G_l
g	grain index
j	junction index
V_g	grain vertices of graph G
V_j	junction vertices of graph G
n_g	number of grains of graph G
n_j	number of junctions of graph G
E_{jg}	junction-grain edges of graph G
E_{jj}	junction-junction edges of graph G
N_j	grain neighbors of junction j
N_g	junction neighbors of grain g

2.1. Problem formulation and high fidelity model

Here we describe the model we used to generate the training data. We use a phase field simulation. We would like to emphasize that GrainGNN does not depend on any particular model or numerical method for generating the grain microstructure. For example level sets, different grain formation models, or cellular automata could be used. The main assumption in our surrogate does not capture nucleation.

Regarding the particular grain formation model, we adopt a grain-scale phase field model described in Ref. [52] and ignore nucleation.[1] Each phase field component ϕ_α is associated with one of n_g different crystalline orientations, where n_g is the number of grains. The dynamics of ϕ_α is governed by the following equation.

$$\tau_\alpha \frac{\partial \phi_\alpha}{\partial t} = \nabla \cdot (W_\alpha^2 \nabla \phi_\alpha) + \sum_{j=x,y,z} \partial_j \left[|\nabla \phi_\alpha|^2 W_\alpha \frac{\partial W_\alpha}{\partial(\partial_j \phi_\alpha)} \right] + \phi_\alpha - \phi_\alpha^3 - \lambda(1-\phi_\alpha^2)^2 \frac{T - T_M}{L_f / C_p}$$
$$- \omega \frac{\phi_\alpha + 1}{2} \sum_{\beta \neq \alpha} \left(\frac{\phi_\beta + 1}{2} \right)^2, \quad (L_x, L_y, L_z') \times (0, t_H], \quad \alpha = 1, ..., n_g, \tag{2.1a}$$

$$\text{BCs:} \quad \phi_\alpha(x=0) = \phi_\alpha(x=L_x) \tag{2.1b}$$

$$\phi_\alpha(y=0) = \phi_\alpha(y=L_y) \tag{2.1c}$$

$$\partial_z \phi_\alpha(z=0, L_z') = 0 \tag{2.1d}$$

$$\text{IC:} \ \phi_\alpha(\boldsymbol{x}, t=0) = \begin{cases} \tanh\left(\frac{z_0 - z}{W_0}\right), & \text{if } (x,y) \in \Omega_\alpha(\boldsymbol{\xi}) \\ -1, & \text{if } (x,y) \notin \Omega_\alpha(\boldsymbol{\xi}) \end{cases} \tag{2.1e}$$

where

$$W_\alpha = W_0 \left(1 - 3\epsilon_c + 4\epsilon_c \frac{(\partial_x \phi_\alpha)^4 + (\partial_y \phi_\alpha)^4 + (\partial_z \phi_\alpha)^4}{|\nabla \phi_\alpha|^4} \right), \tag{2.2a}$$

$$\tau_\alpha = \tau_0 \left(1 + 3\epsilon_k - 4\epsilon_k \frac{(\partial_x \phi_\alpha)^4 + (\partial_y \phi_\alpha)^4 + (\partial_z \phi_\alpha)^4}{|\nabla \phi_\alpha|^4} \right), \tag{2.2b}$$

$$T = T_M + G_z(z - R_z t). \tag{2.2c}$$

[1] We remark that the model we use neglects the concentration field and only considers the evolution of phase fields with temperature fields in the rapid solidification regime. The grain structure can be modeled with even more accurate and even more expensive dendrite-resolving models [53].

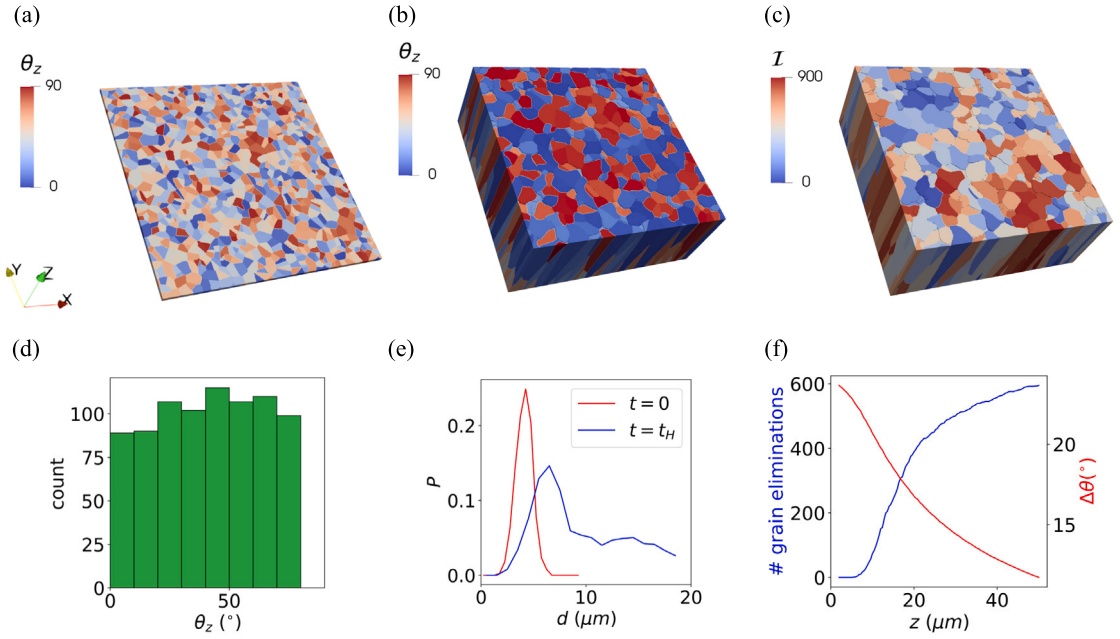

Fig. 2. A phase field simulation of epitaxial growth of 900 grains. Temperature gradient $G_z = 10$ K/μm and pulling velocity $R_z = 2$ m/s (a) Initial substrate with width 120 μm and height 2 μm. θ_z is the angle between the crystal orientation and the z-axis. (b) Phase field microstructure after 10K time steps. The time step size is 2.41 nanoseconds. The final height of the interface is 50 μm. (c) The corresponding grain index field of (b). Each grain is assigned a unique index from 1 to 900. (d) Initial grain orientation distribution. (e) Probability distributions of grain size d at $t = 0$ and $t = t_H$. (f) Time evolution of the quantities of interest. z represents the height of the current solid-liquid interface. As the height of the interface increases, the number of eliminated grains increases and the volume-weight misorientation $\Delta\theta$ decreases. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Here L_x, L_y, L_z' are the domain dimensions for simulation and t_H is the time horizon; τ_0 is the interface attachment time scale; W_0 is the width of the anisotropic interface; λ is the thermal coupling constant; L_f is the latent heat; C_p is the heat capacity; T_M is the melting temperature; ω is a scalar interaction parameter that sets the repulsive strength between adjacent grains of different orientations [3]. ϵ_c and ϵ_k are the capillary and kinetic anisotropy coefficients respectively, which are assumed to have a four-fold symmetry. In Eq. (2.2c), we assume the temperature gradient G_z and the pulling velocity R_z are aligned with the z-axis and they are constants during the simulation. Under this temperature profile, grains are growing in the z-direction. We name $\boldsymbol{p} = \{G_z, R_z\}$ the process parameters. $\boldsymbol{p}$ are the main free parameters during AM processes given the alloy with its material parameters. We specify the values for the material parameters in the appendix. The discretization of Eq. (2.1a) is discussed in Section 2.8.

We use no-flux boundary conditions at the top and bottom surface ($z = 0, z = L_z'$) and periodic boundary conditions at the four sides of the domain. We initialize ϕ_α as follows. Assume that the initial SLI is at $z = z_0$, we partition the $z = z_0$ plane into n_g regions. For phase field α, if point (x, y) is in region Ω_α we use a tanh function to create a smooth transition from solid (1) to liquid (-1) in the z-direction; if point (x, y) is outside of Ω_α, we set ϕ_α to -1. $\boldsymbol{\xi}$ parameterizes the initial realization of Ω_α. Here we use the Voronoi diagram [6] which creates the Voronoi tessellations from a set of seed points. We let $\boldsymbol{\xi} = \{\boldsymbol{x}_j^0\}_{j=1}^{n_j} \cup \{\boldsymbol{\theta}_g\}_{g=1}^{n_g}$, where $\boldsymbol{x}_j^0$ are the vertices of the Voronoi diagram and $\boldsymbol{\theta}_g$ are the grain orientation for each phase field. With a crystal orientation angle $\boldsymbol{\theta}_g$, the anisotropy functions Eq. (2.2a) and Eq. (2.2b) are modified by replacing all spatial derivatives by the derivatives with respect to the rotated coordinate system (x', y', z') with angle $\boldsymbol{\theta}_g$ [54]. For each phase field simulation, $\boldsymbol{\xi}$ is randomly generated. The orientation of each grain is sampled from the unit sphere. If $\boldsymbol{d}$ is the vector representing the grain's orientation, we first sample $\boldsymbol{d} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and set $\boldsymbol{d} = \boldsymbol{d}/|\boldsymbol{d}|_2$. Samplings of $\boldsymbol{x}_j^0$ are discussed in Section 3.1.

Fig. 2 shows an example of a simulation with domain size (120 μm, 120 μm, 50 μm). The interface width $W_0 = 0.1$ μm and the mesh size $dx = 0.08$ μm [6]. Therefore the grid size is $1500 \times 1500 \times 625$. The initial conditions for grain microstructure in the substrate are generated by sampling $\boldsymbol{x}_j^0$ from a uniform distribution and using a Voronoi diagram with periodic boundary conditions. In Fig. 2a, the substrate has 900 grains. In this simulation G_z and R_z are 10 K/μm and 2 m/s, respectively. The time step size is 2.42 nanoseconds. Fig. 2b is the resulting phase field microstructure after 10K time steps, which required approximately 48 minutes on one Nvidia A100 GPU.

Given the grain microstructure, we compute several *quantities of interest (QoIs)*: number (or percentage) of eliminated grains; grain size distribution; and volume-averaged misorientation. The number of eliminated grains is relevant in epitaxial growth [55]. We compute $n_G = n_G(z)$, the accumulated grain eliminations from the initial interface location to a height z (see Fig. 2f). Grain size distribution is a common descriptor of grain shape statistics and it affects mechanical properties such as strength and ductility. Grain size d of grain g is defined by its volume-equivalent diameter $d_g = (6\mathcal{V}_g/\pi)^{1/3}$, where $\mathcal{V}_g$ is the grain volume. Fig. 2d shows the probability distributions of the grain size for Fig. 2a and Fig. 2b. Volume-weighted misorientation $\Delta\theta$ quantifies the alignment of the polycrystalline orientation with the prescribed temperature gradient direction z. $\Delta\theta = \sum_g \mathcal{V}_g\theta_{z,g} / \sum_g \mathcal{V}_g$ and $\theta_{z,g}$ is the angle

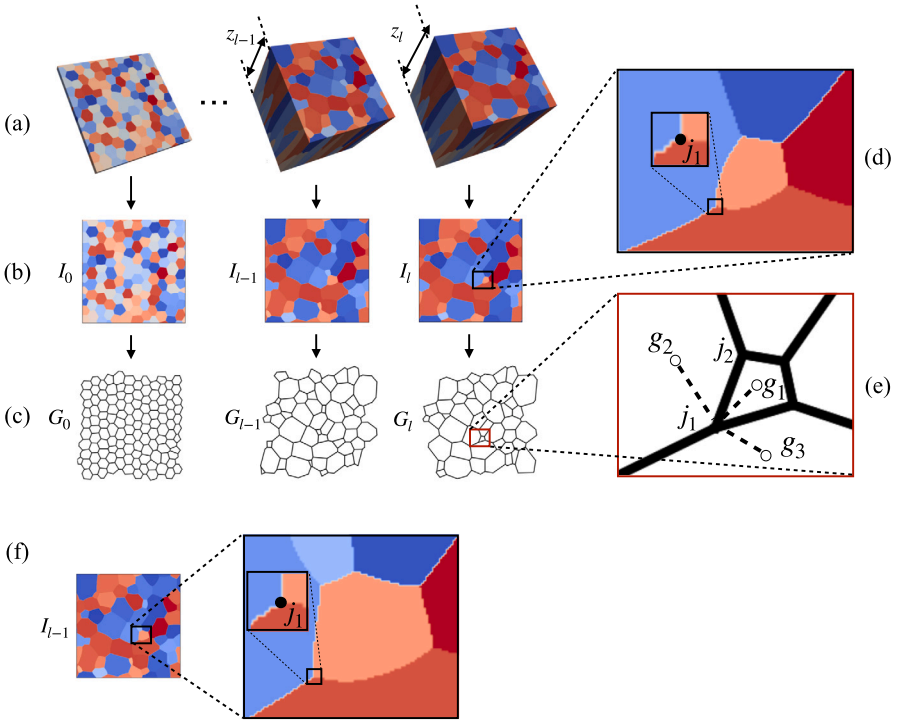

Fig. 3. Graph data extraction from phase field simulations. (a) Snapshots of a phase field simulation. Each snapshot is taken when the solid-liquid interface reaches height $z_l = l\Delta z, l = 0, 1, 2, ...$ (b) I_l is a 2D image of grain index field at height $z = z_l$. (c) Graph G_l extracted from the image I_l. (d) Zoomed-in inset of a region in the image I_l: j_1 is a junction pixel with three neighboring grains, g_1, g_2, g_3. (e) The same region in the graph G_l: G_l has two kinds of vertices and two kinds of edges; j_1 is a junction vertex corresponding to the junction pixel in (d); g_1 is a grain vertex; (j_1, j_2) is a junction-junction edge; (j_1, g_1) is a junction-grain edge. (f) A zoomed-in region in image I_{l-1}: j_1 is a junction with the same grain neighbors as j_1 in (d). We compare the grain neighbors to identify the same junctions in different graphs.

between the crystal orientation of grain g and the z-axis. As shown in Fig. 2f, $\Delta\theta$ decreases as the more aligned grains out-compete the misaligned grains. Due to random initial condition $\boldsymbol{\xi}$, we need to average across several simulations to obtain expectations of the QoIs.

2.2. GrainGNN

The input to GrainGNN is the initial state $\boldsymbol{\xi}$ of the substrate (see Fig. 2a), process parameters $\boldsymbol{p} = \{G_z, R_z\}$, and a domain size $L = (L_x, L_y, L_z)$. The output of GrainGNN is the grain orientation field $\theta(x, y, z)$ in a domain with size L. Notice that here L_z is the height of the final polycrystal (as shown in Fig. 2b); for phase field simulations we let $L'_z > L_z$ to avoid the effect of the top boundary. Table 1 summarizes the major symbols we use in this paper. In Section 2.3-Section 2.6 we explain the basic ingredients of GrainGNN.

- **Image-to-graph step**. We start with a spatial field compression method to reduce the large dimensionality of the phase field data. We define a planar graph G and hand-crafted features F to represent a layer of grain microstructure (see Fig. 3). To define these features from $\{\phi_\alpha\}_{\alpha=1}^{n_g}$ we first combine the fields to a single image $\theta(x, y, z)$. Then we propose an image-to-graph algorithm $\phi_\alpha(x, y, z) \to (G_l, F_l)$, $l = 0, 1, 2, ...$ to extract vertices and edges of grains at cross-sections defined at z_l. The features contain geometric information about the size of the grains and their boundary. The definitions of the image, graph, and features are discussed in Section 2.3.
- **Graph evolution step**. We now train a network for the map $(G_{l-1}, F_{l-1}) \to (G_l, F_l)$. The challenge is that graphs G_{l-1} and G_l can be topologically different due to grain and/or edge elimination events. To address this we need to predict both vertex feature evolution ΔF as well as a set of *grain neighbor switching events* $\mathcal{S}_E$ and a set of *grain elimination events* $\mathcal{S}_G$. We discuss the graph evolution in detail in Section 2.4. The prediction (G_l, F_l) is done by first using GrainGNN to predict ΔF and *probabilities* of events and then using an algorithm to reconstruct G_l and its features.
  - **Graph evolution, LSTM substep.** We design an LSTM architecture to learn the predictions $(\Delta F, \mathcal{S}_E, \mathcal{S}_G) = \text{GrainGNN}(G_{l-1}, F_{l-1})$. GrainGNN consists of two networks $\mathcal{R}$ and $\mathcal{C}$. $\mathcal{R}$ is a regressor that outputs ΔF and $\mathcal{S}_G$; $\mathcal{C}$ is a classifier that predicts $\mathcal{S}_E$. We present the summary of the network architecture Fig. 5 and the details in Section 2.5.
  - **Graph evolution, graph reconstruction substep.** Given the LSTM predictions we introduce a graph update algorithm that completes the graph prediction $(G_{l-1}, \mathcal{S}_E, \mathcal{S}_G) \to G_l$, where we create orderings for $\mathcal{S}_E$ and $\mathcal{S}_G$ to reconstruct the

graph G_l (see Fig. 6). The features are updated $(F_{l-1}, \Delta F, S_E, S_G) \rightarrow F_l$. We give the details of the graph reconstruction algorithm in Section 2.6.

- **Graph-to-image, microstructure reconstruction.** Using GrainGNN we can compute a graph "trajectory" $\mathcal{G} = \{G_0, G_1, G_2..., \}$, $\mathcal{F} = \{F_0, F_1, F_2..., \}$, given an initial condition: $(G_0, F_0) \rightarrow (\mathcal{G}, \mathcal{F})$. Once the graph trajectory is computed, we introduce a graph-to-image algorithm $(\mathcal{G}, \mathcal{F}) \rightarrow \theta(x, y, z)$ to reconstruct the grain orientation field; the algorithm is described in Section 2.7.

2.3. Graph representation of microstructure

Now, let us describe the compressed representation of the microstructure using hand-crafted features. Consider a solution of Eq. (2.1a). From this simulation we sample n_l layers of field data. We define a "layer" at height z_l as the microstructure at the time t_l when the solid-liquid interface (SLI) reaches z_l (see Fig. 3a). More precisely, t_l is defined as the time at which the lowest point (order in z coordinate) at the SLI reaches z_l.

We define $I_l(x, y)$ to be an *image*, a 2D scalar representation of the microstructure on the $x - y$ plane at z_l. Each layer I_l is a 2D image on the x-y plane (Fig. 3b):

$$I_l(x, y) = \arg\max_{\alpha} \phi_\alpha(x, y, z_l, t_l). \tag{2.3}$$

We solve Eq. (2.1a) to generate training data and verify the predictions of the surrogate. Although a simulation can have thousands of time steps, we subsample to n_l planes using equispaced sampling for z_l with height increment Δz, so $z_l = l\Delta z$. Typically we use $n_l = 20$. We discuss the choice of Δz and n_l in Section 2.4.

Definition of edges and vertices Once I_l have been identified, we extract graph $(G_l, F_l$ (Fig. 3c) as follows. $G(V, E)$ is an undirected graph where V and E denote the sets of vertices and edges respectively. In GrainGNN, G has two types of vertices $V = V_g \cup V_j$. $V_g = \{g_1, g_2, ..., g_{n_g}\}$ represents grains. $V_j = \{j_1, j_2, ..., j_{n_j}\}$ are junction vertices, where n_j is the number of junctions. A junction vertex refers to a point in image I_l that has three grain neighbors (e.g., j_1 in Fig. 3d). Notice that the junction vertices are just the vertices of the Voronoi diagram. G has two types of undirected edges $E = E_{jj} \cup E_{jg}$. E_{jj} are the edges between junction vertices and can be thought of as a representation of actual inter-grain boundaries on the x-y plane. E_{jg} are the junction-grain edges (dashed lines in Fig. 3e).

As we will see later we need to define a maximum degree (number of edges) for a junction vertex. The maximum number of edges for a junction vertex depends on the physical setting of the problem. In rapid solidification of metals, the grain boundary anisotropy is relatively weak compared to kinetic anisotropy [56], and a triple junction is more stable than a quadruple junction [19]. Thus we only consider triple junctions in this work, which means a junction vertex can only connect to the other three junction vertices and three grain vertices. In this case, $n_j = 2n_g$, $|E_{jj}| = 3n_g$, and $|E_{jg}| = 6n_g$. Our handling of the graph topological changes discussed in Section 2.4 is based on this simplification of the graph structure.

Let's now discuss how to identify vertices and edges from I_l. First, we identify the junction locations (x_j, y_j). For each pixel (x, y) of I_l, we inspect its eight neighbors each associated with a grain index. If there are three distinctive grain indices in eight neighbors, we consider pixel (x, y) to be a junction pixel, for example, j_1 in Fig. 3d. Every junction j is associated with a unique triplet of grain indices $N_j = \{g_1, g_2, g_3\}$. Commonly multiple adjacent pixels will have the same triplet and we only keep one of them.[2] Then using the triplet indices N_j we add three edges to the E_{jg} set: e.g., for j_1, the E_{jg} edges are (j_1, g_1), (j_1, g_2), (j_1, g_3). We create E_{jj} edges by $E_{jj} = \{(j_1, j_2) : N_{j_1} \cap N_{j_2} = 2, \forall j_1, j_2 \in G\}$. For example in Fig. 3e, edge (j_1, j_2) exists because j_1 and j_2 share the same neighboring grains g_1 and g_2.

Definition of features We clarify that these are input features to the LSTM and serve as a reduced representation of the grain structure. They are not the network hidden features, which are of course determined during training. We define vertex and edge features as follows:

- **Junction vertices:** The feature vector of a junction vertex is defined as:

$$\boldsymbol{f}_j = \frac{1}{\boldsymbol{f}_j^0}\begin{bmatrix} x_j & y_j & z_l & G_z & R_z & \Delta x_j & \Delta y_j & \Delta z \end{bmatrix}, \tag{2.4}$$

where x_j, y_j, and z_l are the 3D coordinates of a junction vertex; Δx_j and Δy_j are in-plane displacements of junctions with respect to their locations in the previous layer G_{l-1}, i.e., $\Delta x_j = x_{j,l} - x_{j,l-1}, \Delta y_j = y_{j,l} - y_{j,l-1}$. $\Delta z = z_l - z_{l-1}$ is the distance between two sampled images. For $l = 0$, we set $\Delta x_j = \Delta y_j = \Delta z = 0$. G_z and R_z here are constants; they're repeated at each vertex and we allow them to vary for each junction location.

[2] If two or more pixels have the same triplet $\{g_1, g_2, g_3\}$. We count the occurrences of each grain index in the eight neighbors of each pixel and use the one pixel whose index occurrences are more even. For example, for pixel 1 the occurrences are $\{3, 3, 2\}$ and for pixel 2 the occurrences are $\{4, 3, 1\}$ and we choose pixel 1.

- **Grain vertices:** For a grain vertex, we define $\boldsymbol{f}_g$ as:

$$\boldsymbol{f}_g = \frac{1}{\boldsymbol{f}_g^0}\left[x_g \;\; y_g \;\; z_l \;\; s_g \;\; v_g \;\; \cos\theta_x \;\; \sin\theta_x \;\; \cos\theta_z \;\; \sin\theta_z \;\; \Delta s_g \;\; \Delta z\right], \tag{2.5}$$

  where x_g, y_g, and z_l are the coordinates of the grain vertices; s_g is the grain cross-sectional area; and v_g is the grain excess volume above the interface height z_l [17]. For x_g, we average the coordinates of its junction neighbors $x_g = \sum_{k\in N_g} x_{j,k}/|N_g|$. N_g is the set of junctions connected to grain g. θ_z is the angle between the z-axis and the preferred growth direction $<100>$; θ_x is the angle between the x-axis and the projection of $<100>$ direction on the x-y plane. $\Delta s_g = s_{g,l} - s_{g,l-1}$ is the change of cross-sectional area; $\Delta s_g = 0$ at $l = 0$. In Eq. (2.5) and Eq. (2.4), $\boldsymbol{f}_j^0$ and $\boldsymbol{f}_g^0$ are constant vectors to normalize features to [0, 1]. $\boldsymbol{f}_j^0 = \left[L_x, L_y, L_z, G_{\max}, R_{\max}, L_x, L_y, L_z\right]$ and $\boldsymbol{f}_g^0 = \left[L_x, L_y, L_z, L_xL_y, L_xL_yL_z, 1, 1, 1, 1, L_xL_y, L_z\right]$. $G_{\max}$ and $R_{\max}$ are maximum values of G_z and R_z in the training data.
- **Edge features:** For each E_{jj} and E_{jg} edge, we include its length as the edge feature. Notice the length of an edge is calculated with periodic boundary conditions. We neglect the curvature of E_{jj} edge for simplicity (but notice that the grain boundary curvature in the z-direction is still captured).

In summary, F contains three feature matrices $F_j = \left[\boldsymbol{f}_{j,1} \;\; \boldsymbol{f}_{j,2} \;\; \cdots \;\; \boldsymbol{f}_{j,n_j}\right]$, $F_g = \left[\boldsymbol{f}_{g,1} \;\; \boldsymbol{f}_{g,2} \;\; \cdots \;\; \boldsymbol{f}_{g,n_g}\right]$, and edge features $F_E = \{|\boldsymbol{x}_i - \boldsymbol{x}_j|, \forall (i,j) \in E\}$. The values of Δx_j, Δy_j, Δs_g are obtained by subtracting F_l and F_{l-1}, which is discussed in the next section.

2.4. Graph evolution

We now present how to model the graph-to-graph mapping $(G_{l-1}, F_{l-1}) \to (G_l, F_l)$. The dynamics observed in epitaxial grain growth can be decomposed into three basic types [18,19]: (i) junction vertex movement, (ii) grain neighbor switching, and (iii) grain elimination. We remark that (i) does not change the graph, only the value of its features; (ii) changes features and it removes and adds edges. (iii) does both (i) and (ii) but also removes vertices, both junction and grain vertices. We define two basic operations that compose the topological changes from G_{l-1} to G_l. One is a neighbor-switching operation O_E; another is a vertex-removal operation O_G. By operation we mean a sequence of deterministic actions that add/remove vertices or edges. One neighbor switching event causes one O_E; one grain elimination event causes one O_G and multiple O_E. Next we discuss these operations in detail.

Grain neighbor switching events: Grain neighbor switching happens when a grain edge becomes smaller and eventually disappears. A disappearing edge creates a four-edge junction (Fig. 4a). Since in our PDE model four-edge junctions are unstable, a new edge starts growing and the four-junction becomes two triple junctions. This new edge causes the neighbor to switch. Physically, a neighbor-switching event involves four grains, where two grains lose one face and the other two grains gain one face. From I to II, grain g_1 and g_2 push the vertices j_1 and j_2 to move toward each other. At some intermediate time, j_1 and j_2 merge to form a quadruple junction. Then from II to III, the quadruple junction separates into two new junctions j_1' and j_2', which replace j_1 and j_2. This event changes several edges, grain g_1 and g_2 grain an additional junction-grain edge; g_3 and g_4 lose one junction-grain edge; (j_2', j_5) and (j_1', j_4) replace (j_1, j_5) and (j_2, j_4); the total number of edges remains the same. The grain neighbor-switching event is characterized by the removal of (j_1, j_2). We will refer to a grain neighbor switching event as an "edge" event in the following discussion.

Grain elimination events: Fig. 4b showcases a grain elimination event on g_3 which has three grain neighbors. In this example, face (j_1, j_2) shrinks faster than two other faces and triggers a neighbor switch with j_1', j_2' the new junctions. II to III is an O_G operation. In this operation, we remove vertices g_3, j_1', j_3, and edges connected to j_1' and j_3. Finally, we add a new edge (j_2', j_4). Note that one grain about to be eliminated can have a different number of faces, typically three to seven. Elimination of an $|N_g|$-side grain needs $|N_g| - 2$ O_E operations and one O_G operation. One O_G operation removes one V_g vertex, two V_j vertices, three E_{jj} edges, and six E_{jg} edges.

Matching G_{l-1} and G_l and handling elimination events: Our graph update requires computing $\Delta F, \mathcal{S}_E, \mathcal{S}_G$. We define

$$\Delta F = \{\Delta x_j, \Delta y_j, \Delta s_g, v_{g,l}, \; \forall j, g \in G_{l-1}\}. \tag{2.6}$$

To create ΔF for training data, we first need to match the vertices between G_{l-1} and G_l.

If there were no topological changes, the two graphs can be easily matched as follows. V_g are identical for the two graphs. Junction vertices are matched by their grain index triplets N_j. For example in Fig. 3d and Fig. 3f, the magnified junctions have the same grain neighbors so they are the same junction vertex. $\Delta x_j, \Delta y_j, \Delta s_g$ are then obtained by subtracting features of their matched vertices.

Grain elimination events $\mathcal{S}_G$ can also be handled relatively easily. They cause grain vertices in G_{l-1} to be missing from G_l. These vertices are defined by the set $\mathcal{S}_G = \{g : g \in G_{l-1} \wedge g \notin G_l\}$.

Neighbor-switching events are harder to handle: they result in junctions whose triplets N_j are different in G_l and G_{l-1}. Let $\mathcal{U}_{l-1}$ be the junction vertices in G_{l-1} but not matched in G_l, $\mathcal{U}_{l-1} = \{j : j \in G_{l-1} \wedge j \notin G_l\}$. We enumerate every two junctions in $\mathcal{U}_{l-1}$ and check (i) if they have an E_{jj} edge and (ii) if they become two new vertices of G_l through a neighbor switching event. For the second check, we use their N_j triplets. We use Fig. 4a as an example. $N_{j_1} = \{g_1, g_3, g_4\}$ and $N_{j_2} = \{g_2, g_3, g_4\}$. If an edge event happened on (j_1, j_2), two new triplets $N_{j_1'} = \{g_1, g_2, g_3\}$ and $N_{j_2'} = \{g_1, g_2, g_4\}$ must have formed at an intermediate time. We check that if $N_{j_1'}$ and $N_{j_2'}$ exist in G_l, criterion (ii) is satisfied. Thus neighbor-switching events $\mathcal{S}_E$ are junction pairs in $\mathcal{U}_{l-1}$ that satisfy (i)

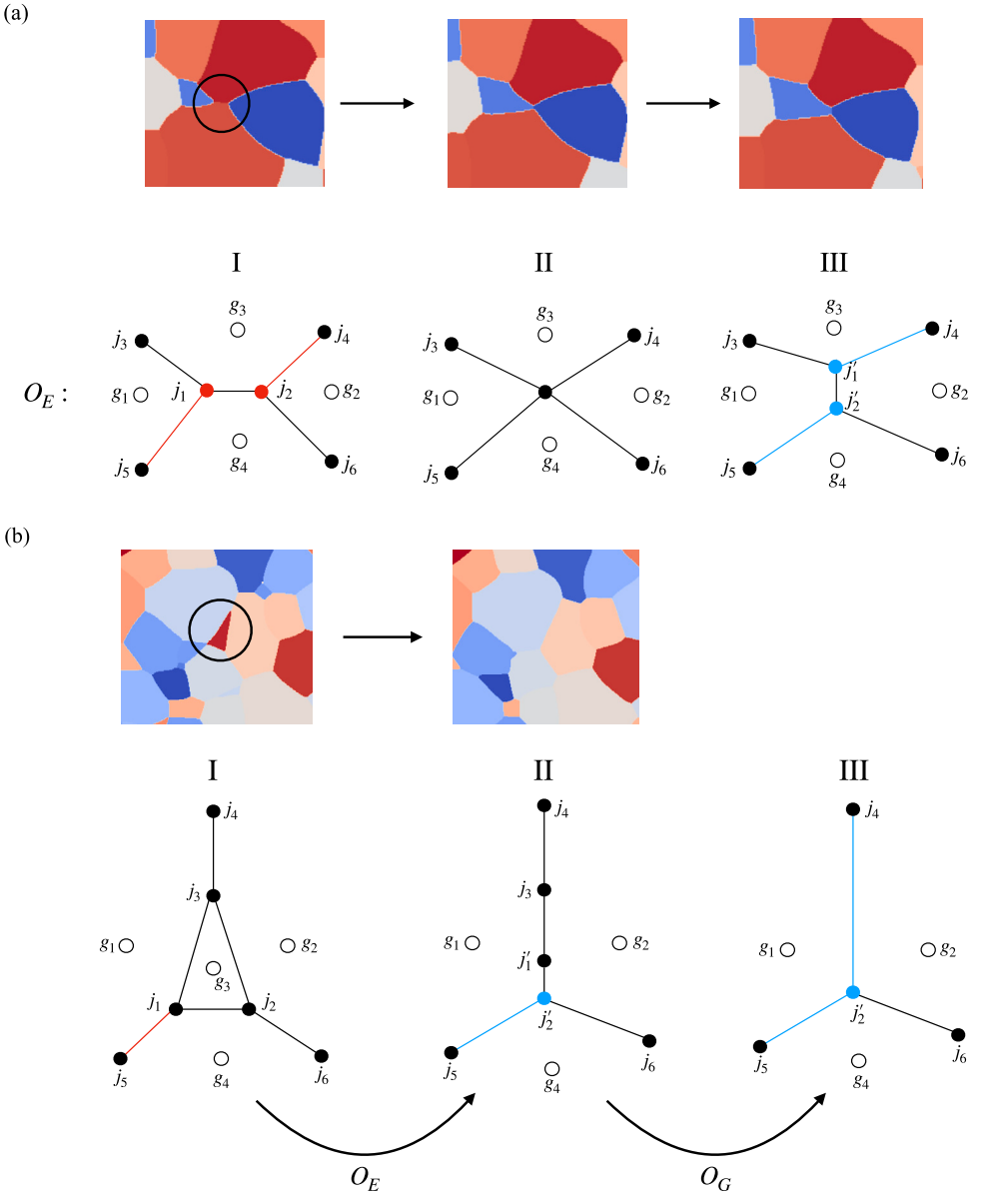

Fig. 4. Topological events with the update of graph connectivity. (a) A neighbor-switching event requires changes of the graph topology: an O_E operation. Grains g_3 and g_4 initially are neighbors. After applying O_E, g_1 and g_2 are neighbors. From I to II, j_1, j_2 merge to a quadruple junction; from II to III, the quadruple junction splits into two triple junctions. This event replaces edges (j_1, j_5), (j_2, j_4) (in red color) with the edges (j_2', j_5), (j_1', j_4) (in blue color). The total number of edges remains the same. (b) A grain elimination event. From I to II, a neighbor-switching event occurs on (j_1, j_2); g_3 becomes a 2-side grain. II to III is an O_G operation. It removes g_3, j_1', j_3, a couple of edges, and replace them with (j_2', j_4).

and (ii). We remark this method cannot match all the junction vertices. When a junction has been involved in two or more edge or grain elimination events (e.g., Fig. 6a), we fail to detect the events. In this case, we mask out (in training) the unmatched junctions and E_{jj} edges. These are typically less than 1% of the total vertices and edges.

Choosing Δz: Finally, we discuss how to choose Δz, the distance between I_{l-1} and I_l. If Δz is too small, GrainGNN will need too many steps during inference, and make the surrogate too expensive. If Δz is too large, I_{l-1} and I_l will be exceedingly different as they will involve a large number of topological events. With these observations in mind, we choose the number of layers n_l such that:

$$\frac{n_G(z = L_z)}{n_g^0 n_l} \approx 3\%, \qquad \Delta z = L_z/(n_l - 1), \tag{2.7}$$

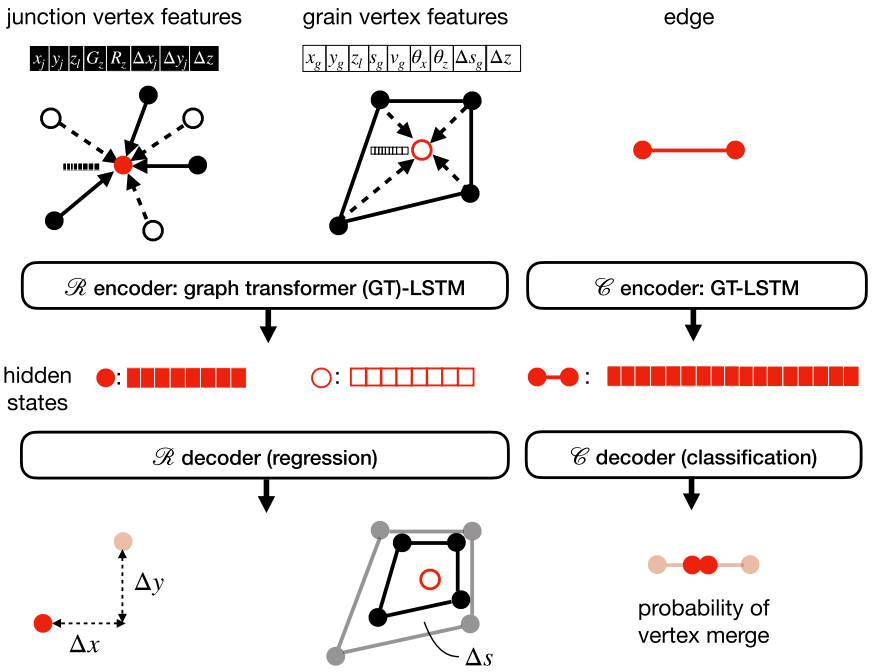

Fig. 5. The LSTM architecture. We use a graph transformer LSTM with a regressor $\mathcal{R}$ and a classifier C—both having an encoder-decoder structure. The classifier and regressor are trained separately. The junction nodes (solid circles) and grain nodes (hollow circles) have vertex features defined in Eq. (2.4) and Eq. (2.5). For each target vertex (red circles), graph transformer operators in Eq. (2.8) aggregate the features of the vertex itself with the neighboring vertices into a hidden vector. The decoder $\mathcal{R}$ then uses Eq. (2.11) to transform the hidden vector to the target outputs, which are displacements of junctions, area change, and excess volume of grains. For each junction-junction edge, the model C concatenates the hidden vectors of the two connecting vertices and predicts the probability of the edge event (see Eq. (2.13)).

where n_g^0 is the number of grain vertices of G_0. We allow about 3% of grains to be eliminated per graph update. Notice that $n_G < n_g^0$ so $n_l \sim O(1)$. For training data, we count n_G and calculate n_l and Δz with Eq. (2.7). For example, if we run a 100-grain phase field simulation to have 40 grains at the end, $n_l = (100-40)/100/3\% = 20$. For GrainGNN inference, we don't know n_G beforehand. We first create map $\Delta z(\boldsymbol{p})$ using the training $\boldsymbol{p}$ grid. Then for a testing $\boldsymbol{p}$, we use the nearest neighbor interpolation for Δz.

2.5. The LSTM architecture

In this section, we discuss the neural network at the heart of the GrainGNN surrogate. It comprises a regressor $\mathcal{R}$ and a classifier C. Both have an encoder-decoder structure. Graph features of F_{l-1} are encoded to intermediate hidden states H_l through the encoder and then H_l are decoded to the target outputs (see Fig. 5). We use a graph transformer LSTM as the encoder for both $\mathcal{R}$ and C. The encoders for $\mathcal{R}$ and C have the same architecture but they have different weights and they're trained separately.

LSTM encoder architecture. Each encoder has several identical LSTM [29] layers stacked on top of each layer. The LSTM layer hidden states are denoted by $H \in \mathbb{R}^{D_h \times (n_j + n_g)}$; its cell states by $C \in \mathbb{R}^{D_h \times (n_j + n_g)}$. Here D_h, the hidden dimension, is a network architecture hyperparameter. Hidden and cell states are intermediate outputs of LSTM to store short- and long-term prior information. The inputs to each LSTM layer are the feature matrices F_{l-1} and the graph adjacency matrix A_{l-1} ($A_{ij} = \mathbb{1}_{(i,j)\in E}$). The outputs are the updated hidden and cell states H_l, C_l. Let $U_{l-1} = [F_{l-1}, H_{l-1}]$ be the input matrix, and one LSTM layer is defined as follows:

$$\begin{aligned} i_l &= \sigma\left(\mathcal{T}_i\left(U_{l-1}, A_{l-1}\right) + b_i\right), \\ f_l &= \sigma\left(\mathcal{T}_f\left(U_{l-1}, A_{l-1}\right) + b_f\right), \\ C_l &= f_l C_{l-1} + i_l \tanh\left(\mathcal{T}_c\left(U_{l-1}, A_{l-1}\right) + b_c\right), \\ o_l &= \sigma\left(\mathcal{T}_o\left(U_{l-1}, A_{l-1}\right) + b_o\right), \\ H_l &= o_l \tanh\left(C_l\right), \end{aligned} \tag{2.8}$$

where $\mathcal{T}_i$, $\mathcal{T}_f$, $\mathcal{T}_c$, $\mathcal{T}_o$ are graph transformer operators [57] with trainable weights; σ is the sigmoid function; b_i, b_f, b_c, b_o are biases. For the first layer, H_{l-1} and C_{l-1} are initialized as zeros; for other layers, H_{l-1} and C_{l-1} are initialized with H_l and C_l of the previous layer.

The graph transformer operator $\mathcal{T}$, is a message-passing neural network [45]. $\mathcal{T}$ operates on every node in the graph with the node's neighboring nodes. In Fig. 5 (top row), the red circles indicate a node on which we evaluate $\mathcal{T}$ and the black circles its neighbors. Every neighbor passes its vertex and edge features to the target node. $\mathcal{T}$ aggregates the passed information and the

features of the target node into a single vector. Let $\boldsymbol{u}_i = [\boldsymbol{f}_i \ \boldsymbol{h}_i]$ be the input vector of node i; $\boldsymbol{h}_i \in \mathbb{R}^{D_h}$ is node i's hidden vector. $\mathcal{T}(U,A)_i \in \mathbb{R}^{D_h}$, the output vector for node i, is given by [57]:

$$\mathcal{T}(U,A)_i = W_1\boldsymbol{u}_i + \sum_{k\in N_i} \beta_{i,k}\left(W_2\boldsymbol{u}_{k,i} + W_3\boldsymbol{f}_{ik}\right), \tag{2.9a}$$

$$\beta_{i,k} = \mathrm{softmax}\left(\frac{\left(W_4\boldsymbol{u}_i\right)^\top\left(W_5\boldsymbol{u}_{k,i} + W_3\boldsymbol{f}_{ik}\right)}{\sqrt{D_h}}\right), \tag{2.9b}$$

where W_1, W_2, W_3, W_4, W_5 are trainable weights for one $\mathcal{T}$ operator; N_i is the set of neighbors of vertex i, $N_i = \{k \in V : A(i,k) \neq 0\}$; $\boldsymbol{f}_{ik}$ is the edge feature of the edge (i,k); $\beta_{i,k} \in (0,1)$ is the attention coefficient [51] that measures the coupling strength between node i and k. $\boldsymbol{u}_{k,i}$ is the *passed* input vector of node k; it is modified from $\boldsymbol{u}_k$. Recall for all vertices, the first three elements of $\boldsymbol{u}_k$ are its absolute coordinates $\boldsymbol{x}_k = (x_k, y_k, z_k)$. We replace $\boldsymbol{x}_k$ with $\boldsymbol{x}_{k,i}$, the relative coordinates with respect to node i, and account for the periodic boundary conditions:

$$\boldsymbol{x}_{k,i} = \boldsymbol{x}_k - \boldsymbol{x}_i - \mathrm{nint}(\boldsymbol{x}_k - \boldsymbol{x}_i), \tag{2.10}$$

where nint is the nearest integer function. For example, if $x_k = 0.8$ and $x_i = 0.1$, $\mathrm{nint}(x_k - x_i)$ gives 1 and $x_{k,i} = -0.3$. Thus, the couplings of nodes are only sensitive to their distance and thus are translation invariant. If we translate the coordinate system, the coupling term won't change. In Section 3.3, we will see this treatment allows GrainGNN to scale to large systems.

Regression decoder. The hidden states H_l of the *last* LSTM layer are passed to decoders. The decoder of model $\mathcal{R}$ is just a single layer perceptron: a linear layer with an activation function:

$$\Delta x_j = \tanh(W_{hx}\boldsymbol{h}_j + b_x), \tag{2.11a}$$

$$\Delta y_j = \tanh(W_{hy}\boldsymbol{h}_j + b_y), \text{ for } j = 1,...,n_j, \tag{2.11b}$$

$$\Delta s_g = \tanh\left(W_{hs}\boldsymbol{h}_g + b_s\right), \tag{2.11c}$$

$$v_{g,l} = \mathrm{ReLU}\left(W_{hv}\boldsymbol{h}_g + b_v\right), \text{ for } g = 1,...,n_g. \tag{2.11d}$$

$W_{hx}, W_{hy}, W_{hs}, W_{hv}$ and b_x, b_y, b_s, b_v are trainable weights and biases. Here $\Delta x_j, \Delta y_j, \Delta s_g, v_{g,l}$ are normalized outputs. We use tanh activation for Δx_j, Δy_j, and Δs_g because their values are in [-1, 1]. We use Rectified Linear Unit (ReLU) as the activation function for v_g to ensure its non-negativity. From Eq. (2.11c) we compute the updated cross-sectional area $s_{g,l} = s_{g,l-1} + \Delta s_g$. Grain g is removed from graph when $s_{g,l} < \epsilon_G$, where ϵ_G is a hyperparameter, that is determined by parameter sweep during network training. Typically $\epsilon_G = 10^{-4}$ in our experiments.

Regression loss function. Let ΔF be the training data and $\Delta\tilde{F}$ be the corresponding network prediction, the L_2-loss function for $\mathcal{R}$ is:

$$L_2 = \frac{1}{n_j}\sum_{j=1}^{n_j}\left[\left(\Delta x_j - \Delta\tilde{x}_j\right)^2 + \left(\Delta y_j - \Delta\tilde{y}_j\right)^2\right] + \frac{1}{n_g}\sum_{g=1}^{n_g}\left[\left(\Delta s_g - \Delta\tilde{s}_g\right)^2 + \left(v_g - \tilde{v}_g\right)^2\right]. \tag{2.12}$$

Classification decoder. The decoder of the classifier C predicts the probability of the edge event P for each E_{jj} edge. As shown in Fig. 5, C forms a vector that concatenates $\boldsymbol{h}_i$, $\boldsymbol{h}_j$, and $\boldsymbol{f}_{ij}$. Then C uses a linear layer followed by a sigmoid function to output the probability $P_{ij} \in (0,1)$:

$$P_{ij} = \sigma\left(W_{hc}[\boldsymbol{h}_i, \boldsymbol{h}_j, \boldsymbol{f}_{ij}] + b_c\right), \ \forall (i,j) \in E_{jj}, \tag{2.13}$$

where W_{hc} and b_c are trainable weights and biases. An E_{jj} edge with probability P_{ij} higher than $\epsilon_E \in (0,1)$ is classified as a positive event of neighbor switching, where ϵ_E is the edge classification threshold.

Classification loss function. We use a binary cross-entropy (BCE) loss for C. Let $Y_{ij} = \{0,1\}$ be the truth labels of whether (i,j) is eliminated, the loss function is:

$$L_{CE} = \frac{1}{|E_{jj}|}\sum_{\forall (i,j)\in E_{jj}} -Y_{ij}\log P_{ij} - (1 - Y_{ij})\log\left(1 - P_{ij}\right). \tag{2.14}$$

Network evaluation metrics. We use several metrics to evaluate the accuracy of the two networks. For the regression network, we compute the Relative Root Mean Square Error (RRMSE) of the outputs:

$$\mathrm{RRMSE} = \sqrt{\frac{\sum_{i=1}^{n}\left(x_i - \tilde{x}_i\right)^2}{\sum_{i=1}^{n} x_i^2}} \times 100, \tag{2.15}$$

where x_i and $\tilde{x}_i$ are the ground truth and prediction respectively; n is the number of samples. For classification accuracy, we use F_1-score. F_1-score is the harmonic mean of the prediction precision and recall, where precision = TruePositive / (TruePositive + FalsePositive) and recall = TruePositive / (TruePositive + FalseNegative). Precision and recall both depend on the classification thresholds. We calculate the area under the curve (AUC) of the precision-recall curve drawn from different thresholds; a higher

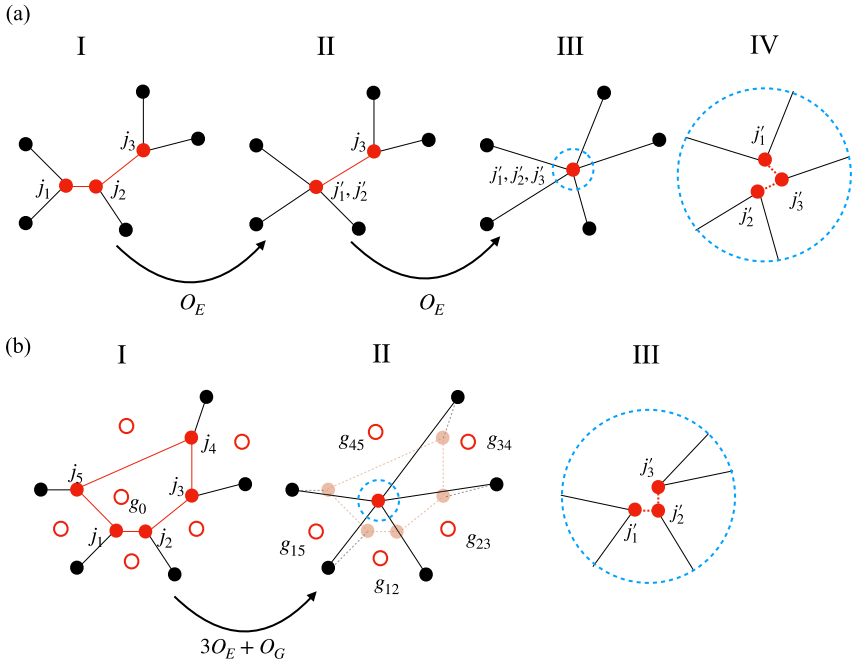

Fig. 6. GrainGNN graph update orderings given events lists $\mathcal{S}_E$ and $\mathcal{S}_G$. (a) Order of edge events. (j_1, j_2) and (j_2, j_3) are edges with elimination probability $P_{12} > P_{23} > \epsilon_E$. GrainGNN applies O_E on (j_1, j_2) before (j_2, j_3). For each O_E, the edge connectivity is updated as shown in Fig. 4a. We assume new junction points j_1', j_2' are both located at the midpoint of edge (j_1, j_2). After event (j_2, j_3), j_1, j_2, j_3 are at the same location but the connectivity is as shown in IV. (b) Elimination of grain g_0. The list of O_E operations is sorted by the area increment of neighboring grains. Here $\Delta s_{g_{15}} < \Delta s_{g_{12}} < \Delta s_{g_{23}} < \Delta s_{g_{45}} < \Delta s_{g_{34}}$. Thus GrainGNN applies O_E in the order $(j_1, j_5), (j_1, j_2), (j_2, j_3)$.

AUC represents a more accurate classifier. From the precision-recall curves, we find the optimal classification thresholds ϵ_E and ϵ_G leading to the highest F_1-scores. The details of the training setup are provided in Section 3.1.

2.6. Graph reconstruction and GrainGNN algorithm

Given the trained LSTM networks and classification thresholds, we apply the graph-to-graph update algorithm Algorithm 1 that implements $(G_{l-1}, F_{l-1}) \to (G_l, F_l)$. This in essence the GrainGNN surrogate along with the image-to-graph and graph-to-image pre- and post-processing steps.

In GrainGNN's lines 2-3, the LSTMs $\mathcal{R}$ and C compute ΔF and P (Eq. (2.11), Eq. (2.13)) respectively using the input features F_{l-1}. In lines 4-7, we update the features x_j, y_j, z_l, s_g, v_g with ΔF and create the lists of events $\mathcal{S}_E$ and $\mathcal{S}_G$. Next, we update the graph with $\mathcal{S}_E$ and $\mathcal{S}_G$.

Note that during one graph update, some junctions are involved in more than one event for example j_2 in Fig. 6a. Different orders of applying $\mathcal{O}_E(j_1, j_2)$ and $\mathcal{O}_E(j_2, j_3)$ will result in different graphs. We address this by imposing an operator event ordering based on the predicted features. We next discuss this for grain elimination events and grain neighbor switching events.

Ordering grain elimination events. We sort $\mathcal{S}_G$ by the predicted grain area s_g. Grains with smaller areas are expected to be eliminated first. Lines 11-20, define required graph topology updates when grain g is eliminated. Recall a grain elimination event requires $|N_g| - 2$ O_E operations followed by an O_G operation. We denote the O_E operations of grain g by $\mathcal{S}_{E,g}$. An O_E operation includes the updates of the edges as shown in Fig. 4a. If (j_1, j_2) is given as an edge event, we find their neighboring nodes $g_1, g_2, g_3, g_4, j_3, j_4, j_5, j_6$ by searching the edge list E for edges having node j_1 or j_2. Then we perform edge replacements $(j_1, g_4) \to (j_1', g_2), (j_2, g_3) \to (j_2', g_1), (j_1, j_5) \to (j_2', j_5), (j_2, j_4) \to (j_1', j_4)$. O_E also predicts the coordinates of j_1', j_2'. Currently, we do not implement this part in the networks; we approximate the coordinates of j_1' and j_2' simply with the midpoint of the edge (j_1, j_2), i.e., $x_{j_1'} = x_{j_2'} = 0.5(x_{j_1} + x_{j_2})$ (Fig. 6a). We use this approximation due to the lack of training edge events, which is only about 2% of the number of E_{jj} edges.

Ordering grain switching neighbor events. We need to perform this for both $\mathcal{S}_E$ and for switch events $\mathcal{S}_{E,g}$ triggered by grain eliminations. We discuss the latter first. In line 12 we initialize $\mathcal{S}_{E,g}$ as the edges of grain g. Then, we sort $\mathcal{S}_{E,g}$ by the area change of g's neighboring grains g_{ij}, where g_{ij} is the grain which shares the edge (i, j) with g. In one O_E operation, both g and g_{ij} lose edge (i, j). Generally, a grain with an expanding cross-sectional area Δs is more likely to gain faces rather than lose faces. Thus we choose to first update the edge of the grain whose Δs is the smallest. For example in Fig. 6b, the regressor $\mathcal{R}$ predicts $\Delta s_{g_{15}} < \Delta s_{g_{12}} < \Delta s_{g_{23}} < \Delta s_{g_{45}} < \Delta s_{g_{34}}$, so our updating order is $(j_1, j_5), (j_1, j_2), (j_2, j_3)$. In lines 16-19, we apply O_E on each edge (i, j) of $\mathcal{S}_{E,g}$ and if (i, j) also appears in $\mathcal{S}_E$, we remove it so that it won't be updated again. In lines 21-24, we sort $\mathcal{S}_E$ by the elimination probability P and apply O_E on each event. For example in Fig. 6a, $P_{12} > P_{23}$ so (j_1, j_2) is updated first. We want to emphasize that

although j_1', j_2', j_3' are temporarily at the same location, they are three different points and one of each only connects to three other junctions. Finally, in lines 25-28, we check the graph if it has 2-side grains, whose faces were eliminated during other events. We remove these grains although their area is still larger than 0.

Overall GrainGNN computational complexity. We analyze the complexity of Algorithm 1. The dominant cost of networks $\mathcal{R}$ and $\mathcal{C}$ is the evaluation of graph transformers. In Eq. (2.9), the cost of one matrix-vector multiplication is $O(D_h^2)$; therefore, the cost of one network inference of the entire graph is $O(n_g D_h^2)$. The cost of lines 4-7 is $O(n_g)$. For a graph update, the number of events $|\mathcal{S}_G| \sim O(n_g)$, $|\mathcal{S}_E| \sim O(n_g)$. Sorting $\mathcal{S}_E/\mathcal{S}_G$ in lines 10 and 22 is $O(n_g \log(n_g))$ cost. One O_E or O_G consists of a couple of vertex and edge searches. Each event is $O(n_g)$ complexity. Thus the complexity per graph update is $O(D_h^2 n_g + n_g^2)$. Recall that the time complexity n_l is $O(1)$ so the overall complexity is $O(D_h^2 n_g + n_g^2)$. In our experiments typically we have $n_g < D_h^2$ and the cost of GrainGNN roughly scales linearly with the number of grains n_g.

Algorithm 1 GrainGNN graph-to-graph update algorithm $G_{l-1}(V_{l-1}, E_{l-1}), F_{l-1} \to G_l(V_l, E_l), F_l$
Parameters: ϵ_E and ϵ_G are classification thresholds.

1: /* GrainGNN components $\mathcal{R}$ and $\mathcal{C}$ make predictions */
2: $\Delta F = \mathcal{R}(F_{l-1}, E_{l-1})$ feature changes
3: $P = \mathcal{C}(F_{l-1}, E_{l-1})$ edge event probability
4: $F_l \leftarrow F_{l-1} + \Delta F$
5: /* find events based on thresholds ϵ_E, ϵ_G */
6: $\mathcal{S}_E \leftarrow \{(m,n) \in E_{jj,l-1} : P_{mn} > \epsilon_E\}$ edge events
7: $\mathcal{S}_G \leftarrow \{g \in V_g : s_{g,l} < \epsilon_G\}$ grain elimination events
8: /* update graph with $\mathcal{S}_G$ */
9: $V_l, E_l \leftarrow V_{l-1}, E_{l-1}$
10: $\mathcal{S}_G \leftarrow$ Sort $\mathcal{S}_G$ by $s_{g,l}$ in ascending order
11: **for** $g \in \mathcal{S}_G$ **do**
12: $\mathcal{S}_{E,g} = \{(m,n) \in E_{jj,l} : m, n \in N_g\}$ edge events of eliminated grain
13: $\mathcal{S}_{E,g} \leftarrow$ Sort $\mathcal{S}_{E,g}$ by $\Delta s_{g_{ij}}$ in ascending order
14: $\mathcal{S}_{E,g} \leftarrow$ Remove last two elements
15: /* update graph with $\mathcal{S}_{E,g}$ */
16: **for** $(i,j) \in \mathcal{S}_{E,g}$ **do**
17: $E_l, F_l \leftarrow O_E(E_l, F_l, i, j)$ coordinates of new junctions
18: **if** $(i,j) \in \mathcal{S}_E$ **then**
19: $\mathcal{S}_E \leftarrow$ Remove (i,j) avoid updating again in $\mathcal{S}_E$
20: $V_l, E_l \leftarrow O_G(V_l, E_l, g)$
21: /* update graph with $\mathcal{S}_E$ */
22: $\mathcal{S}_E \leftarrow$ Sort $\mathcal{S}_E$ by P_{ij} in descending order
23: **for** $(i,j) \in \mathcal{S}_E$ **do**
24: $E_l, F_l \leftarrow O_E(E_l, F_l, i, j)$ coordinates of new junctions
25: /* remove 2-side grains */
26: $\mathcal{S}_G' \leftarrow \{g \in V_g : |N_g| = 2\}$
27: **for** $g \in \mathcal{S}_G'$ **do**
28: $V_l, E_l \leftarrow O_G(V_l, E_l, g)$

2.7. Graph-to-image microstructure reconstruction

We conclude with the post-processing step that converts the output GrainGNN to a 3D grain orientation field or directly to quantities of interest. Given G_0, F_0 Algorithm 1 computes the trajectory $(\mathcal{G}, \mathcal{F}) := \{G_l, F_l\}_{l=0}^{n_l}$. We remark that with $\mathcal{G}, \mathcal{F}$ we can directly compute the quantities of interest. For instance, the volume of a grain g is given as $\mathcal{V}_g = \Delta z \sum_{l=0}^{n_l-1} s_{g,l} + v_g$. Therefore size, aspect ratio, and orientation statistics can be readily computed. If needed, we can also reconstruct the pointwise orientation $\boldsymbol{\theta}(x,y,z)$. We first reconstruct slices I_l of the $x-y$ plane at different z_l positions. For each grain $g \in G_l$, we find its junction neighbors N_g and their coordinates. We draw a polygon using the junctions and set $I_l(x,y) = g$ at all interior pixels—at any desired pixel resolution. We repeat it for n_g grains. To combine all the I_l slices we use piecewise constant interpolation in z: we assume that each reconstructed I_l has thickness Δz and then stack them in z-direction to form the 3D grain index field $\mathcal{I}$. We currently neglect the excess volume part of a grain in reconstructed 3D images. The final orientation field $\boldsymbol{\theta}(x,y,z) = \boldsymbol{\theta}_g(\mathcal{I}(x,y,z))$, where $\boldsymbol{\theta}_g$ is the orientation of a grain with index g.

2.8. Discretization and numerical solution of the phase field PDE

We use a second-order finite difference discretization in space and a forward Euler time stepping in time. The phase field code is implemented with Compute Unified Device Architecture (CUDA) in C++. Every grain is associated with a phase field function so one simulation will require storage of n_g phase field functions. To reduce the complexity, we adopt an active parameter tracking (APT) algorithm [58] to reduce the number of phase field variables stored on each grid point to a constant number P. Each grid point stores the largest P numbers of ϕ_α with its index α and treats all the other phase field variables as -1. In each time step at each grid point, only α stored in the point itself and its six direct neighbors compute Eq. (2.1a), and the new P largest ϕ_α are found and stored. APT

reduces the storage by a factor of $n_g/(2P)$. We find $P = 5$ is sufficient for our simulation setup. We also utilize the moving-domain technique [59] to reduce the height of the computational domain.[3] The convergence results of mesh size dx, interface width W_0, the number of phase field variables, and the height of the moving domain are reported in the appendix.

But why an explicit scheme? First, most practitioners [6,52] use explicit schemes so this comparison is the most informative. Second, although there are situations in which linearly-implicit or implicit-explicit solvers are preferable, this is not usually the case because in several rapid solidification regimes either the PDEs are not stiff or the implicit solvers become overly diffusive [53]. Third, implicit solvers complicate the implementation of the active parameter tracking.

3. Results

In this section, we present training and testing experiments for GrainGNN. In Section 3.1 we discuss the data generation for training, the network architecture parameters and number of parameters, metrics of comparison with simulations, and the training accuracy we obtained. In Section 3.2 and Section 3.3, we discuss the testing accuracy. Recall that the input parameters to GrainGNN are $\boldsymbol{p} = \{G_z, R_z\}$ (temperature profile), $\boldsymbol{\xi}$ the grain initial condition at $z_l = z_0$ (substrate of meltpool), and the domain dimensions $L = (L_x, L_y, L_z)$. For the temperature, we select a range of values for G_z and R_z and we select some values for training and different values for testing. The range of values corresponds to temperature profiles in metal alloy rapid solidification conditions. For training L is fixed to $L = L^0$ and $\boldsymbol{\xi}$ is sampled from a fixed distribution. For testing, we consider two settings for L and $\boldsymbol{\xi}$. (1) *In distribution generalization* (Section 3.2) we use $L = L^0$ and $\boldsymbol{\xi}$ is sampled from the same distribution used for training. (2) In our *Out of distribution generalization* experiments (Section 3.3), we use $L \neq L^0$ and $\boldsymbol{\xi}$ is sampled from a different distribution used in training. The latter is critical because although we train for a relatively small number of grains, we can generalize GrainGNN to an arbitrary number of grains and initial conditions without further training.

3.1. Training of LSTM regressor and classifier

Selecting input parameter values for training. We use phase field data of different $\boldsymbol{\xi}$ and $\boldsymbol{p}$ as our training data. For each simulation, we use the same domain size L^0. We try to use as small L^0 as possible to minimize training costs. We choose $L_x^0 = L_y^0 =$ 40 μm to have a sufficient initial number of grains to avoid the effect of periodic boundary conditions. We choose $L_z^0 = 50$ μm to have sufficient grain coarsening. The percentage of eliminated grains at $z_l = 50$ μm is typically 50%−70%. For $\boldsymbol{p}$, we used a uniform grid sampling with G_z values in $(0.5, 10)$ K/μm [4–6], and R_z values in $(0.2, 2)$ m/s [6,60]. The sampling grid is shown in Fig. 10a. The mesh size we use is $\Delta G = 0.5$ K/μm and $\Delta R_z = 0.2$ m/s, thus the total number of sampled $\boldsymbol{p}$ points is 1443. For each $\boldsymbol{p}$ point we sample a different $\boldsymbol{\xi}$. Grain orientations are uniformly sampled from a unit sphere, as discussed in Section 2.1. The sampling of the initial junction coordinates $\boldsymbol{x}_j^0$ is as follows.

We first initialize grains as hexagonal lattices of the same size. Here we set their equivalent diameter $d_0 = 4.1$ μm [61]. The junction coordinates of the hexagonal grid are $\bar{\boldsymbol{x}}_j$. Then we add a perturbation $\boldsymbol{x}_j^0 = \bar{\boldsymbol{x}}_j + 0.1 L_x^0 \eta$ and η is sampled from a Gaussian distribution $\eta \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. For training data, the initial number of grains is in the range of 110−125 and the grain sizes are in the range of 2.7−5.7 μm. For in-distribution generalization, testing $\boldsymbol{\xi}$ is sampled using the same method. For out-of-distribution generalization, we sample $\boldsymbol{x}_j^0$ from a uniform distribution $\boldsymbol{x}_j^0 \sim U(0, 1)$ to test grain sizes with a larger variance.

Generating training data with phase field simulations. To generate training data we use high fidelity phase field simulations with the discretization described in Section 2.8. We use the interface width $W_0 = 0.1$ μm and $dx = 0.8 W_0$. We performed the convergence test to verify that this mesh size provides sufficient accuracy of quantities of interest for the $\boldsymbol{p}$ range we used. The number of grid points used is $500 \times 500 \times 625$. The time step is 2.42 nanoseconds. The simulation is stopped when the SLI reaches 50 μm. The number of time steps varies between 10K for $R_z = 2$ m/s to 100K for $R_z = 0.2$ m/s. The total cost for generating training data was about 15 hours on 24 NVIDIA A100 GPUs.

From each simulation, we extract graph pairs (G_{l-1}, G_l). For $z_l = L_z^0 = 50$ μm, the number of eliminated grains varies between 0−84 so the number of extracted layers per simulation n_l is 2−21. The total number of graph pairs created is 40K (38K for training and 2K for validation). We first train the regressor network $\mathcal{R}$ for 50 epochs with the Adam optimizer [62]. The learning rate was set to 50% decay every 10 epochs. Then we use $\mathcal{R}$'s weights to initialize the encoder of C which is trained for an additional 20 epochs again using the Adam optimizer. The network hyperparameters—D_h, the number of LSTM layers, the batch size, and the initial learning rate—were tuned by grid search. The tuned $\mathcal{R}, C$ have two LSTM layers with $D_h = 96$ for a total 1.2 million weights per network. The total training time was 12 hours on a single A100 GPU. We remark that C and $\mathcal{R}$ run on both CPUs and GPUs. But during inference we also need operations O_E and O_G, which we have implemented only on CPUs. So currently we run our inference only on CPUs and the inference can be further accelerated if we run C and $\mathcal{R}$ on GPUs.

Fig. 7 shows the training losses and accuracy of GrainGNN. For model $\mathcal{R}$, the initial training and validation losses are 0.97 and the final training and validation losses are 0.002. The corresponding accuracy of the output features is shown in Fig. 7c. At epoch 50, the RRMSE for $\Delta x, \Delta y, \Delta s$ and v are 18.7%, 19.5%, 10.6%, and 7.9%, respectively. The average in-plane movement of a junction between two sampled z_l values is roughly 2−3 pixels in x and y direction. Thus, the error of the predicted junction coordinates is

[3] We track a domain with a height smaller than the actual domain height L_z'. The domain is placed around the SLI and moves with the SLI. As the SLI moves one grid point in the z-direction, we add a new layer of liquid on the top of the domain and remove the bottom layer of the solidified part. The removed layer is stored and remains "frozen" till the end of the simulation.

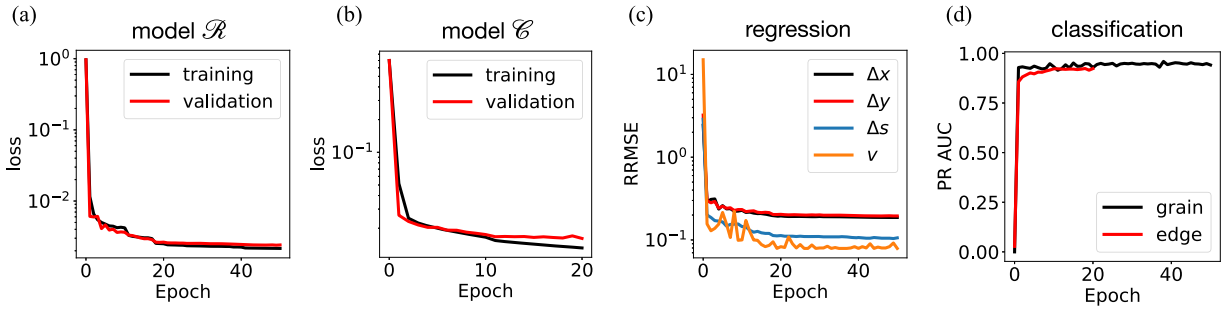

Fig. 7. Training losses and accuracy. (a) Training and validation losses of the regressor $\mathcal{R}$. (b) Training and validation losses of the classifier C. (c) Relative errors (RRMSE) of the regression outputs for validation data. $\Delta x, \Delta y, \Delta s$, and v. (d) The area under the Precision-Recall curve (PR AUC) of the predictions of the grain elimination events and neighbor-switching events.

about half a pixel per graph update. Accuracy of grain events and edges events is shown as the black and red line in Fig. 7d. The final AUCs of grain events and edge events are 0.941 and 0.923 respectively. The optimal classification threshold for grain events we find on the precision-recall curves is $\epsilon_G = 10^{-4}$; the corresponding precision and recall are 0.95 and 0.92. For edge events, we find $\epsilon_E = 0.6$; precision is 0.91 and recall is 0.87.

Next we evaluate the accuracy of GrainGNN-predicted microstructure. I_0 is the PDE-to-image translation of $\boldsymbol{\xi}$, the initial condition for both GrainGNN and phase field PDE. We extract G_0, F_0 from I_0 and compare the final predicted microstructure with phase field simulations. We use a pixel misclassification rate (MR) to measure pointwise errors. MR(z_l) is as the number of grid points classified to a wrong grain, normalized by the total number of pixels:

$$\mathrm{MR}(z_l) = \frac{1}{|I_l|} \sum_i \sum_j \mathbb{1}_{I_{l,\mathrm{PF}}(i,j) \neq I_{l,\mathrm{GNN}}(i,j)}. \tag{3.1}$$

MR is zero when all grain boundaries are exactly reconstructed. For the quantities of interest, we measure the distributional error for grain sizes predicted by phase field simulations and GrainGNN. We use the two-sample Kolmogorov–Smirnov (KS) statistics $\mathrm{KS} = \sup_x \left| D_{\mathrm{PF}}(x) - D_{\mathrm{GNN}}(x) \right|$. D represents the empirical cumulative distribution function (eCDF) of grain sizes. KS statistics quantify the maximum discrepancy between the predicted and the ground truth grain size distributions. The smaller the KS value is the closer the two distributions are. More details about how to interpret KS statistics are provided in the appendix.

3.2. In-distribution generalization

By in-distribution generalization we refer to GrainGNN inference for $L = L^0$, unseen (during training) $\boldsymbol{p}$ values, and unseen $\boldsymbol{\xi}$ values sampled by the same procedure we used for training. We show an example of test case in Fig. 8 with $\boldsymbol{p} = (1.904\ \mathrm{K/\mu m}, 0.558\ \mathrm{m/s})$. Using nearest neighbor interpolation in $\boldsymbol{p}$ space we determine that GrainGNN should be used with $\Delta z = 2.4$ μm, which determines the number of GrainGNN steps. Fig. 8a shows the grain microstructure at four different heights. At each height z_l, we show the G_l and I_l with 500×500 spatial resolution. We compare the reconstructed image with the phase field data at the same height. The misclassified grid points are marked in dark red. The error images indicate that the GrainGNN predictions are accurate representations of the microstructure obtained by our phase field simulations. The last two columns compare the evolution of the 3D microstructure for GrainGNN and phase field predictions. We select two grains to illustrate the grain shape evolution. Fig. 8b-e are different measures of GrainGNN accuracy. Fig. 8b shows the cumulative grain elimination events when the SLI reaches different heights. The blue, red, and dashed red lines depict grain eliminations that happened in the phase field simulation only, GrainGNN inference only, and in both, respectively. We can see that the number of eliminations predicted by GrainGNN is overall accurate but lagged after $z_l = 24$ μm compared to the phase field result. At $z_l = 50$ μm, three grains that should be eliminated still exist on the graph, and only one grain is falsely eliminated by GrainGNN. The precision and recall for grain events are 72/73 and 72/75 respectively. Fig. 8c plots MR for reconstructed images at different heights. MR is 2.4% initially at $z_0 = 2$ μm because the curvature of the grains is neglected in reconstructed images. MR is 11.3% at $z_l = 50$ μm. Fig. 8d shows the evolution of the volume-averaged misorientation as the height of the SLI increases; the curve is well captured by GrainGNN. The initial average misorientation angle is 23.8° and it decreases to 13.8°, which indicates the grains are more aligned with the temperature gradients after the epitaxial growth. Fig. 8e shows the grain size distribution when the SLI reaches 50 μm. The final average grain size is 9.9 μm. The KS statistic between the phase field and GrainGNN distributions is 0.034, which means the two distributions are nearly identical.

We test the accuracy of GrainGNN inference for different values of $\boldsymbol{p}$. In our first test, we select four values of $\boldsymbol{p}$, each of which we sample ten $\boldsymbol{\xi}$ to compute microstructure statistics. In Fig. 9, the 2D images show the grain microstructure at height $z_l = 50$ μm. We can see that lower G_z and higher R_z result in more grain eliminations and more misclassified pixels. The MR average and standard deviation at $z_l = 50$ μm for the four values of $\boldsymbol{p}$ are 16.2%±1.7%, 23.1%±3.2%, 8.6%±0.5%, and 18.8%±2.3%, respectively. The blue and red lines are the mean and standard deviation of the phase field and GrainGNN predictions. The KS average and standard deviation at $z_l = 50$ μm are 4.7%±0.7%, 5.4%±0.9%, 5.2%±0.9%, and 4.4%±0.6% for the four $\boldsymbol{p}$ values.

We further test GrainGNN's MR accuracy on 100 randomly selected $\boldsymbol{p}$ with a single random $\boldsymbol{\xi}$ for each value of $\boldsymbol{p}$ (see Fig. 10a). Fig. 10b show the MR averaged over z_l for each testing case. MR increases with increasing R_z and decreasing G_z; the highest MR

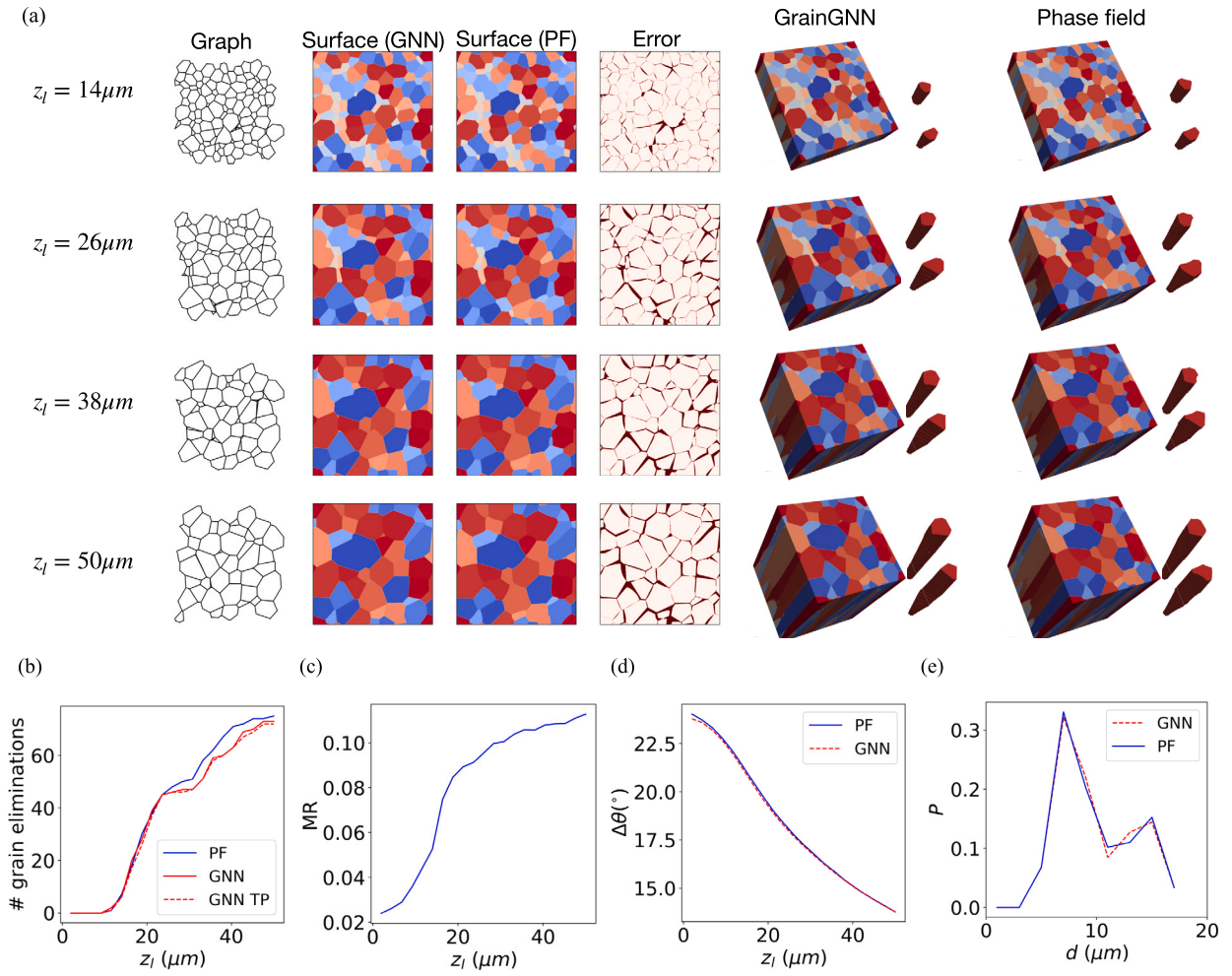

Fig. 8. GrainGNN prediction for one testing case with $G_z = 1.904$ K/μm and $R_z = 0.558$ m/s. (a) The time evolution of the grain microstructure when the solid-liquid interface z_l reaches different heights. At each height from left to right, we show the graph that GrainGNN predicts, the reconstructed microstructure at the current height, the corresponding phase field simulation, the pointwise error between the GrainGNN image and the phase field result, the GrainGNN grain structures which are formed by stacking reconstructed images, and the phase field microstructures. We also explicitly show two of the grains to illustrate the evolution of the grain shape. (b) The number of accumulated grain elimination events when SLI reaches different heights. The solid blue line is the phase field result. The red line is predicted by GrainGNN. The red dashed line is the number of true positive (TP) events among the elimination events predicted by GrainGNN. (c) Misclassification rate (MR) at different heights. (d) Evolution of volume-average misorientation as the SLI reaches different heights. (e) The grain size distribution when $z_l = 50$ μm. The blue solid line and the red dashed line are phase field and GrainGNN predictions, respectively.

is 17% for $G_z = 2.450$ K/μm, $R_z = 1.753$ m/s and $G_z = 3.764$ K/μm, $R_z = 1.901$ m/s. We didn't find a correlation between KS and $\boldsymbol{p}$; the maximum KS is 0.11 for $G_z = 5.826$ K/μm, $R_z = 1.165$ m/s. Fig. 10d shows MR as a function of the height z_l. At the height $z_l = 50$ μm, the mean and standard deviation are 15.3%±3.8% for 100 $\boldsymbol{p}$. The total number of grain elimination events across the 100 cases is 6,597; the number of true positives predicted by GrainGNN is 6,088; the number of false positives is 124.

3.3. Out-of-distribution generalization

Recall that $L^0 = (40\ \mu m, 40\ \mu m, 50\ \mu m)$, mean substrate grain size $d_0 = 4.1$ μm, and grain orientations from the unit sphere. We examine GrainGNN's ability to predict 3D grain formation and QoIs for larger L without changing the $\boldsymbol{\xi}$ distribution. We also test GrainGNN's accuracy for different $\boldsymbol{\xi}$ distributions. Note that these generalizations do not require any retraining of GrainGNN and only involve small modifications when $L > L^0$. We explain each generalization below.

Domain width L_x, L_y: A larger domain width with the same grain size distribution is equivalent to a larger number of grains. As discussed in Section 2.3, the dimensional features in Eq. (2.5) and Eq. (2.4), for example x_j, y_j, s_g, v_g are normalized by the training domain size L_x^0, L_y^0, L_z^0. Although we change the domain size, we still normalize with L^0. As shown in Fig. 11a, we have a testing case with domain width $L_x = L_y = 120$ μm. After normalization, the junction coordinates are in the range of [0, 3]. As the transformer encodes *relative* distances between a vertex and its neighbors, adding offsets to the coordinates won't change the hidden

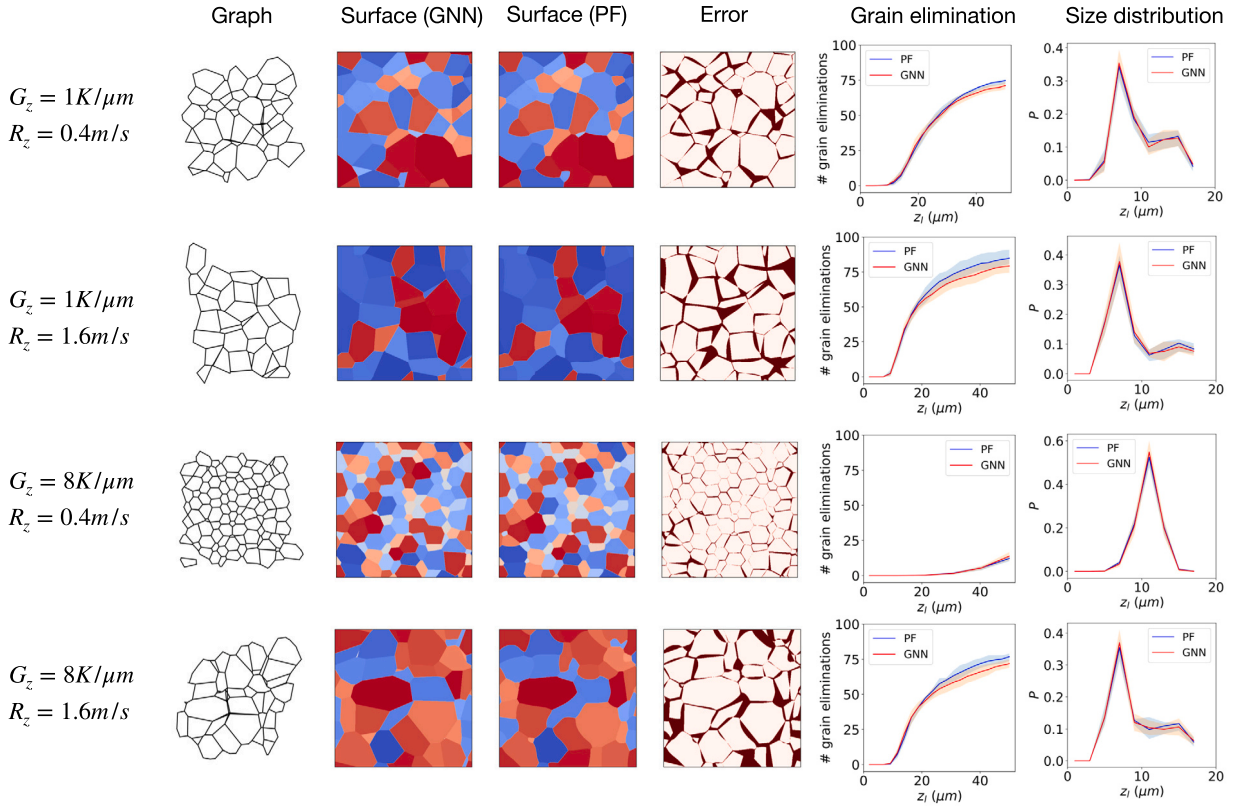

Fig. 9. GrainGNN predictions compared to phase field simulations of different p. We run 10 ξ for each p. The simulations are initialized with the same ξ distribution. The 2D images show the grain structures at height $z_l = 50$ μm. Different p result in different numbers of grain eliminations and final grain size distributions. Each blue/red line shows the mean and standard deviation of the 10 ξ realizations.

states output by the encoder. Thus, we expect no difficulties in using a large domain. We simply evolve GrainGNN on the entire graph and reconstruct the microstructure from the evolved graph.

Fig. 11a shows an example of GrainGNN prediction for a case with domain size (120 μm, 120 μm, 50 μm). $G_z = 10$ K/μm, $R_z = 2$ m/s. The initial number of grains is 1043. At the end of the simulation, the number of eliminated grains is 704, of which 644 grains are predicted by GrainGNN. The MR at the top layer is 18.2% and the KS statistic for the grain size distribution is 0.021. We further test 20 randomly sampled pairs of (G, R) with domain size (80 μm, 80 μm, 50 μm), and repeat for domain size (120 μm, 120 μm, 50 μm). Their MR statistics are shown in Fig. 10f and Fig. 10e. At $z_l = 50$ μm, their average MR and standard deviation are 14.4%±4.0% and 13.9%±3.6%, respectively. Compared to 15.3%±3.8% for in-distribution generalization (Fig. 10d), we don't observe a noticeable increase in pointwise error when increasing the domain width. The accuracy of accumulative grain events is 92.1% (4,271/4,638) for $L_x = 2L_x^0$ and 92.6% (8,956/9,672) for $L_x = 3L_x^0$, which is close to 92.3% for $L_x = L_x^0$. In Fig. 11c, we showcase a GrainGNN inference with a large domain width $L_x = L_y = 10L_x^0$. The total number of grains is 11,600. The number of vertex features of the graph is 313K. The final grain microstructure only requires 15 iterations of the graph, which takes about 220 seconds on one CPU. If we run a phase field simulation with the same configuration, the total number of grid points is $5000 \times 5000 \times 100 = 2.5$ billion. At each grid point if we store five phase field variables and their grain indices, the storage required is 25 billion numbers. The number of time steps for the phase field solver is 50K.

Domain height L_z: In practice the domain height L_z is determined by the meltpool depth. Here we test the ability of GrainGNN to generalize for $L_z > L_z^0$ by simply taking more iterations. Recall that we used $L_z^0 = 50$ μm. We assume for $z_l > L_z^0$, the graph evolution follows the same pattern as the previous graph step. In the network inference when $z_l > L_z^0$, we set z_l as L_z^0 in Eq. (2.4) and Eq. (2.5). As shown in Fig. 10f, we extend the predictions of the 20 runs with width $L_x = 2L_x^0$ to the height $z_l = 98$ μm. From $z_l = 50$ μm to $z_l = 98$ μm, MR increases from 14.4%±4.0% to 26.6%±11.6% and the grain elimination accuracy drops from 92.1% to 90.6% (5,729/6,323). We can see from 50 to 90 μm MR increases almost linearly with the distance the SLI traveled. From 90 to 98 μm, the jump of the MR is due to the drastic increase of error for a case with $G_z = 1.774$ and $R_z = 1.471$, whose MR reaches 71.5% at $z = 98$ μm. Fig. 11b shows one case with $G_z = 4.827$ K/μm, $R_z = 0.690$ m/s. The KS of grain size distribution is 0.024 for $z_l = 50$ μm and 0.028 for $z_l = 98$ μm.

Initial grain size and orientation: We set $L = (80\ \mu m, 80\ \mu m, 50\ \mu m) \neq L^0$ and also vary ξ. As mentioned in Section 3.1, the initial junction coordinates are randomly selected from [0, 1]. We change the average grain size by varying the number of sampled junctions. Fig. 12 shows the initial ($z = 0$) and final ($z = L_z$) grain distributions under $G_z = 4$ K/μm and $R_z = 0.8$ m/s; we also test another four p with the values in Fig. 9. For each p, we test ten different size distributions with mean ranging from $d_0 = 2$ μm

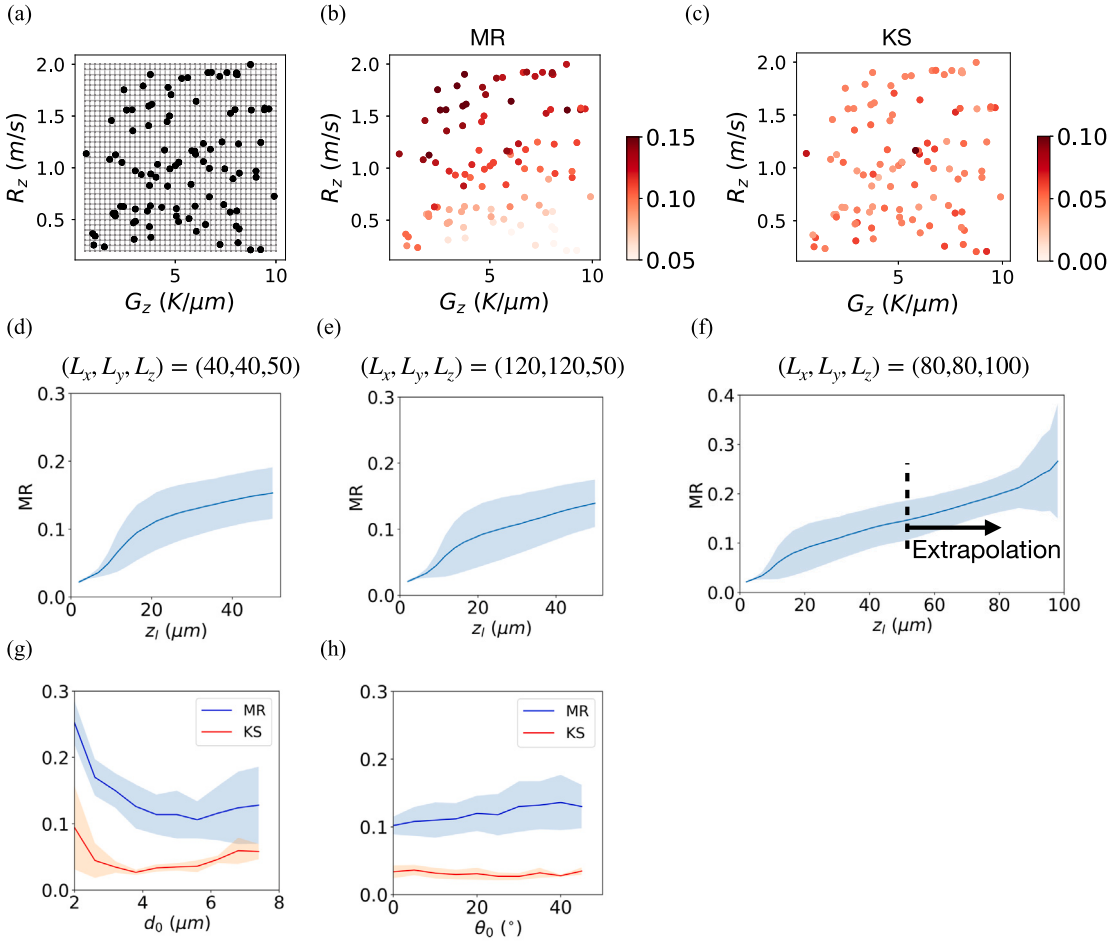

Fig. 10. Error statistics for in-distribution and out-of-distribution generalization. (a) Black lines are $\boldsymbol{p}$ grid used for training and validation. Black dots are 100 $\boldsymbol{p}$ values for testing. (b) Misclassification rate (MR) averaged across sampled heights z_l for each testing $\boldsymbol{p}$. (c) KS statistics (KS) of grain size distribution at the end of simulations for testing $\boldsymbol{p}$. (d) MR mean and standard deviation for 100 testing $\boldsymbol{p}$ with domain size (40 μm, 40 μm, 50 μm). (e) MR for domain size (120 μm, 120 μm, 50 μm). (f) MR for domain size (80 μm, 80 μm, 100 μm). (g) MR and KS for different initial mean grain sizes d_0. (h) MR and KS for different initial grain orientation distributions. θ_0 is the most frequent misorientation angle between the grain orientation and the z-axis.

(Fig. 12a) to $d_0 = 7.4$ μm (Fig. 12b). Their MR and KS are shown in Fig. 10g where each point is averaged across five different $\boldsymbol{p}$. For d_0 in the 4-6 μm range, MR and KS are close to in-distribution errors. MR is higher for runs with smaller grains and is almost doubled for $d_0 = 2$ μm. One reason is smaller grains have a higher surface area-to-volume ratio thus higher percentage of pixels will update their grain indices under the same physical parameters. Another reason is smaller d_0 has higher grain and edge event densities thus introducing larger errors. For larger grains $d_0 > 6$ μm, we observe higher variance for different $\boldsymbol{p}$. For $G_z = 1$ K/μm and $R_z = 1.6$ m/s, MR increases when $d_0 > 6$ μm while MR decreases for the other four $\boldsymbol{p}$. The overall MR for all d_0 is 13.9% and grain elimination accuracy is 92.3% (14,590/15,801).

In the modified orientation distribution, we select dominant grain misorientation angle θ_0 with respect to the z-axis. We vary θ_0 from 0 to $\pi/4$ with ten values sampled for each $\boldsymbol{p}$ and we sample the five $\boldsymbol{p}$ as discussed for grain size distribution. Fig. 12c and Fig. 12d show the initial orientation distribution with $\theta_0 = 0$ and $\theta_0 = \pi/4$ respectively. As shown in Fig. 10h, MR is slightly higher for larger θ_0, which is associated with more grain eliminations we observed for misaligned grains. KS is almost the same for different θ_0. The overall MR and grain elimination accuracy are 12.0% and 91.0% (10,529/11,569), respectively.

4. Discussion

4.1. Accuracy

These results suggest that GrainGNN can predict well the microstructure evolution and the statistics of quantities of interest when compared to the phase field simulations. It can also generalize to unseen G_z and R_z values, domain width and height, and initial grain configurations—without retraining. For pointwise microstructure comparisons, the in-distribution generalization errors are in the 4%–17% range, with higher errors occurring when we have more grain eliminations. The out-of-distribution error is not

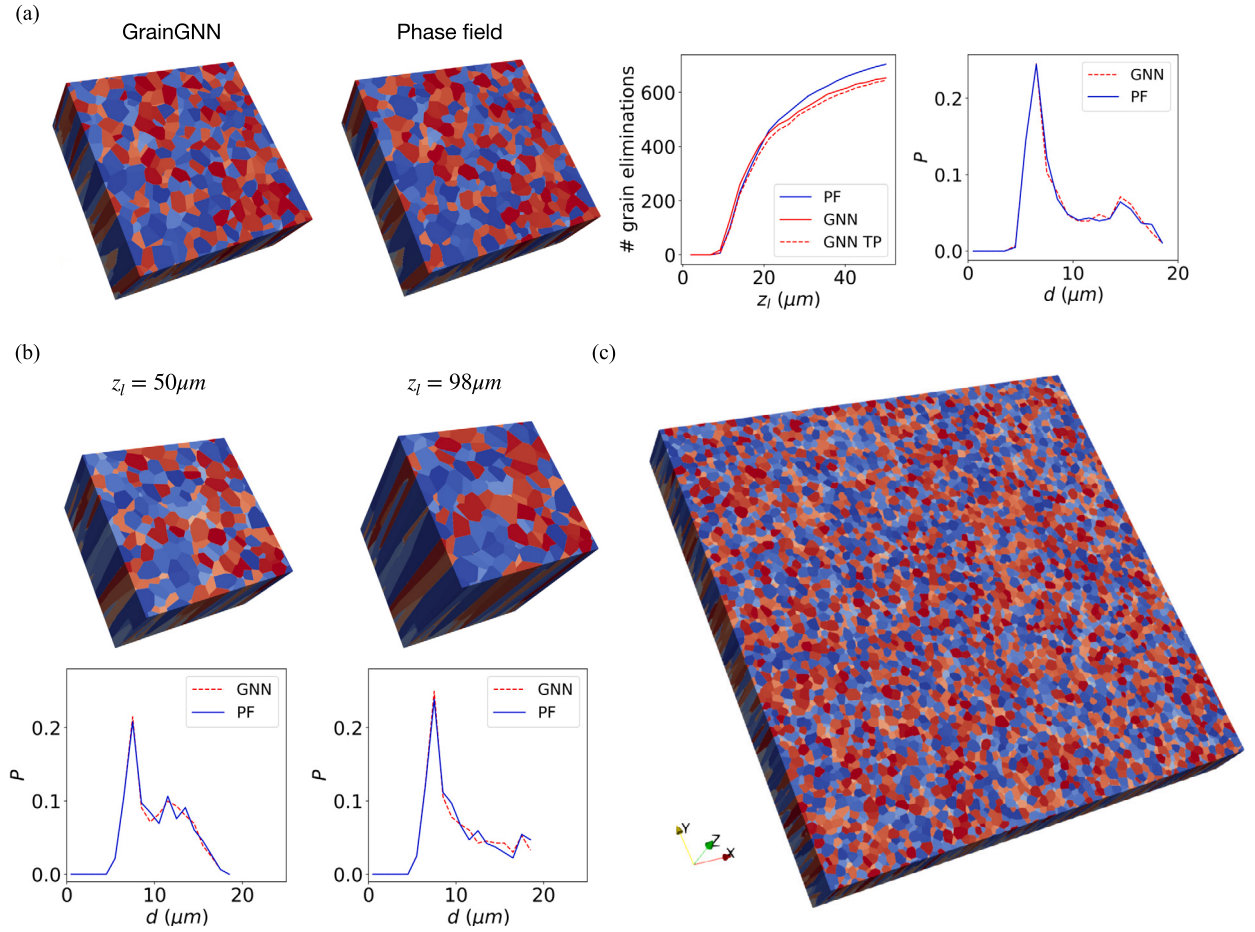

Fig. 11. Generalization with the domain size and the number of grains. (a) Domain size (120 μm, 120 μm, 50 μm) with 1043 grains. $G_z = 10$ K/ μm, $R_z = 2$ m/s. The total number of eliminated grains is 704, among which 644 grains are predicted by GrainGNN. The KS statistic for the grain size distribution is 0.021. (b) Domain size (80 μm, 80 μm, 100 μm) with 461 grains. $G_z = 4.827$ K/ μm, $R_z = 0.690$ m/s. The left and right plots are GrainGNN predictions at $z_l = 50$ μm and $z_l = 98$ μm, respectively. Their KS for grain size distribution are 0.024 and 0.028. (c) GrainGNN prediction for a case with domain size (400 μm, 400 μm, 50 μm), which is 100 times the training domain size. The total number of grains is 11600. $G_z = 2$ K/μm, $R_z = 0.4$ m/s. The GrainGNN inference time is 220 seconds on one CPU.

sensitive to the domain width or the number of grains but increases with decreasing initial grain size. The pointwise errors increase gradually with the height of the SLI. When comparing the statistics quantities of interest, we find the distributional error of grain size distribution is not sensitive to G_z, R_z, and the domain size. The KS statistics are in the 0.03–0.06 range. In all test cases, grain elimination accuracy is above 90%. In total, we ran 280 test phase field simulations with *random* initial realization and the number of grains ranging from 100 to 1600. Demonstrating its robustness, GrainGNN successfully completed all 280 inferences, among which only one test case, with $L_z = 2L_z^0$, yielded significantly different results.

We compare GrainGNN's accuracy with three other models, (i) a graph convolution LSTM (GC-LSTM) [48], (ii) a graph transformer operator (TransformerConv) [57], and (iii) a graph convolutional operator (GraphConv) [43]. We use the same training data and 2.4 million weights for all models. The metrics we used are RRMSE for $|\Delta \boldsymbol{x}| = \sqrt{\Delta x^2 + \Delta y^2}$ and Δs, AUC for both events, and the average MR for the 100 testing simulations. Compared to GC-LSTM, GrainGNN adds the attention coefficient $\beta_{i,k}$ in Eq. (2.9). As shown in Table 2, GrainGNN outperforms GC-LSTM for all the metrics. The self-attention mechanism of GrainGNN is important to learn the spatial correlations between different nodes. If we drop LSTM and only keep the TransformerConv as an encoder, regression accuracy RRMSE-$|\Delta \boldsymbol{x}|$ and RRMSE-Δs decrease significantly. GraphConv has a higher AUC-G but lower RRMSE-$|\Delta \boldsymbol{x}|$, RRMSE-Δs, and AUC-E compared to TransformerConv. We also compute the average MR of 100 testing simulations for different models and GrainGNN produces the most accurate microstructure images.

We also investigate the effect of the amount of training data on the GrainGNN accuracy. From Table 2, adding more training data generally improves the accuracy of GrainGNN. The regression model $\mathcal{R}$ is less sensitive to the size of training data. From 5K training pairs to 38K training pairs, the relative error of $|\Delta x|$ and Δs improve 1% every doubling the training size, and the AUC for grain elimination events stays around 94%. In contrast, the AUC for the neighbor-switching events improves significantly from 85.2% to 92.3%. The difference between the two types of events is that the neighbor-switching events are highly imbalanced. The positive-to-negative ratio is about 1 to 30. Because the positive neighbor-switching events are deficient, adding more training graph

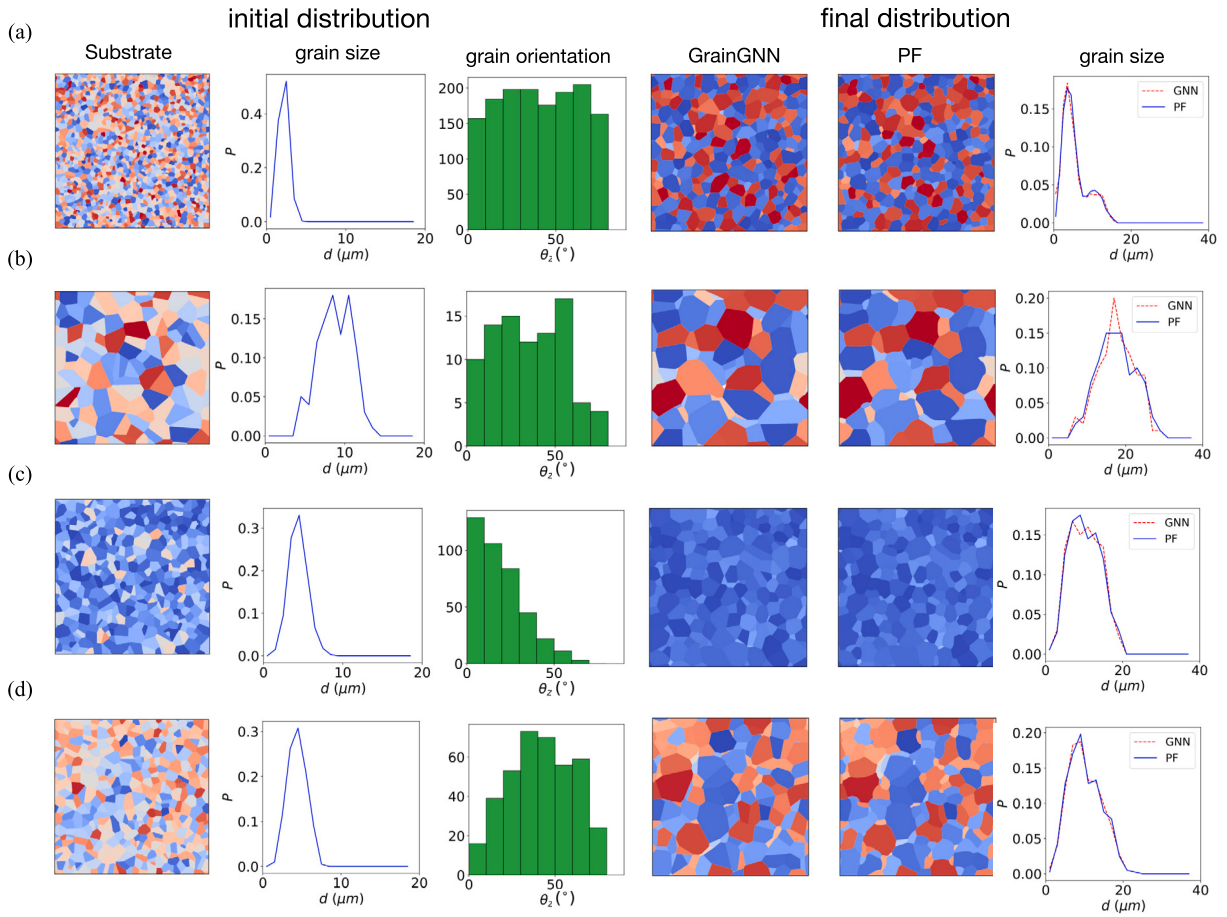

Fig. 12. Generalization with different initial grain configurations. We use $L_x = L_y = 80$ μm, $G_z = 4$ K/μm, and $R_z = 0.8$ m/s for all four cases shown above. (a) Initial size distribution with mean $d_0 = 2$ μm. (b) Initial size distribution with mean $d_0 = 7.4$ μm. (c) Initial orientation distribution with the maximum frequency at $\theta_0 = 0°$. (d) Initial orientation distribution with the maximum frequency at $\theta_0 = 45°$.

pairs can largely enhance the classification precision. To deal with the data imbalance, we have tried to increase the weights of the positive events in Eq. (2.14) and downsampled the negative events; the improvement, however, is limited. We plan to use more data augmentation techniques to balance the labels and improve the accuracy of edge events.

The ordering of grain and edge events when updating the graph is another factor that affects the accuracy of the prediction. We currently choose the sorting method based on the classifier probabilities; we have not tested alternatives. One constraint of using Algorithm 1 is the input graph can only have triple junctions and the output graph is guaranteed to have only triple junctions. Algorithm 1 has two noteworthy limitations, although they do not seem to limit the overall GrainGNN accuracy. One is the approximation of new coordinates of j_1, j_2 in an edge event. The other is GrainGNN doesn't guarantee planar graph outputs, which means non-physical edge intersections are possible. The intersections could cause grains to have overlapping areas or create holes in the reconstructed images. We currently ignore this issue since it won't break Algorithm 1 or calculations of quantities of interest.

4.2. Computational efficiency

GrainGNN achieves a substantial reduction in the storage of variables and the number of required time steps, which leads to significant speedups over phase field simulations. For the training domain size, the required number of phase field variables is about 500M despite using the active parameter tracking and moving-domain algorithms, while the number of features of GrainGNN is about 3.2K. The network has about 2.4M parameters. Thus, GrainGNN requires $10^2 - 10^5$ times less storage than the phase field solver. The computation efficiency of our phase field solver and GrainGNN is listed in Table 3. Our phase field solver is optimized with GPUs [53]. For the G_z and R_z values investigated in this paper, the time cost per phase field simulation is 300−3000 seconds on one A100 GPU, while the time cost of GrainGNN is 0.2−3 seconds. GrainGNN on a single CPU achieves 150×–2000× speedup over our phase field code.

The number of GrainGNN iterations n_l scales linearly with the number of eliminated grains; therefore n_l is a function of G_z and R_z. In Table 3, $n_l = 5$ for $G_z = 8$ K/μm and $R_z = 0.4$ m/s and $n_l = 20$ for other three parameters. We also compute the time consumed by network inferences with the domain width of $2L_x^0, 3L_x^0, 10L_x^0$. The number of grains per simulation is roughly 400,

Table 2
Model accuracy comparison for three network architectures and different numbers of training graph pairs. The total number of trainable parameters is roughly 2.4 million for all the models. For the regression tasks, we compute the Relative Root Mean Square Error (RRMSE) of $|\Delta \boldsymbol{x}|$ and Δs; for the classification tasks, we compare the area under the precision-recall curve (AUC) for neighbor-switching events (AUC-E) and grain elimination events (AUC-G). These metrics are evaluated on the same validation datasets. We also compute the average of misclassification rate (MR) for the 100 testing simulations with different $\boldsymbol{p}$.

model	#training pairs	RRMSE-$\|\Delta \boldsymbol{x}\|$	RRMSE-Δs	AUC-E	AUC-G	MR
GrainGNN: LSTM + TransformerConv	5K	0.225	0.143	0.852	0.938	0.140
	10K	0.208	0.128	0.864	0.947	0.127
	20K	0.197	0.119	0.892	**0.953**	0.121
	38K	**0.187**	**0.106**	**0.923**	0.941	**0.106**
LSTM + GraphConv [48]	38K	0.231	0.137	0.908	0.930	0.122
TransformerConv [57]	38K	0.311	0.322	0.902	0.919	0.201
GraphConv [43]	38K	0.388	0.394	0.847	0.936	0.250

Table 3
Computational efficiency of GrainGNN for different $\boldsymbol{p}$ and domain size L. Time for solving phase field equations was measured on one NVIDIA A100 GPU with 40 GB memory. The domain size is (40 μm, 40 μm, 50 μm). GrainGNN inference time was measured on a single AMD EPYC 7763 CPU core. $a_x \times a_y \times a_z$ represents the GrainGNN inference time for a domain size of $(a_x L_x^0, a_y L_y^0, a_z L_z^0)$. Every measured time is averaged across 10 initial realizations of grains.

Physical parameters		storage		time (seconds)					
G_z (K/μm)	R (m/s)	PF	GrainGNN	PF	GrainGNN	$2\times2\times1$	$3\times3\times1$	$10\times10\times1$	$2\times2\times2$
1	0.4	500M	3.2K	1474.6	2.9	10.2	23.2	297.0	15.5
1	1.6			478.7	2.6	8.2	20.3	271.2	13.2
8	0.4			1440.9	0.9	3.5	7.8	69.5	6.4
8	1.6			413.4	2.8	9.6	21.8	278.7	15.1

1000, and 10,000, respectively. We can see the time cost scales linearly with the domain size and with the number of grains. If further increasing the number of grains to $n_g > 10^4$, the $O(n_g^2)$ graph update may start to dominate the cost. In this case, we plan to store a hash table of a node to its neighbors. Thus the cost per event can be reduced from $O(n_g)$ to $O(1)$. Another benefit of GrainGNN is that the cost doesn't increase with the grain size and number of grid points per grain. With increasing L_z the graph size and inference time decrease due to grain eliminations. The inference time for $L_z = 98$ μm is only approximately 1.5 times the inference time for $L_z = 50$ μm (compared to approximately 2 times, if each height increment required a fixed amount of time).

4.3. Extensions and limitations

There are several planned extensions to the current framework of GrainGNN. One is generalizing the computational domain to no-flux boundary conditions and non-rectangular geometries. For no-flux boundary conditions, we consider padding domain boundaries with halo grains that have mirrored properties (e.g., orientation) with respect to the grains on the boundary. The number of padding grains depends on the domain and physical parameters. For domain geometry, we plan to follow the rectangular-to-curvilinear domain mapping strategy presented in GrainNN [17]. The idea is to find geometric coefficients that map a curved surface to a 2D plane. We run GrainGNN in a rectangular domain and use the geometric coefficients to scale the network outputs for example the junction displacements. The output structure is mapped back to the original geometry. A second extension is to improve the representations of grain boundaries. We neglect the in-plane curvature of the grain boundaries, which can be significant for high R_z values. Another goal is to use experimental design and active learning [63,64] methods for sampling physical parameters used for training. Currently, we use a uniform grid of G_z and R_z to generate training data. The MR plot in Fig. 10 indicates that more data should be drawn from the region with high R_z and low G_z to reduce the pointwise error. To make the data generation computationally trackable for higher dimensional parameter space, an efficient adaptive sampling algorithm needs to be developed. From the network architecture perspective, the current one-to-one LSTM prediction can be extended to sequence-to-sequence prediction to improve accuracy as we did in [17]. However it is unclear how to handle topological changes in sequence-to-sequence configurations.

Limitations. The two major limitations of GrainGNN are dealing with complex meltpool geometries and ignoring grain nucleation. We discussed geometry in the previous paragraph. Grain nucleation introduces new grains to the system thus adding vertices and edges to the graph. The modifications of the graph are two kinds depending on the nucleation density. For dilute nucleation, we can still utilize GrainGNN. We will introduce a new operation O_N that adds vertices and edges, which will be the inverse operation of O_G. For dense nucleation, the equiaxed growth dominates the grain formation. This evolution is substantially different enough from epitaxial growth that a third network (beyond $\mathcal{R}$ and C) will have to be added to predict the graph generation during grain nucleation. A third more fundamental limitation is detecting failure cases, in short some kind of a posteriori error estimates for network predictions. GrainGNN doesn't have any performance guarantees other than the empirical evaluation we discussed. A fourth

limitation is that the inference phase of GrainGNN runs only CPUs, and thus potential speedups are still possible. We are currently working to address these challenges.

5. Conclusions

We presented a surrogate for microstructure evolution in 3D epitaxial grain formation. We introduced a heterogeneous graph model and hand-crafted graph features that combined to achieve a significant spatiotemporal compression of grain microstructure. We proposed an image-to-graph method to extract graphs from phase field data. We modeled microstructure formation with graph-to-graph evolution, where we decomposed the evolution into feature changes and topological events. GrainGNN is implemented with an LSTM-based regressor and classifier to predict the features and events. The LSTMs include graph transformers and can scale to a larger number of grains than those used for training. We also proposed graph and microstructure reconstruction methods to address topological changes. GrainGNN is trained with phase field data in a wide range of G and R values in AM process conditions. It can predict both quantities of interest and pointwise accurate microstructure for unseen process parameters and initial grain configurations. We also showed that GrainGNN generalizes to domain size, number of grains, and initial grain parameters. GrainGNN is orders of magnitude faster than high fidelity simulations and scalable to a large number of grains.

What about training costs? Consider 3D printing a 1 cm^3 volume part with an average 10^6 μm^3 meltpool. Directly simulating solidification of the entire part would require one million phase field meltpool solidification calculations—without accounting for ensemble calculations and meltpool overlaps. Such calculations are currently infeasible. A GPU-optimized GrainGNN on a multi-GPU leadership system could perform such a calculation in less than a day. Our long-term goal is to train GrainGNN using 1000s of phase field simulations using a small domain and then, by accounting for meltpool geometry, deploy it for inference for entire-part microstructure prediction. Thus, training costs will be amortized across the entire build and insignificant compared to the potential overall speedup.

CRediT authorship contribution statement

Yigong Qin: Conceptualization, Data curation, Investigation, Methodology, Software, Visualization, Writing – original draft. **Stephen DeWitt:** Conceptualization, Methodology, Writing – review & editing. **Balasubramaniam Radhakrishnan:** Conceptualization, Methodology, Writing – review & editing. **George Biros:** Conceptualization, Funding acquisition, Project administration, Resources, Supervision, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This material is based upon work partially supported by NSF award OAC 2204226 and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program, Mathematical Multifaceted Integrated Capability Centers (MMICCS) program, under award number DE-SC0023171. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the DOE, and NSF. Computing time on the Texas Advanced Computing Centers Stampede system was provided by an allocation from TACC and the NSF.

Appendix A. Phase field solver

The material parameters of stainless steel 316L are listed in Table A.1. Given values of Γ, L_f/C_p, and μ_k, phase field parameters λ and τ_0 can be derived using asymptotic analysis [53,56]:

$$\frac{\Gamma C_p}{L_f} \equiv d_0 = a_1 \frac{W_0}{\lambda}, \tag{A.1}$$

$$\frac{C_p}{\mu_k L_f} \equiv \beta_0 = a_1 \frac{\tau_0}{\lambda W_0} - a_1 a_2 \frac{W_0}{D_h}, \tag{A.2}$$

where $a_1 = 5\sqrt{2}/8$, $a_2 = 47/75$; d_0 is the thermal capillarity length; β_0 is the kinetic coefficient; D_h is the heat diffusion coefficient. The obstacle parameter ω in Eq. (2.1a) is set to $\lambda u/I$, where u is the nondimensional undercooling and I is a constant, here we choose $I = 1/12$ [3]. The anisotropy of τ_α requires the spatial derivatives of ϕ in the rotated coordinates with angle θ. Let θ_z be the

Table A.1
Material parameters for stainless steel 316L.

symbol	meaning (units)	value [4,6]
Γ	Gibbs-Thompson coefficient (Km)	3.47×10^{-7}
L_f/C_p	latent heat/heat capacity (K)	229
μ_k	linear kinetic coefficient (m/s/K)	0.217
λ	thermal coupling constant	58.3
W_0	length scale (μm)	0.1
τ_0	time scale (ns)	40
T_M	melting temperature (K)	1783
ϵ_k	kinetic anisotropy coefficient	0.11
k	partition coefficient	0.791
D_h	heat diffusion coefficient (m^2/s)	3.6×10^{-6}
ΔT_0	freezing range (K)	15.7
v_a	absolute stability velocity (m/s)	0.17

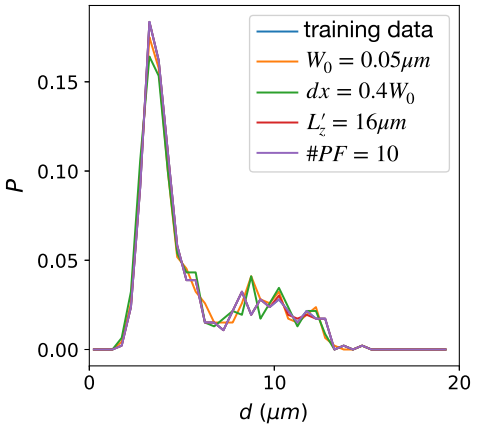

Fig. A.1. Convergence test for a phase field simulation. $G_z = 10$ K/μm $R_z = 2$ m/s. The number of grains is 463. The simulation parameters used for generating the training data (blue line) are interface width $W_0 = 0.1$ μm, mesh size $dx = 0.8W_0$, moving-domain height $L_z' = 8$ μm, and five phase field variables per grid point. The yellow and green lines show the convergence of the discretizations by halving W_0 or dx. The red line shows the convergence of the moving-domain algorithm by doubling the moving-domain width. The convergence of the blue and purple lines indicates the negligible difference between storing five phase fields and ten phase fields.

angle between the z-axis and the preferred growth direction $<100>$ and θ_x be the angle between the x-axis and the projection of $<100>$ direction on the x-y plane. The derivatives appearing in Eq. (2.2b) should be replaced with:

$$\begin{bmatrix} \partial_x \phi' \\ \partial_y \phi' \\ \partial_z \phi' \end{bmatrix} = \begin{bmatrix} \cos(\theta_x)\cos(\theta_z) & \sin(\theta_x)\cos(\theta_z) & -\sin(\theta_z) \\ -\sin(\theta_x) & \cos(\theta_x) & 0 \\ \cos(\theta_x)\sin(\theta_z) & \sin(\theta_x)\sin(\theta_z) & \cos(\theta_z) \end{bmatrix} \cdot \begin{bmatrix} \partial_x \phi \\ \partial_y \phi \\ \partial_z \phi \end{bmatrix} \tag{A.3}$$

Because the capillary anisotropy is much weaker than kinetic anisotropy in the rapid solidification regime, we set $\epsilon_c = 0$.

The phase field solver is available at https://github.com/YigongQin/cuPF. The CUDA functions have been optimized. The most expensive right-hand side calculations have the performance of 270 GFlops on one Nvidia A100 GPU. We use CUDA-aware MPI which scales well on multiple GPUs and multiple nodes.

Fig. A.1 shows the convergence results for dx, W_0, the height of the moving domain, and the number of phase field variables per grid point used in the active parameter tracking algorithm. The training data is generated using the configuration: $W_0 = 0.1$ μm, $dx = 0.8W_0$, $L_z' = 8$ μm, and five phase fields. We can see that the grain size distributions converge well when we halve the mesh size or increase the domain height or the number of phase fields.

Appendix B. Kolmogorov–Smirnov test

Kolmogorov-Smirnov test can be used to compare two samples and test whether the two samples could have come from the same distribution. If sample A has m data points and sample B has n data points, the null hypothesis that two samples coming from the same distribution is rejected when:

$$\mathrm{KS} > \sqrt{-\ln\left(\frac{\alpha}{2}\right) \cdot \frac{1 + \frac{n}{m}}{2n}}, \tag{B.1}$$

where KS is the Kolmogorov-Smirnov statistic and α is the confidence level. For our setup, if a testing case has $n = 120$ grains and we think GrainGNN and phase field results are the same with a 95% confidence level, the critical KS value is $KS_c = \sqrt{-\ln\left(\frac{0.95}{2}\right)/120} = 0.079$. As discussed in Section 3.2, the KS values for in-distribution generalization are mostly in the range of 0.03−0.06, meaning that GrainGNN predicts the same grain size distribution as the phase field model. For out-of-distribution generalization we change the number of grains. KS_c is 0.043 for $n = 400$ and 0.022 for $n = 1600$. In Fig. 10h we vary the initial orientation distribution with $n = 400$. We can see $KS \approx KS_c$ for all θ_0, which means we have 95% confidence GrainGNN and phase field have the same grain size distributions. In Fig. 10g when we decrease initial grain sizes, $KS > KS_c$ for $d_0 < 3$ μm, which indicates a higher distribution mismatch for smaller grains.

Appendix C. Code and data availability

GrainGNN codes are available at https://github.com/YigongQin/GrainGraphNN. Codes contain how to create a graph using the Voronoi diagram and how to extract a graph from grain microstructure images. The neural networks are developed based on the Pytorch Geometric Library. Training and testing of the networks with comparisons against the phase field results are provided. The GitHub repository also provides the trained models including the regressor and the classifier. The phase field data used for training can be reproduced with the CUDA codes at https://github.com/YigongQin/cuPF.

References

[1] J. Smith, et al., Linking process, structure, property, and performance for metal-based additive manufacturing: computational approaches with experimental support, Comput. Mech. 57 (2016) 583–610.
[2] I. Steinbach, F. Pezzolla, A generalized field method for multiphase transformations using interface fields, Phys. D: Nonlinear Phenom. 134 (1999) 385–393.
[3] N. Ofori-Opoku, N. Provatas, A quantitative multi-phase field model of polycrystalline alloy solidification, Acta Mater. 58 (2010) 2155–2164.
[4] T. Pinomaa, M. Lindroos, M. Walbrühl, N. Provatas, A. Laukkanen, The significance of spatial length scales and solute segregation in strengthening rapid solidification microstructures of 316L stainless steel, Acta Mater. 184 (2020) 1–16.
[5] M. Yang, L. Wang, W. Yan, Phase-field modeling of grain evolutions in additive manufacturing from nucleation, growth, to coarsening, npj Comput. Mater. 7 (2021) 1–12.
[6] A.F. Chadwick, P.W. Voorhees, The development of grain structure during additive manufacturing, Acta Mater. 211 (2021) 116862.
[7] C.-A. Gandin, M. Rappaz, A 3D cellular automaton algorithm for the prediction of dendritic grain growth, Acta Mater. 45 (1997) 2187–2195.
[8] M. Rolchigo, R. Carson, J. Belak, Understanding uncertainty in microstructure evolution and constitutive properties in additive process modeling, Metals 12 (2022) 324.
[9] Matt Rolchigo, Samuel Temple Reeve, Benjamin Stump, Gerald L. Knapp, John Coleman, Alex Plotkowski, James Belak, ExaCA: a performance portable exascale cellular automata application for alloy solidification modeling, Comput. Mater. Sci. 214 (2022) 111692.
[10] T.M. Rodgers, J.D. Madison, V. Tikare, Simulation of metal additive manufacturing microstructures using kinetic Monte Carlo, Comput. Mater. Sci. 135 (2017) 78–89.
[11] E. Miyoshi, et al., Ultra-large-scale phase-field simulation study of ideal grain growth, npj Comput. Mater. 3 (2017) 1–6.
[12] E. Miyoshi, et al., Large-scale phase-field study of anisotropic grain growth: effects of misorientation-dependent grain boundary energy and mobility, Comput. Mater. Sci. 186 (2021) 109992.
[13] K. Chang, L.-Q. Chen, C.E. Krill III, N. Moelans, Effect of strong nonuniformity in grain boundary energy on 3-D grain growth behavior: a phase-field simulation study, Comput. Mater. Sci. 127 (2017) 67–77.
[14] J. Burke, D. Turnbull, Recrystallization and grain growth, Prog. Met. Phys. 3 (1952) 220–292.
[15] M. Hillert, On the theory of normal and abnormal grain growth, Acta Metall. 13 (1965) 227–238.
[16] E.A. Holm, G.N. Hassold, M.A. Miodownik, On misorientation distribution evolution during anisotropic grain growth, Acta Mater. 49 (2001) 2981–2991.
[17] Y. Qin, S. DeWitt, B. Radhakrishnan, G. Biros, GrainNN: a neighbor-aware long short-term memory network for predicting microstructure evolution during polycrystalline grain formation, Comput. Mater. Sci. 218 (2023) 111927.
[18] D. Zöllner, P.R. Rios, Topological changes in coarsening networks, Acta Mater. 130 (2017) 147–154.
[19] C. Torres, M. Emelianenko, D. Golovaty, D. Kinderlehrer, S. Ta'asan, Numerical analysis of the vertex models for simulating grain boundary networks, SIAM J. Appl. Math. 75 (2015) 762–786.
[20] R. Ohayon, C. Soize, Advanced Computational Vibroacoustics: Reduced-Order Models and Uncertainty Quantification, Cambridge University Press, 2014.
[21] M. Frangos, Y. Marzouk, K. Willcox, B. van Bloemen Waanders, Surrogate and reduced-order modeling: a comparison of approaches for large-scale statistical inverse problems, in: Large-Scale Inverse Problems and Quantification of Uncertainty, 2010, pp. 123–149.
[22] P. LeGresley, J. Alonso, Airfoil design optimization using reduced order models based on proper orthogonal decomposition, in: Fluids 2000 Conference and Exhibit, 2000, p. 2545.
[23] B.R. Noack, M. Morzynski, G. Tadmor, Reduced-Order Modelling for Flow Control, vol. 528, Springer Science & Business Media, 2011.
[24] X. Li, et al., A transfer learning approach for microstructure reconstruction and structure-property predictions, Sci. Rep. 8 (2018) 1–13.
[25] R. Bostanabad, Reconstruction of 3D microstructures from 2D images via transfer learning, Comput. Aided Des. 128 (2020) 102906.
[26] I. Goodfellow, et al., Generative adversarial nets, Adv. Neural Inf. Process. Syst. 27 (2014).
[27] Z. Yang, et al., Microstructural materials design via deep adversarial learning methodology, J. Mech. Des. 140 (2018).
[28] X.Y. Lee, et al., Fast inverse design of microstructures via generative invariance networks, Nat. Comput. Sci. 1 (2021) 229–238.
[29] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.
[30] D. Montes de Oca Zapiain, J.A. Stewart, R. Dingreville, Accelerating phase-field-based microstructure evolution predictions via surrogate models trained by machine learning methods, npj Comput. Mater. 7 (2021) 3.
[31] C. Hu, S. Martin, R. Dingreville, Accelerating phase-field predictions via recurrent neural networks learning the microstructure evolution in latent space, Comput. Methods Appl. Mech. Eng. 397 (2022) 115128.
[32] K. Yang, et al., Self-supervised learning and prediction of microstructure evolution with convolutional recurrent neural networks, Patterns 2 (2021) 100243.
[33] V. Oommen, K. Shukla, S. Goswami, R. Dingreville, G.E. Karniadakis, Learning two-phase microstructure evolution using neural operators and autoencoder architectures, npj Comput. Mater. 8 (2022) 190.
[34] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229.

[35] M. Syha, D. Weygand, A generalized vertex dynamics model for grain growth in three dimensions, Model. Simul. Mater. Sci. Eng. 18 (2009) 015010.
[36] K. Kawasaki, T. Nagai, K. Nakashima, Vertex models for two-dimensional grain growth, Philos. Mag. B 60 (1989) 399–421.
[37] F. Wakai, N. Enomoto, H. Ogawa, Three-dimensional microstructural evolution in ideal grain growth—general statistics, Acta Mater. 48 (2000) 1297–1311.
[38] T. Xue, Z. Gan, S. Liao, J. Cao, Physics-embedded graph network for accelerating phase-field simulation of microstructure evolution in additive manufacturing, npj Comput. Mater. 8 (2022) 201.
[39] M. Dai, M.F. Demirel, Y. Liang, J.-M. Hu, Graph neural networks for an accurate and interpretable prediction of the properties of polycrystalline materials, npj Comput. Mater. 7 (2021) 103.
[40] J.M. Hestroffer, M.-A. Charpagne, M.I. Latypov, I.J. Beyerlein, Graph neural networks for efficient learning of mechanical properties of polycrystals, Comput. Mater. Sci. 217 (2023) 111894.
[41] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint, arXiv:1609.02907, 2016.
[42] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (2008) 61–80.
[43] C. Morris, et al., Weisfeiler and Leman go neural: higher-order graph neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 4602–4609.
[44] O. Wieder, et al., A compact review of molecular property prediction with graph neural networks, Drug Discov. Today Technol. 37 (2020) 1–12.
[45] D.K. Duvenaud, et al., Convolutional networks on graphs for learning molecular fingerprints, Adv. Neural Inf. Process. Syst. 28 (2015).
[46] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, in: International Conference on Machine Learning, PMLR, 2017, pp. 1263–1272.
[47] T. Xie, J.C. Grossman, Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties, Phys. Rev. Lett. 120 (2018) 145301.
[48] J. Chen, X. Wang, X. Xu, GC-LSTM: graph convolution embedded LSTM for dynamic network link prediction, Appl. Intell. (2022) 1–16.
[49] A. Pareja, et al., Evolvegcn: evolving graph convolutional networks for dynamic graphs, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 5363–5370.
[50] P. Goyal, S.R. Chhetri, A. Canedo, dyngraph2vec: Capturing network dynamics using dynamic graph representation learning, Knowl.-Based Syst. 187 (2020) 104816.
[51] A. Vaswani, et al., Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.
[52] T. Pinomaa, et al., Process-structure-properties-performance modeling for selective laser melting, Metals 9 (2019) 1138.
[53] Y. Qin, Y. Bao, S. DeWitt, B. Radhakrishnan, G. Biros, Dendrite-resolved, full-melt-pool phase-field simulations to reveal non-steady-state effects and to test an approximate model, Comput. Mater. Sci. 207 (2022) 111262.
[54] D. Tourret, A. Karma, Growth competition of columnar dendritic grains: a phase-field study, Acta Mater. 82 (2015) 64–83.
[55] T. Takaki, et al., Competitive grain growth during directional solidification of a polycrystalline binary alloy: three-dimensional large-scale phase-field study, Materialia 1 (2018) 104–113.
[56] J. Bragard, A. Karma, Y.H. Lee, M. Plapp, Linking phase-field and atomistic simulations to model dendritic solidification in highly undercooled melts, Interface Sci. 10 (2002) 121–136.
[57] Y. Shi, et al., Masked label prediction: unified message passing model for semi-supervised classification, arXiv preprint, arXiv:2009.03509, 2020.
[58] S. Vedantam, B. Patnaik, Efficient numerical algorithm for multiphase field simulations, Phys. Rev. E 73 (2006) 016703.
[59] A. Badillo, C. Beckermann, Phase-field simulation of the columnar-to-equiaxed transition in alloy solidification, Acta Mater. 54 (2006) 2015–2026.
[60] T. Pinomaa, et al., Phase field modeling of rapid resolidification of Al-Cu thin films, J. Cryst. Growth 532 (2020).
[61] F. Xu, F. Xiong, M.-J. Li, Y. Lian, Three-dimensional numerical simulation of grain growth during selective laser melting of 316L stainless steel, Materials 15 (2022) 6800.
[62] Z. Zhang, Improved Adam optimizer for deep neural networks, in: 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), IEEE, 2018, pp. 1–2.
[63] T. Lookman, P.V. Balachandran, D. Xue, R. Yuan, Active learning in materials science with emphasis on adaptive sampling using uncertainties for targeted design, npj Comput. Mater. 5 (2019) 1–17.
[64] A. Krogh, J. Vedelsby, Neural network ensembles, cross validation, and active learning, Adv. Neural Inf. Process. Syst. 7 (1994).