Using Graph Neural Networks to Approximate Mechanical Response on 3D Lattice Structures

Elissa Ross*, [1], Daniel Hambleton[1]

[1] MESH Consultants Inc.
401 Wellington St. W. 3rd Floor, Toronto ON M5V 1E7

* Corresponding author e-mail: elissa@meshconsultants.ca

Abstract

The promise of computer aided manufacturing (CAM) is to make materializable structures that could not be fabricated using traditional methods. An example is 3D lattices, which may be arbitrarily complex. Variation in the lattice geometry and print media can define a vast spectrum of resulting material behaviour, ranging from fully flexible forms to completely stiff examples with high strength. Panetta et al. (2015) outline a methodology to generate lattice geometries with specified material properties. However, their method relies heavily on finite element analysis of beam models to determine the material properties of a discretely sampled space of lattices. In the present study, we use machine learning to perform a stiffness analysis on highly symmetric lattice geometries with periodic boundary conditions. We train a graph convolutional network on a dataset of lattices sampled continuously from the space of all lattices, then use the trained model to predict deflections for previously unseen lattices. With this approach we are able to approximate the material behaviour of the vast space of all lattice geometries, which offers potential for real-time material feedback at the design stage. It also offers a method to explore a space of building components that are materially sparse yet offer high strength and stiffness. The symmetry of the lattice geometry together with the stiffness analysis creates a homogenized material model, which can be applied to different designs to obtain similar material performance. We illustrate the approach with several examples across different scales.

Keywords: Lattice structures, machine learning, graph neural networks, material properties, performance-aware design.

1 Introduction

Computer-aided manufacturing (CAM) is fundamental to the design and fabrication of bridges, high-rise towers, medical protheses, and much more. From automating repetitive tasks at an incredible rate to unlocking the potential of unusual shapes in structural optimization, the partnership of computer, designer and builder has been fruitful. Yet some things have not changed. Variation and complexity of design intent is still tempered by cost and the need to standardize parts and processes. There is therefore great benefit to understanding how repetitive components can be deployed with as much design freedom and variation as possible.

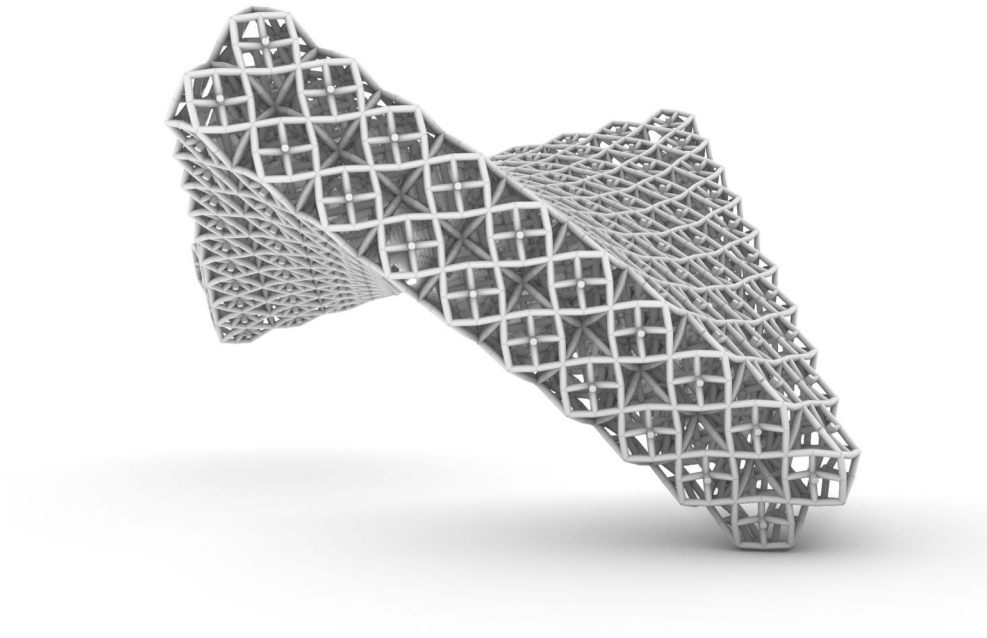

Figure 1: An example geometry composed of lattice elements.

Perhaps the most prototypical type of repetitive structures from a geometric point of view is the lattice (**fig. 1**). Lattices are highly symmetric structures that can be joined together without gaps, overlaps, or distortion. This complex, but orderly, arrangement of nodes and beams is present in a huge range of natural and human-made structures from bathroom tiles to the molecular structure of crystals. From a design perspective, this orderliness is often their undoing. The underlying mathematical rules of a lattice do not readily allow for the variation that will arise during the transition from abstract form to physical object. When a lattice is distorted the features that make it an attractive candidate for a structural system,

such as regularity of node geometry, will change and become difficult to predict. Managing the impact of this change in a design setting can be challenging (Tam et al. 2018).

Our project uses pre-computed performance metrics of lattices and their variations to train a graph neural network (GNN) machine learning model. This model is then used to predict the relative performance of lattice elements encountered in aggregate structures that exhibit considerable geometric variation. The goal is to help designers and engineers to make performance driven decisions by providing real time feedback of high-quality simulation results.

1.1 Outline

In Section 2 we outline background information on lattice geometries and graph neural networks. We also mention some other machine learning projects that attempt to learn from geometric inputs. In Section 3 we describe the methods of the present investigation, specifically the four synthetic lattice datasets we created and their compression simulation. We present the graph neural network architecture we employed and discuss a variety of data representations for geometric lattices as inputs to the graph neural network. Section 4 reports on the results of the machine learning models. In Section 5 we present the findings in an applied context by discussing several design applications of the trained machine learning model.

2 Background

2.1 Lattice geometries

As material structures we can imagine lattices as structural systems composed of repeating units of struts and nodes (Tam et al. 2018). In this work we focus on *cubic lattices*, which are lattices that possess the full symmetries of the cube and are therefore invariant under the 48 rotational and reflectional symmetries comprising the symmetry group of the cube (Conway et al. 2008). Such a lattice can be defined by a *unit cell*: a cube containing a lattice 'recipe' that can be packed together to generate arbitrarily large 3D lattice structures. In fact, we can describe an even more concise 'recipe' for such a lattice, by dividing the cube into 48 equal-sized tetrahedra, each invariant under any symmetry preserving the cube (see **fig. 2a**).

Following the description in Panetta et al. (2015), we construct a lattice by defining a *lattice pattern* in a single tetrahedron. The lattice pattern consists of *lattice nodes* and *edges* connecting those nodes. The nodes may be places on the vertices, edges or faces of the tetrahedron, or within its volume (**fig. 2b**), with 0,1,2 and 3 degrees

of freedom respectively. We therefore have 4 vertex nodes, 6 edge nodes, 4 face nodes and 1 tetrahedron centre node, for a total of 15 possible node placements. We describe the set of 15 possible nodes with the following notation:

$$\{v_0, v_1, v_2, v_3, e_0, e_1, e_2, e_3, e_4, e_5, f_0, f_1, f_2, f_3, t_0\}.$$

In this way the description of a lattice involves both topological[1] and geometric characteristics. The topology of a lattice defines the *graph* of the lattice; specifically, what nodes and edges are present (e.g. the node e_3 is connected to the node f_2 by the edge (e_3, f_2)). The topology of a lattice pattern is completely defined by its edge list, for example $\{(e_0, f_1), (e_0, e_3), (v_2, f_1)\}$.

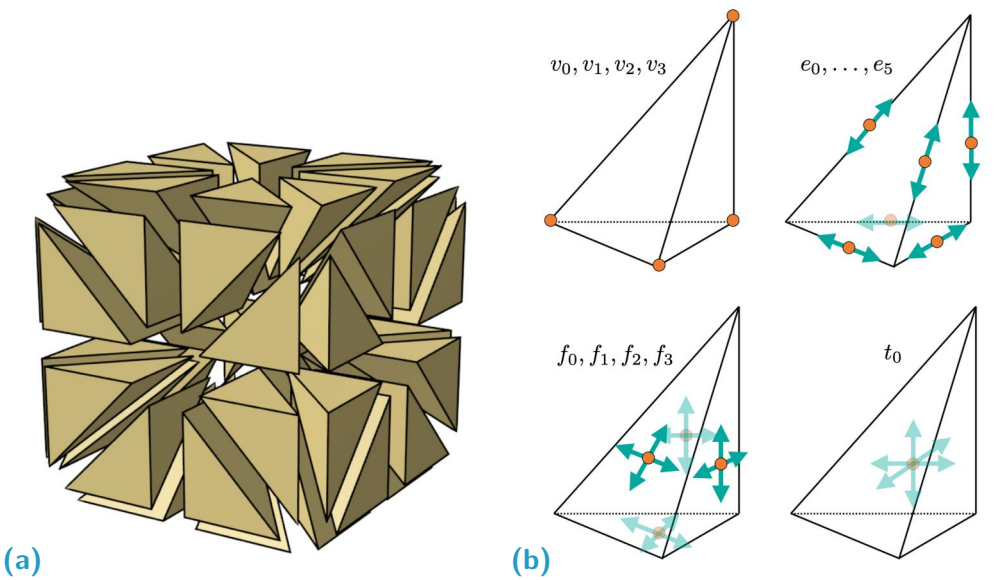

Figure 2: (a) A cube broken into 48 equal tetrahedra (Panetta et al. 2015); (b) the 15 possible node positions on the lattice pattern tetrahedron, with indicated degrees of freedom.

The geometry of a lattice describes the position of the lattice pattern nodes, which we denote $pos(n)$ for node n. We use barycentric coordinates to define the position of nodes on a lattice pattern relative to the four corners (vertices) of the lattice pattern tetrahedron. We call these relative coordinates the *offsets* as in Panetta et al. (2015). For example, the node e_0 is on the tetrahedral edge connecting v_0 and v_1. If e_0 has offset '0.2', its position is described by $pos(e_0) = 0.2pos(v_0) + 0.8pos(v_1)$ (**fig. 3**).

[1] It may be mathematically more precise to describe these qualities as *combinatorial*; we use the term 'topological' to maintain consistency with the architectural geometry literature.

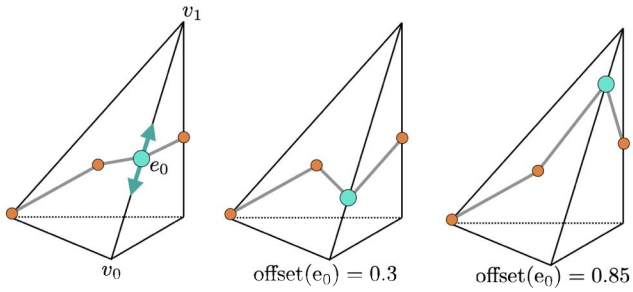

Figure 3: The geometry of a single topological lattice type is controlled by offsetting the nodes according to their degrees of freedom. In this example, the edge node is pictured sliding along its single degree of freedom. The face node in this image has two degrees of freedom (not pictured), and the nodes at the vertices are fixed.

When the lattice pattern is used to populate each of the 48 tetrahedra comprising the unit cube, the result is a cubic lattice. See **fig. 4** which illustrates this process in a 2D analogue, using a triangle as the lattice pattern for the construction of a square lattice. Note that this method does not guarantee that the resulting lattice will be connected, but we can ensure connectivity through an attention to the nodes on the boundaries of the lattice patterns, and the unit cell. In a graph, the *valence* of a node is the number of incident edges. For lattices note that the valence of a node in the lattice pattern is likely different from the valence of a node in a unit cell, because of the effect of the rotational and reflectional symmetries of the cube. This may again be different from the valence of a node in a periodic lattice because of the effect of the translational periodic symmetry.

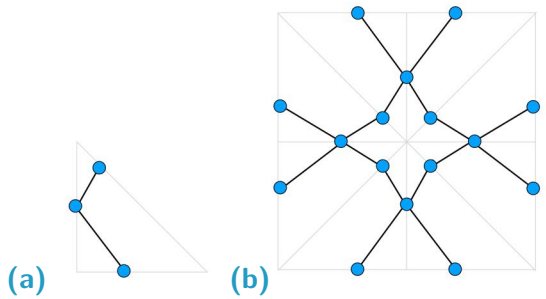

Figure 4: Illustration of the lattice creation through a 2D analogue: (a) The lattice pattern is a triangle populated with three nodes and two edges; (b) after applying the symmetries of the square, we obtain a 2D lattice.

2.2 Graph neural networks

A *neural network* is a network of (typically simple) computational units combined with simple nonlinear functions that together can approximate arbitrarily complex nonlinear functions on a given dataset (see e.g. Goodfellow et al. 2016). *Deep learning* is concerned with neural networks with multiple computational layers, typically operating on large datasets. A *convolutional neural network* (CNN) is

a special neural network in which the processing units are convolutional filters (kernels) which are applied to the data to detect local features. CNNs have been used with great success in areas such as image recognition.

In recent years there has been an interest in building on the success of neural networks to apply to a wider range of input data types. *Geometric deep learning* is an attempt to generalize deep neural networks (convolutional networks in particular) to non-Euclidean domains, such as graphs, point clouds or 3D meshes (Wu et al. 2020). This is challenging because graph-structured data is often heterogeneous (graphs may have different numbers of nodes and edges), and the convolution operator does not generalize to these non-Euclidean settings, but must instead be redefined (Bronstein et al. 2017). These *graph neural networks* (GNNs) are usually defined by either a spatial or spectral approach to convolution (Coa et al. 2020). Typical applications include predicting links in social networks, point cloud reconstruction, or drug discovery. Most relevant to the present application are studies in molecular chemistry that attempt to learn properties of molecules from a description (and in some cases the position) of their atomic structure, as in Gilmer et al. (2017).

Since the goal of our project is to predict the stiffness of a lattice, we work with a graph regression model. Following Gilmer et al. (2017) we can conceptualize graph neural networks as "message-passing" networks. Each node has features ("messages"), which are then aggregated ("passed") according to the nodes in the 1-ring neighbourhood of a particular node (those immediately connected by an edge). The k-th layer of a GNN will aggregate features from nodes that are k-hops away. In the case of graph classification or regression, a pooling operation will convert the aggregated messages on the nodes into a result (a labelling or regression target) on the whole graph. In this way, the differences between different GNNs for graph regression are primarily based on the choice of aggregation method and the choice of pooling operation.

2.3 Related work

In Panetta et al. (2015), the authors introduce a set of tileable cubic lattices that can be 3D-printed to achieve a range of compressive material properties. The project systematically generates a large sample of 3D-printable potential lattice types, then selects a small group of these lattices to optimally represent a range of compressive behaviours. The six lattices in this small group may be tiled together to obtain larger structures with desired emergent material behaviour. The focus of their project is on small structures that are 3D printed, and the authors describe a design tool to achieve variation in the material properties within a design.

Our work is highly influenced by Panetta et al. (2015), and both projects are concerned with the homogenized material properties of isotropic (cubic) lattices. While Panetta et al. focused on creating a 'kit' of lattices which would exhibit a range of material behaviour, we are interested in analyzing the compressive behaviour of arbitrary lattices. In addition, the authors of that paper sample the space of all lattices discretely (e.g. node positions are constrained to lie on one of predetermined 5 locations along the tetrahedron edge), while we sample the offset space continuously. We also do not restrict our study to 3D printable lattices. While we use a simulation similar to finite element analysis to generate our datasets, our goal is to then use machine learning to make similar predictions on previously unseen lattices. This is not part of the goal of Panetta et al. (2015).

Although interest in machine learning, and deep learning in particular has seen an explosion of interest in the last two decades, the application of machine learning to geometry has been slower to gain momentum. What are highly successful methods in two dimensions become computationally burdensome in three dimensions. For example, a naïve approach to this generalization would be to represent 3D space by voxels in the same way that 2D space is represented by pixels, in which case the machinery of convolutional neural networks then applies directly (Bronstein et al. 2017). Although these methods have achieved some successes (Xu et al. 2017), the challenge of computation necessitates a low resolution that is likely inappropriate for lattice structures. The closest related work therefore comes from Graph Neural Networks, and models of molecules. However, even here most graph models of molecules do not use *conformation* (position of atoms in a molecule) as input because molecular geometry is not always known and may be challenging to collect (in fact there are some promising attempts to *predict* conformation from molecular properties (Mansimov et al. 2019)).

A project that more closely related to the goals of the present work is in Liang et al. (2017). The authors use deep learning as a surrogate for finite-element analysis to study the biomechanics of human tissues, in particular to predict the stress distributions on patient aorta geometries. To our knowledge, this is the first example of a neural network being used as a proxy for finite-element analysis, which is also the ultimate goal of our project. In contrast to our work, however, the geometry of the aorta is modeled by a uniform 3D mesh that is fit to a range of patient geometries. In other words, the authors *homogenize* the input geometry, and then encode the shape of the aorta using principal component analysis. A neural network (not graph-based) is then applied to the encoded shape to obtain the stress information. While the nature of the input geometry of this project is very different from ours, the accuracy achieved by Liang et al. provides a promising

argument that it is possible to learn the nonlinear relationship between shape and stress using deep learning.

More broadly, our work fits into a class of problems in architecture and engineering that aims to use data-driven methods as a surrogate for more time and resource-intensive structural simulation, particularly in the early design stages which often lack structural information (Tseranidis et al. 2016). For example, neural network models were applied in Danhaive and Mueller (2018) in an examination of the deformation of shell structures under self-weight, leading to the availability of (almost) real-time structural information. A potential application of that work (and our own), would be to embed some structural feedback into the design phase of a project.

3 Methods

3.1 Dataset creation

The space of potential lattices is infinite when sampled continuously. We therefore constrained our focus to lattices that are connected, have nodes with valence between 2 and 16 (after incorporating periodic symmetries), and have at most 3 or 4 nodes per lattice pattern tetrahedron. This restriction results in approximately six thousand different lattice topologies.

From this collection of lattices, we created several different datasets:

1. The *All Lattices* dataset consists of four different randomly generated offset positions for the lattice pattern nodes, for each of the 6K different topological lattice types (top row, **fig. 5**), resulting in approximately 24K lattice instances.

2. The *One Type A* dataset consists of a single topological lattice type given by the edges $\{(e_0, f_1), (e_0, e_3), (v_2, f_1)\}$, with 25K different offset positions for the nodes. The topological type was selected for large number of degrees of freedom of the nodes, which results in a wide range in the geometric expression of the lattice (middle row, **fig. 5**).

3. The *One Type B* dataset consists of the single topological type defined by the edges $\{(e_0, f_2), (e_0, e_3), (v_2, f_2)\}$, with 10K different offset positions for the nodes.

4. The *One Type A Morphed* dataset consists of 25K morphed versions of the 'One Type A' dataset (bottom row, **fig. 5**). The morphs were defined by a box morph of the cube enclosing the lattice unit cell. Each morph is randomly determined.

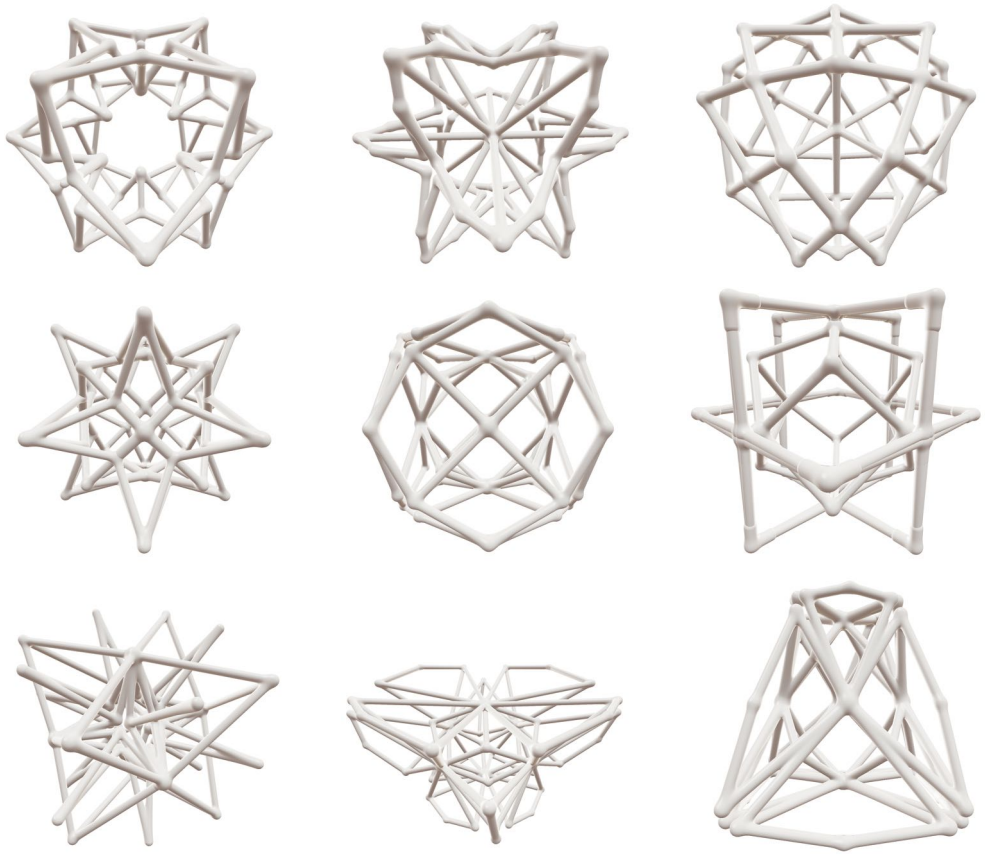

Figure 5: Lattice datasets. Top Row: Samples from the 'All Lattices' dataset; Middle Row: Samples from the 'One Type A' dataset (note the variation in geometry despite a single lattice topology); Bottom Row: Samples from the 'Morphed' dataset.

3.2 Compression simulation

The goal of the compression simulation is to measure to what degree different periodic lattice geometries resist a compressive force. This resistance is captured by taking the difference between the top-most node in the compressed lattice and the height of the bounding box of the uncompressed lattice. We call this number the *max compression*. Given that our focus in this project is on architectural scale objects, we define our units and boundary conditions as follows:

Units. We use the Standard Units (kg, m, s) convention. Each lattice unit cell is defined on the interior of a 1m x 1m x 1m cube.

Cross Section. Each beam in the lattice is assigned a solid circular cross section with radius of 0.05m.

Material. We use a Young's modulus of 215 GPA and Shear modulus of 80. This is roughly equivalent to a stainless steel material model.

Load. Each lattice unit cell is loaded on the positive Z plane ($+Z$), with a distributed load of 500 MPa in the negative Z direction. We use the notation $+Z$ to describe the halfspace bounded by the plane with positive unit Z vector as its normal. The planes $+X, -X, +Y, -Y, +Z, -Z$ therefore describe the faces of the bounding box of the unit cube.

Boundary Conditions. There are three boundary conditions:

1. Each node in the $-Z$ plane is allowed translation in the XY directions. Translation in Z and all rotations are clamped.
2. Nodes on the $+X$ and $-X$ planes are linked by a periodic transformation. Similarly, nodes on the $+Y$ and $-Y$ planes are linked. This creates a periodic boundary condition in the X and Y directions.
3. Finally, every node in the system is allowed translation but not rotation. This creates a fixed joint, much like a welded steel connection.

Solver. To simulate this scenario we use Kangaroo, a plugin for Rhinoceros3D and Grasshopper. We make use of the *6DOF* elements which can provide accurate results to the type of loads and elements in our experiment. Other solvers could be used to obtain the features required to train the neural net. Recognizing that the output of Kangaroo does not match a full finite element analysis, our goal with this project was to create a "black box" scenario which can take as input the lattice unit cells and produce as output some measure of performance. This black box could later be populated with higher fidelity FEA, or another performance measure.

With our simulation defined and the solver in place, we processed a total of 100K lattices. Along with the maximum compression, we recorded the maximum displacement vector and the fully deflected shape.

3.3 Machine learning model

We use the *PyTorch-Geometric* (Fey and Lenssen 2019) extension library for *PyTorch* (Paszke et al. 2019) to train the neural network. We considered two main models for graph-level regression, specifically the Graph Isomorphism Network (GIN) model (Xu et al. 2019), and the "neural message-passing" model (NNConv) (Gilmer et al. 2017). We found the NNConv model to be more substantially more performant, likely due to the fact that the GIN model does not incorporate edge features. In the results described below, we use the neural message-passing model, with the following architecture:

1. 5 iterations of {NNConv operator (dimension = 25), Gated Recurrent Unit}.
2. Graph level pooling operation by addition.
3. 2 fully connected layers, with Rectified Linear Units.

We experimented with different pooling operations to obtain a scalar result for the whole graph, such as *set2set*, global addition (sum of the signal on the nodes) and global mean pooling (mean of the signal on the nodes), with modest variations in success. For consistency we use global addition in the results reported below. The learning rate was initialized to 0.01 and scheduled to reduce upon training loss plateau. We also experimented with depths (the number of iterations in step 1), with the best results in the 3-5 layer range.

The *NNConv* operator uses both node and edge features, which we use to encode the geometric information of the lattice, which we now describe. The general pipeline is illustrated in **fig. 6**.

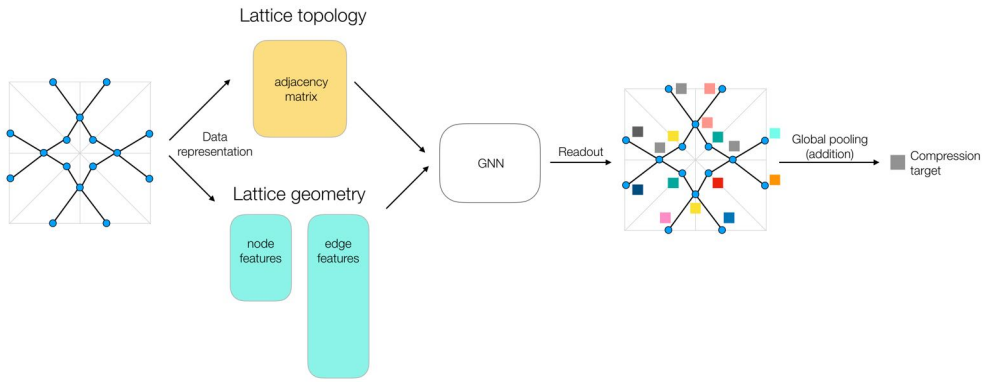

Figure 6: Schematic diagram illustrating the lattice GNN pipeline.

3.4 Data representation

The way that the lattice data is captured for input to the GNN involves a number of choices about how to record both the topological and geometric characteristics of the lattice, which may directly impact the model performance.

Topological aspects. Cubic lattices admit a highly concise description in the form of the (tetrahedral) lattice pattern. However, to make use of the "message-passing" capabilities of the GNN, we require a larger graph (the lattice patterns in our datasets have only 2 or 3 edges). For this reason, we choose to use the lattice unit cell as the basic unit of data.

We investigate two models to capture the topology of lattice unit cells. The first is simply the graph of the unit cell, and we call this the *unit cell representation*. A potential downside of this representation is that it does not capture the periodicity of the lattice, although we can compute the adjusted valence of boundary nodes by considering the effect of periodic symmetry on the lattice connectivity. The second model identifies boundary vertices that are invariant with respect to the periodic symmetry. We call this the *merged boundary representation* (see **fig. 7** for a 2D analogue). Identifying boundary vertices in this way allows us to naturally capture the periodic structure of the lattice, and also reduces the size of the lattice graph. In both representations, the topological data is passed to the GNN using an *adjacency matrix*, the (0,1)-matrix capturing the edges of the graph.

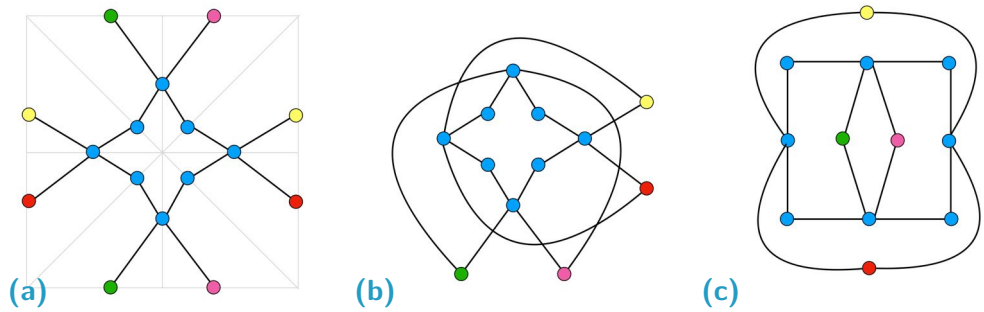

Figure 7: (a) An example 2D lattice, with boundary vertices coloured according to their identification under the periodic symmetry; (b) The merged boundary representation, with boundary vertices identified; (c) Another embedding of the same graph, emphasizing that this is a topological, not geometric, representation of the lattice structure.

Geometric aspects. A challenge of machine learning models on geometric data is that the data must be invariant to transformations such as rotation, translation and dilation. Just as we would hope an image recognition model would identify a cat no matter where the cat appears within the frame of the image, we require the data representation of geometric objects like lattices to be independent of their embedding. For this reason, we focus on the *geometric features* of a lattice as a proxy of its position in space. We use *node features* and *edge features* as follows:

1. The complete set of node features we considered are: valence of the periodic lattice nodes ($v \in [2,16]$); node type ($nt \in [0,14]$), e.g. v_0, e_3, f_2 etc.; average edge length of incident edges (ℓ_a); total length of incident edges (ℓ_t); average over the incident edges of the absolute value of the dot product of the edge direction vector with $(0,0,1)$ (denoted z); and offsets (o_0, o_1, o_2). We focus on the z-direction as this is the direction in which the force is applied. For simplicity in this report we describe two distinct feature sets:

(a) *Offsets*: Valence,node type, and offset information: (v, nt, o_0, o_1, o_2), or

(b) *Geometric Features*: Metrics calculated from the lattice information on a per-node basis: (v, ℓ_a, ℓ_t, z)

2. Edge features consisting of: edge length (ℓ_e) and the dot product of the edge direction vector with each of the three unit direction vectors: (ℓ_e, x, y, z).

4 Results

We use mean absolute error (MAE) as our primary metric of success. For each model we also include the MAE for the dummy regressor which always predicts the mean and is independent of the choice of data representation. The MAE of the GNN model is reported after 200 training epochs. We did not observe significant further reduction in training error after this number of epochs. In addition, since the training relies on some nondeterministic processes, the reported MAE is the mean of three or more training cycles. The results are presented in **tab. 1** to **4**. Here we use "GNN" to describe the model outlined in the previous section, specifically the GNN model from Gilmer et al. (2017), as implemented in *PyTorch-Geometric* (Fey and Lenssen 2019).

For the single lattice datasets A and B, we compare the results of the GNN model to the results of "classical" machine learning algorithms, specifically a multi-layer perceptron (MLP) regression in scikit-learn (Pedregosa et al. 2011). We are able to do this due to the homogenous nature of the lattice data in this case. That is, since we are constraining our interest to a single lattice type, the data lacks the topological variability of the "All Lattices" dataset. We also consider reductions to the size of the training datasets, and look at different combinations of data representations. Further comments on the relative merits of these approaches are provided in the next section.

We note that the error of the Type B Lattice dataset is lower than that of the Type A dataset, however it must be compared against the dummy regressor which is also lower (and reflects less variance in the sample data compression targets). Both cases exhibit at least a six-fold increase in accuracy over the dummy regressor.

In the case of the morphed dataset it is no longer possible to use the merged boundary representation of the lattices because the morphed representations may not be periodically symmetric. We therefore use the unit cell model. It is also not possible to use the offsets as node features. While the offsets determine the starting positions of the lattice nodes, the morphs modify their final positions.

Data Representation	ML Model	MAE
any	Dummy regressor	0.11526
Merged boundary & geometric features	GNN	0.08084
Merged boundary & offsets	GNN	0.06955

Table 1: All Lattices dataset.

Data Representation	ML Model	MAE
any	Dummy regressor	0.07509
geometric features only	Classical MLP	0.03034
Merged boundary & geometric features	GNN	0.01472
Reduction to 10K lattices	GNN	0.01488
Reduction to 2K lattices	GNN	0.01954
Merged boundary & offsets (25K lattices)	GNN	**0.00922**
Unit Cell & geometric features	GNN	0.01327

Table 2: One Type Lattice A dataset (25K lattices).

Data Representation	ML Model	MAE
any	Dummy regressor	0.03314
geometric features only	Classical MLP	0.01699
Merged boundary & geometric features	GNN	0.00526
Merged boundary & offsets	Classical MLP	0.01474
Merged boundary & offsets	GNN	**0.00441**

Table 3: One Type Lattice B dataset (10K lattices).

Data Representation	ML Model	MAE
any	Dummy regressor	0.08184
Unit Cell & geometric features	Classical MLP	0.02435
Unit Cell & geometric features	GNN	**0.01521**

Table 4: One Type Lattice A Morphed dataset (25K lattices).

Dataset	Data Representation	MAPE	Accuracy
One Type A	Merged boundary & offsets	7.58%	92.42%
One Type B	Merged boundary & offsets	7.18%	92.82%
One Type A Morphed	Unit Cell & geometric features	13.71%	86.29%

Table 5: MAPE and accuracy for best performing models.

Table 5 presents the mean absolute percentage error (MAPE) and accuracy (as 100% – MAPE) for the best performing models above. These metrics are the mean of three or more training iterations.

5 Discussion and design applications

As the results of the previous section indicate, the GNN model in its various realizations was able to learn a measure of the compressibility of a lattice from the lattice geometry. Although the accuracy of the learning was limited for the 'All Lattices' dataset, it is clear from the single lattice studies that much higher levels of accuracy are possible in more restricted settings.

A perhaps surprising finding is that the accuracy for the single lattice models is only slightly reduced when the size of the training data set is drastically reduced (from 25K to 10K or even 2K). This offers a potential pathway to training a heterogeneous data set like the 'All Lattices' set, simply by more densely sampling the offset space for each topological lattice. While the 'All Lattices' set had only four random samples per lattice topology, we could imagine an enlarged dataset with 2K random samples per topology. Even this enlarged dataset would not be particularly large by the standards of contemporary deep learning (Goodfellow et al. 2016), and is a planned topic for further investigation.

While it is not always possible to use the offset information as node features (e.g. in the morphed lattice dataset), it is notable that when we are able to use offset information, the model performs better. This is significant because it opens the door to the inverse model: setting a compressibility target then driving the network backward to generate lattice suggestions. In the case of geometric features, these do not give a recipe for how to generate a lattice, they are merely properties *of* a lattice. However, a model that outputs offsets would be capable of specifying lattice positions.

5.1 Significance

To our knowledge, this is the first instance of material properties being predicted from lattice geometry using machine learning. The potential of this finding is that we may be able to offer real-time structural feedback to designers without the cost or time of finite element analysis. While a simple compression study in Kangaroo may take upwards of six hours for 25 thousand lattices, for a trained machine learning model the results are available in seconds. In the specific case of the results presented here, we acknowledge the limitations of the simulation model (Kangaroo), which are not intended to replace the output of finite element analysis.

Yet, as a relative measure of compressibility we think this is a positive step in the direction of performance-aware design using machine learning. We now outline an example use case.

5.2 Design application: morphed geometries

This example consists of a volume populated with a single lattice geometry that is morphed to fit within (or equivalently a finite section of lattice is morphed to fit a surface, see **fig. 8** for a spherical example, and see **fig. 9** for an example on a hyperbolic paraboloid). As the design volume is pushed/pulled, the trained ML model offers predictions of relative compressibility on a *per-cell* basis. This is encoded visually by a colour scheme indicating where lattice units are more or less stiff than the undeformed case. In this way, a designer is able to understand the relative structural implications of various transformations of the lattice geometries. While the ML model does not make predictions about the aggregate lattice geometry, it is helpful to understand areas of relative strength and weakness within that volume.

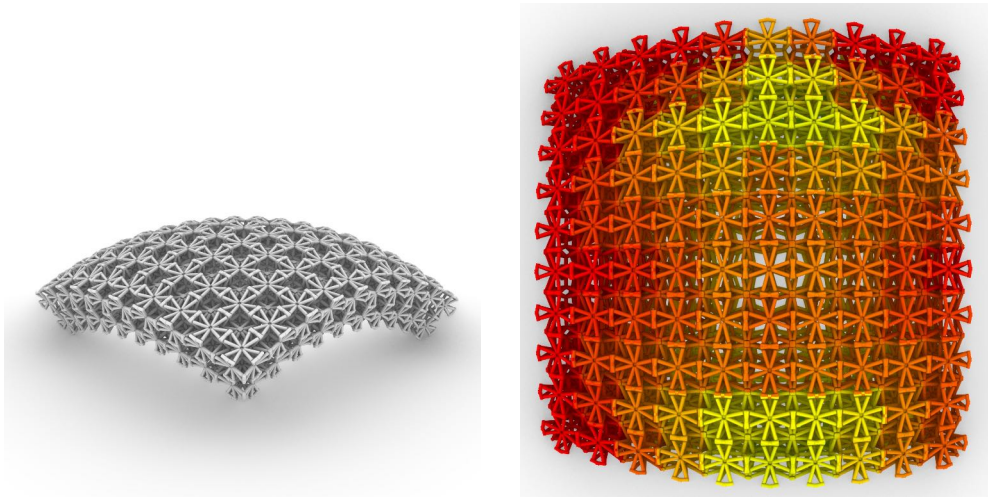

Figure 8: Left: Lattice A fit to the surface of a sphere (19,200 beams). Right: The colours represent the ratio of the predicted compression to the baseline undeformed simulation compression (predicted / actual). Yellow is 1.0 (the same compression), green is 0.5X compression and red is 1.5X compression. For this lattice the baseline (undeformed) compression is 0.025, the average in the morphed geometry is 0.038, with 0.013 for the least compressed cell, and 0.090 for the most compressed cell.

6 Conclusion & further work

In this paper, we present a novel end-to-end pipeline that predicts material properties of lattice structures using machine learning. Our tests show that the neural network defined in **sec. 4** can predict the compression on unseen lattices to an accuracy of 93%. Moreover, we can use the trained model to rapidly assess the relative per-

formance of a lattice unit cell that has been deployed on a design volume. While it is clear that using a trained model as a surrogate for detailed finite element analysis is faster than running that analysis for each design iteration, it is worth noting that the trained model also outperforms a lookup table or interpolation approach. In either case, the number of samples needed would be on the order of 10K per offset lattice instance (a single topological lattice in a single offset position). Apart from the large amount of data required, there is also the difficulty of defining an appropriate distance metric on the space of lattices or lattice features. These high dimensional spaces are notoriously challenging for nearest neighbour queries.

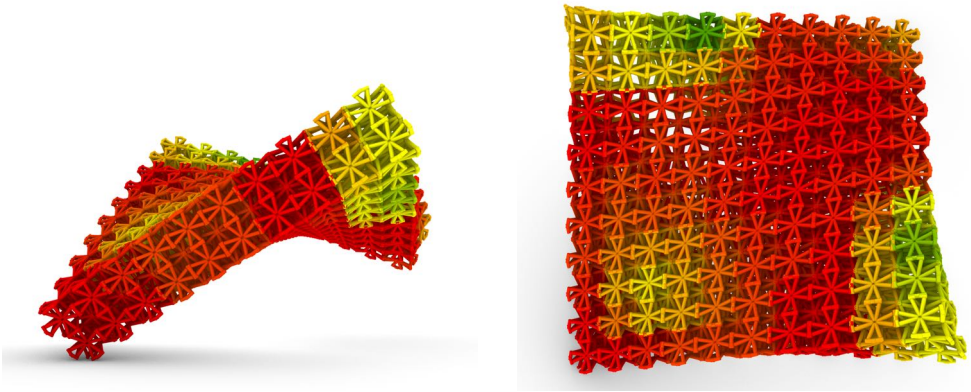

Figure 9: Lattice fit to the surface of a hyperbolic Paraboloid. See also **fig. 1**. Colour coding is as in **fig. 8**. The baseline compression is 0.025, the average in this geometry is 0.041, with the least compressed cell 0.010 and the most compressed cell 0.067.

A key part of our project is that it can incorporate data from any analysis defined on the lattice data sets. The same pipeline would work with a fine-grained volumetric analysis or different structural cross-sections run with more standard industry solvers. This means that rapid design iteration can be done without reducing the fidelity of the analysis on the training set. We believe that this approach has great potential for performance-aware design tools.

As mentioned in **sec. 5**, the next step in this work is to explore the inverse regression problem. The model would take as input a compression target and a target domain (i.e. a hexahedral element) and produce a lattice (or selection of lattices) as output. The result would be fast iteration of visually diverse, performance-driven lattice structures. We also intend to run the simulation with different FEA solvers and test the model using different features. A longer term goal is to identify some of the top performing lattices across a variety of metrics in the space of cubic lattices. Understanding this set and building a suite of finely tuned machine learning models to deploy them on industrial projects would be a powerful application of this research program.

Acknowledgements

The authors thank Nicholas Hoban, David Reeves and Tomasz Reslinski for fruitful discussions on this topic. We also thank the anonymous reviewers for drawing our attention to some additional references.

References

Bronstein, M., J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine 34*(4).

Coa, W., Z. Yan, Z. He, and Z. He (2020). A comprehensive survey on geometric deep learning. *IEEE Access 8*, 35929–35949.

Conway, J. H., H. Burgiel, and C. Goodman-Strauss (2008). *The Symmetries of Things*. Boca Raton: CRC Press.

Danhaive, R. and C. Mueller (2018). Structural metamodelling of shells. *Proceedings of IASS Annual Symposia 2018*(25), 1–4.

Fey, M. and J. E. Lenssen (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl (2017). Neural message passing for Quantum chemistry. In *ICML'17: Proceedings of the 34th International Conference on Machine Learning*, Volume 70, pp. 1263–1272.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. Cambridge: Massachusetts Institute of Technology.

Liang, L., M. Liu, C. Martin, and W. Sun (2017). A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. *J. R. Soc. Interface 15.*

Mansimov, E., O. Mahmood, S. Kang, and K. Cho (2019). Molecular geometry prediction using a deep generative graph neural network. *Scientific Reports 9*.

Panetta, J., Q. Zhou, L. Malomo, N. Pietroni, P. Cignoni, and D. Zorin (2015). Elastic textures for additive fabrication. *ACM Transactions on Graphics 34*(4).

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *In Advances in Neural Information Processing Systems 32*, 8024–8035.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research 12*, 2825–2830.

Tam, K.-M. M., D. J. Marshall, M. Gu, J. Kim, Y. Huang, J. Lavallee, and C. T. Mueller (2018). Fabrication-aware structural optimisation of lattice additive-manufactured with robot-arm. *International Journal of Rapid Manufacturing 7*.

Tseranidis, S., N. Brown, and C. T. Mueller (2016). Data-driven approximation algorithms for rapid performance evaluation and optimization of civil structures. *Automation in Construction 72*, 279–293.

Wu, Z., S. Pan, F. Chen, G. Long, C. Zhang, and P. Yu (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*.

Xu, K., W. Hu, J. Leskovec, and S. Jegelka (2019). How Powerful are Graph Neural Networks? In *ICLR*.

Xu, K., V. Kim, Q. Huang, and E. Kalogerakis (2017). Data-driven shape analysis and processing. *Computer Graphics Forum 36*(1).