UNIVERSITY OF
CAMBRIDGE

Symbolic Distillation of Graph Neural Networks

Liz Tan

Downing College

This report is submitted on June 2025 for the MPhil in Data Intensive Science

Word count: 6884

Abstract

We investigate the symbolic distillation of force laws from graph neural networks (GNNs) trained on simulated n-body datasets, reproducing the framework introduced by Cranmer et al. (2020) [1]. By leveraging the inductive biases inherent in GNN architectures and enforcing sparsity through various model variations we demonstrate that the internal edge messages can recover physically meaningful representations. Applying symbolic regression to these learned messages, we successfully reconstruct analytical force equations across multiple force types. Our results differ to that of the original paper - specifically the standard model variation (that contains no dimensionality constraint or added regularisation to encourage sparse representation) performed significantly better at recovering force laws in our project than in the original paper. We also identified an important data handling step - where outliers were removed before fitting the true forces to the GNN internal representation - that was left out in the original paper. This step proved crucial in obtaining similar results to the original work. Lastly, we extend the original work through the introduction of a new model variation, the pruning model.

List of Figures

List of Tables

Contents

1 Introduction

Historically, empirical physical laws were found by eye. Scientists observed natural phenomena, recorded measurements, and manually identified consistent patterns or relationships between variables. These patterns were distilled into mathematical expressions — such as Kepler's laws of planetary motion [2] or Ohm's law [3] -before a theoretical framework existed to explain them. The process relied heavily on human intuition, trial and error, and visual inspection of graphs and tables. Today, with the rise of high-dimensional data and complex systems, this manual approach becomes infeasible.

Deep learning models have become powerful tools for uncovering patterns in large datasets. However, they often operate as black boxes, mapping inputs to outputs through high-dimensional latent representations that are difficult to interpret in terms of underlying physical principles.

Conversely, symbolic regression offers an interpretable alternative by searching for analytical expressions that accurately model data using a predefined set of mathematical operations. Unlike deep learning, symbolic regression produces closed-form equations that are compact, human-readable, and often align with physical intuition. However, traditional symbolic regression methods struggle to scale to high-dimensional datasets due to their combinatorial search space.

The paper *Discovering Symbolic Models from Deep Learning with Inductive Biases* by Cranmer et al., presents a framework for discovering empirical laws in high-dimensional data [1] by leveraging both deep learning, specifically Graph Neural Networks (GNNs) with sparse representations, and symbolic regression. In this project, we successfully reproduce the key results of the original paper, focusing on the reconstruction of known force laws from simulation datasets. Our findings demonstrate the robustness and generalisability of the framework across different physical systems.

We build on the original work by introducing a new model variation: the pruning model. While the original study encourages sparse, low-dimensional representations through fixed architectural constraints and regularisation terms, our pruning approach dynamically reduces the message dimensionality throughout training. By progressively masking less informative components, the model adaptively compresses its representation to match the system's intrinsic dimensionality—without needing to predefine it in advance.

2 Methods

In this section, we first outline the overall framework of our project (Section 2.1), followed by a detailed description of the GNN architecture (Section 2.2.1) and the inductive biases that make GNNs well-suited to our task (Section 2.2.2). We then describe the different GNN model variations explored (Section 2.2.3), followed by the model training setup, including the dataset used (Section 2.3) and the training configuration such as duration, optimiser, and learning rate scheduler (Section 2.4). To assess whether the models have learned the correct force functions, we analyse linear fits between the true forces and the learned message representations (Section 2.5). Section 2.6 discusses hyperparameter selection for the pruning models. Finally, we explain the symbolic regression procedure used to extract force laws from the trained GNNs (Section 2.7).

2.1 Framework

In this project, we follow the same framework as the original paper. Firstly, we train a GNN to predict instantaneous particle accelerations from our dataset containing particle properties. To verify that the GNN has generally captured the true force function, we perform a robust linear regression by fitting the true forces to the network's internal message representations. Separately, we then use symbolic regression to attempt to distill force laws from the trained models.

2.2 Graph Neural Networks

2.2.1 Graph Neural Network Architecture

In this section, we follow the notation introduced by Battaglia et al. (2018) [4]. The structure of the GNN used in this project has two main components: nodes and edges. Each node contains information about a particle. If a pair of particles interact, then they have an edge between them. In our simulations, all particles interact, hence our graph is fully connected and undirected. The node information, $\mathbf{v}_i \in \mathcal{V} \subset \mathbb{R}^{L^v}$ where L^v is the number of node features, in our GNN includes

$$\mathbf{v}_i = [x, y, \dot{x}, \dot{y}, q, m] \tag{1}$$

where x, y are the 2-D positions of the particle i, $\dot{x}$ and $\dot{y}$ are the discretised velocities (distance moved in simulation time Δt), q is the charge and m is the mass. Hence, the dimensionality of the node features, L^e, depends on the dimensionality of the system. The

inputs to the model comprises of this information.

There are two multilayer perceptrons (MLPs) involved in the GNN [1] The first is the edge model (or edge function), $\phi^e : \mathcal{V} \times \mathcal{V} \to \mathcal{E}$, where $\mathcal{E} \subseteq \mathbb{R}^{L^e}$ and L^e is the dimensionality of the messages. In a forward pass of the GNN, the edge model takes the features of two connected nodes as inputs [2] and outputs the edge message, $\mathbf{e}'_k \in \mathcal{E}$, where k denotes the edge. We concatenate the node features when inputting them into the MLP. This function can be written as

$$\mathbf{e}'_k = \phi^e(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}) \tag{2}$$

where $\mathbf{v}_{r_k}$ denotes the features of the receiving node, and $\mathbf{v}_{s_k}$ denotes the sending node of edge k. The edge message encodes the interactions between these two nodes which, in our case, are the forces between the particles. Because the forces between the particles are symmetric, a well-trained, ideal GNN will produce the same output message whether one particle is treated as the sender and the other as the receiver, or vice versa.

The messages to receiving node i are aggregated through an element-wise summation [3],

$$\bar{\mathbf{e}}'_i = \sum_{j \neq i} \phi^e(\mathbf{v}_i, \mathbf{v}_j) \tag{3}$$

where $\bar{\mathbf{e}}'_i$ is the aggregated message.

The second MLP involved in the GNN is the node model (or node function), $\phi^v : \mathcal{V} \times \mathcal{E} \to \mathcal{D}$, where $\mathcal{D} \subseteq \mathbb{R}^D$ and D is the dimensionality of the system (if we are predicting accelerations)[4]. This model outputs the updated node features for a specific node. It takes the node features and the aggregated message for that node as an input and calculates the node update, $\hat{\mathbf{v}}'_i$.

$$\hat{\mathbf{v}}'_i = \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i) \tag{4}$$

In predicting particle dynamics, the node update could be the updated node features (eg. positions at the next time step) and instead we would have $\phi^v : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$. But, as we intend on distilling force laws from the GNN, we configure the node model to predict the particle accelerations. Hence, the target variable in our pipeline is the instantaneous accelerations of the particles. GNNs can be trained using backpropagation,

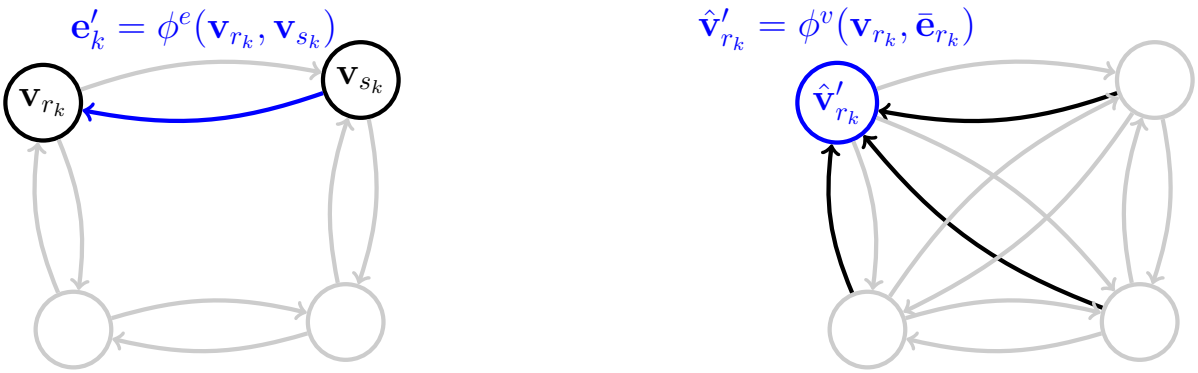
[1] The edge model and the node model do not necessarily need to be MLPs, rather just some differentiable function.

[2] In general GNN architectures, the edges may contain their own information too and the edge model thus has three inputs: $\phi^e(\mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_k)$.

[3] There are other aggregation operators other than an element-wise summation, such as mean aggregation. The summation aggregation works well for our system because the resultant force is a summation of individual forces acting.

[4] In general GNNs, there may also be a global model, which takes all aggregated messages and all updated nodes as inputs and outputs a global property for the entire graph [5].

as all components are end-to-end differentiable.



(a) Edge update (b) Node update

Figure 1: Updates during a GNN forward pass. Elements being updated are shown in blue, while elements involved in the computation are shown in black. This figure is adapted from Battaglia et al. (2018) [4]. Refer to Section 2.2.1 for the corresponding equations.

2.2.2 Inductive Biases of Graph Neural Networks

GNNs provide good frameworks for n-body simulations because of their inductive biases, which is the tendency for a model to prefer a certain subset of solutions over others independent of observed data [4].

1. *Graphs are permutation invariant.*
   The structure of a GNN involves nodes and edges, which are permutation invariant, which makes them good candidates for analysing n-body systems as the ordering of particle data does not affect dynamics.

2. *Pairwise interactions represented by edges.*
   Forces between pairs of particles are a function of their properties (positions, mass etc.). These can be well described by edge messages, which are a function of the features of receiving and sending nodes.

3. *Aggregation of messages analogous to summing forces.*
   The resultant force acting on a particle is the sum of the individual forces. GNNs can aggregate forces by element-wise summation, which naturally mirrors this physical principle.

4. *Node updates can predict dynamics.*
   The inputs to the node model are the properties of the node that is undergoing the

update, and the aggregated edge messages to that node. Hence, this MLP can be trained to predict the particle dynamics, in this case the instantaneous accelerations, as they are a function of the node properties and the resultant forces acting on the node.

While the GNN architecture may resemble physical systems, the models have much more flexibility. For example, while physical forces in an n-dimensional system are represented by n-dimensional vectors (e.g., 2D or 3D), the learned message vectors in a GNN are not constrained to match this and can exist in a higher-dimensional latent space. As we see later, we can add regularisation to encourage sparsity in the message vectors to allow us to make a more direct comparison to the force vectors. The edge and node model are arbitrarily learned functions. The comparison of Newtonian physics to GNNs is to demonstrate that these architectures are good for describing such systems.
Lastly, GNNs allow us to split the problem into two components, the edge model and the node model, which makes the symbolic regression tractable by constraining the search space.

2.2.3 Model Variations

As in the original paper, we configured four different variations of a GNN. We further added one extra variation - 'pruning'.

1. **Standard**. Consisted of a GNN where the message dimension, L^e, was set to 100.

2. **Bottleneck**. The message dimension was set to match the dimensionality of the system, which was 2 in all the experiments.

3. **L1**. We applied L1 regularisation, $\mathcal{L}_e$ to the edge messages,

$$\mathcal{L}_{L1} = \frac{\alpha_{L1}}{N^e} \sum_{k=0}^{N^e-1} |\mathbf{e}'_k| \tag{5}$$

   where N^e is the total number of parameters in the edge messages, and $\alpha_{L1} = 10^{-2}$ is the regularisation constant. The dimension of the message vectors were set to be 100, the same as the standard model variation. This regularisation encourages sparse representation of messages by the absolute values of the message components, effectively driving many of them toward zero.

4. **Kullback–Leibler (KL)**. We added a standard Gaussian prior, $\mathcal{N} \sim (\mathbf{0}, \mathbf{1})$, to the components of the messages. Unlike the other model variants, the edge model in

this case outputs both the mean and log-variance for each message element, which doubles the output dimensionality. This allows the messages to be treated as samples from a Gaussian distribution rather than fixed feature vectors, as shown below

$$\mathbf{e}'_k \sim \mathcal{N}(\boldsymbol{\mu}'_k, \mathbf{diag}[\boldsymbol{\sigma}'^2_k]) \tag{6}$$

where $\boldsymbol{\mu}'_k = \phi^e_\mu$ and $\boldsymbol{\sigma}'^2_k = \exp(\phi^e_{\sigma^2})$. We trained the model such that ϕ^e_μ are all even outputs and $\phi^e_{\sigma^2}$ are all odd outputs from the edge model. During training, the edge messages are sampled from the distribution defined by Equation 6 before being aggregated and inputted to the node model. A regularisation term equivalent to the KL-divergence between the standard normal prior and the probability distribution defined in Equation 6 is added to the loss:

$$\mathcal{L}_{KL} = \frac{1}{N^e} \sum_{k=0}^{N^e-1} \sum_{j=0}^{L^e-1} \frac{1}{2}(\mu'^2_{k,j} + \sigma'^2_{k,j} - \log(\sigma'^2_{k,j})) \tag{7}$$

The KL regularisation term encourages sparsity in the messages by penalising deviations from a standard normal distribution, effectively pushing the learned mean and variance of each message component toward zero mean and unit variance. If the model is in an evaluation setting, we do not sample and just take the message elements to equal the means.

5. **Pruning**. As training progresses, we incrementally zero-mask message dimensions so that by the end of training, the number of active message dimensions is equal to the true dimensionality of the system. This acts similarly to a bottleneck variation, as it forces the network to compress information into a small dimension and does not require an added regularisation term in the loss. To decide which dimensions to prune, we pass a random 10,240 data points from the validation set through the model and measure the standard deviation each message dimensions across all the edges. Dimensions with the lowest standard deviation are those which contribute less to the outcome of the model, and are pruned by applying a zero-mask. Pruning is a similar model variation to bottleneck, but has the advantage that we do not require knowledge of the dimensionality of the system beforehand - we could keep pruning dimensions until we see a large spike in the loss to imply that the model no longer has the complexity for the problem. We explored multiple pruning schedules to control how this masking is applied throughout training.

2.3 Dataset

The dataset used in training the GNN was created using the source code from the original paper code repository [6]. The name of the datasets used in our training, and their respective potentials (potential denoted V) are

- Charge
$$V = \frac{q_1 q_2}{r} \tag{8}$$
where q_1 and q_2 denote the charges of the interacting particles and r denotes the distances between them with the addition of a small $\epsilon = 10^{-2}$ for numerical stability.

- r^{-1}
$$V = m_1 m_2 \log r \tag{9}$$
where m_1 and m_2 denote the masses of the interacting particles.

- r^{-2}
$$V = -\frac{m_1 m_2}{r} \tag{10}$$

- Spring
$$V = (1 - r)^2 \tag{11}$$

Each sample in the input data contained a 2-D tensor containing the positions in x and y, the discretised velocities, ν_x, ν_y, the charges, q, and the masses, m, for 4 interacting particles. The target data included the accelerations of the particles, found by taking the negative derivative of their respective potentials (to calculate the force) and dividing by their mass.
Each dataset was created by running 10,000 different simulations over 1000 timesteps and then only collecting data from every 5th timestep to reduce correlation between samples. This resulted in datasets containing 1 million samples, which were then randomly split into training, validation and test sets with ratio 70/15/15.

2.4 Model Configuration and Training

2.4.1 Model Configuration

To create and train the GNNs, we used PyTorch [7] and PyTorch Geometric [8]. In all experiments, the edge model and node model MLPs contained three hidden layers each with 300 hidden units and ReLU activations between each layer.

2.4.2 Data Augmentation

The node model outputs predictions of the instantaneous accelerations of particles, as shown in Equation 2.2.1. Before passing the node features into the model, we augment the data by adding Gaussian noise with a standard deviation of 3 to the position coordinates of all nodes simultaneously. This follows the approach used in the original paper's code and was likely introduced to improve model robustness by reducing overfitting to precise spatial positions, while also simulating the presence of noise in real-world data.

2.4.3 Predictions and Loss

The base loss on our model was calculated to be the mean absolute error between the predicted accelerations, $\hat{\mathbf{v}}'_i$, and the actual accelerations, $\mathbf{v}'_i$, from our dataset.

$$\mathcal{L} = \frac{1}{N^v} \sum_{i=0}^{N^v-1} |\mathbf{v}'_i - \hat{\mathbf{v}}'_i| \tag{12}$$

L2 regularisation was also used in every training instance,

$$\mathcal{L}_{L2} = \frac{\alpha_{L2}}{N^l} \sum_{l=0}^{N^l} |w_l|^2 \tag{13}$$

where N^l are the total number of network parameters, denoted w_l, and $\alpha_{L2} = 10^{-8}$ is the L2 regularisation constant. Hence, the total loss is $\mathcal{L} + \mathcal{L}_{L2}$ for the standard, bottleneck and pruning model variations. For the L1 model variation, the loss is $\mathcal{L} + \mathcal{L}_{L2} + \mathcal{L}_{L1}$, and for the KL model variation the loss is $\mathcal{L} + \mathcal{L}_{L2} + \mathcal{L}_{KL}$.

2.4.4 Training

To train our models, we performed gradient descent using the Adam [9] optimiser with a Cosine Annealing learning rate scheduler. Each model was trained for 100 epochs. As the batch size was 64, and there were 700,000 data points in the training set, this is equivalent to training the model for approximately 1.1 million training steps. The training and validation loss was monitored every epoch. Experiment tracking was done using WandB [10].

2.5 Messages as Linear Transformations of True Forces

The original paper argues that for a GNN that has learned to predict accelerations from particle properties, the message vectors are linear transformations of the true forces,

provided that the message dimension is equal to the dimensionality of the forces. This can be shown mathematically.

In Newtonian mechanics, the resultant force, $\bar{\mathbf{f}}_i$, acting on particle i is equal to the sum of the individual forces, $\mathbf{f}_k$.

$$\sum_k \mathbf{f}_i = \bar{\mathbf{f}}_k \tag{14}$$

If we ignore the particle mass, the node model predicts the resultant force on the receiver particle r_k.

$$\bar{\mathbf{f}}_i = \hat{\mathbf{v}}'_i = \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i) = \phi^v\left(\mathbf{v}_i, \sum_{r_k=i} \mathbf{e}'_k\right) \tag{15}$$

If we only consider a single interaction, and hence a single edge, the force is

$$\bar{\mathbf{f}}_i = \mathbf{f}_{k,r_k=i} = \phi^v(\mathbf{v}_i, \mathbf{e}'_{k,r_k=i}) \tag{16}$$

Again, the resultant force is the sum of the individual forces, so we can use the above equation in the many-particle case and equate this to Equation 2.5. Explicitly,

$$\sum_{r_k=i} \phi^v(\mathbf{v}_i, \mathbf{e}'_k) = \phi^v\left(\mathbf{v}_i, \sum_{r_k=i} \mathbf{e}'_k\right) = \bar{\mathbf{f}}_i \tag{17}$$

which demonstrates that ϕ^v is a linear function in its second argument:

$$\phi^v(\mathbf{v}_i, \mathbf{e}'_a + \mathbf{e}'_b) = \phi^v(\mathbf{v}_i, \mathbf{e}'_a) + \phi^v(\mathbf{v}_i, \mathbf{e}'_b) \tag{18}$$

Provided ϕ^v is invertible in $\mathbf{e}'_k$, which is true when the dimensionality of $\mathbf{e}'_k$ matches the dimensionality of $\hat{\mathbf{v}}'_i$, then you can invert Equation 2.5.

$$\mathbf{e}'_k = (\phi^v(\mathbf{v}_i, \cdot))^{-1}(\mathbf{f}_k) \tag{19}$$

Hence, the messages $\mathbf{e}'_k$ are just linear transformations of the true forces $\mathbf{f}_k$. If we constrain the message dimensionality to match that of the physical system, we can fit the learned messages using a linear regression on the true forces. A strong linear correspondence indicates that the model has successfully captured the underlying physical forces.

2.5.1 Choosing the Best Messages

The bottleneck and pruning model variations force the model to constrain the message dimensions to the dimensionality of the system. The other model variations, however,

have a message dimension larger than that of the system - we need to only choose the best messages for the linear regression.

For the standard and L1 model variations, we choose the most important messages depending on the standard deviation across the samples in the test set. We pass the test set through the trained edge model and this outputs a tensor containing the messages for all edges in the set (the total number of edges would be the number of edges in one sample multiplied by the number of samples in the test set). We calculate the standard deviation of each message dimension over all the edges, and choose the top two dimensions (because our system is 2-D) for the linear regression. A message element with high standard deviation across samples suggest that they contain meaningful information and that the model is using them to capture differences in the input that influence the output. Figure 2a shows a visual depiction of how the best message dimensions are chosen for the standard and L1 model variations.

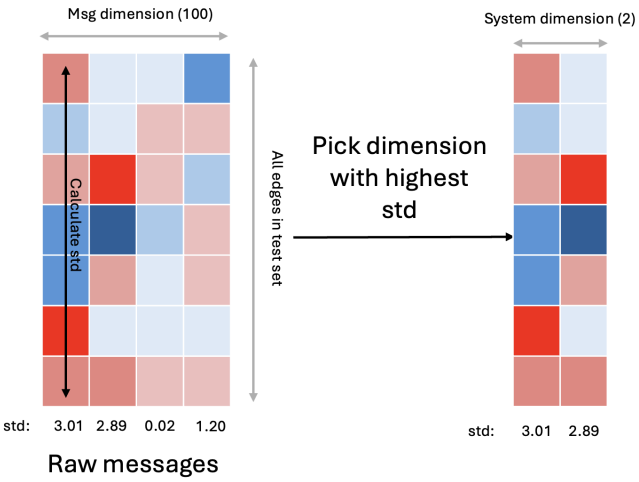
For the KL model variation, the edge model outputs the mean and log-variance of the message elements, which defines a probability distribution for each message element. To choose the best message dimension, we calculate the KL divergence between the message element probability distribution and a standard Gaussian prior, as in Equation 7, for all of the messages. We then find the average of the KL divergences for each message across all of the edges and choose the top two message elements as the ones with the highest KL divergence. Figure 2b shows a visual depiction of how the best message dimensions are chosen for the KL model variation.

The chosen elements are then scaled to have zero mean and unit variance over the edge samples.

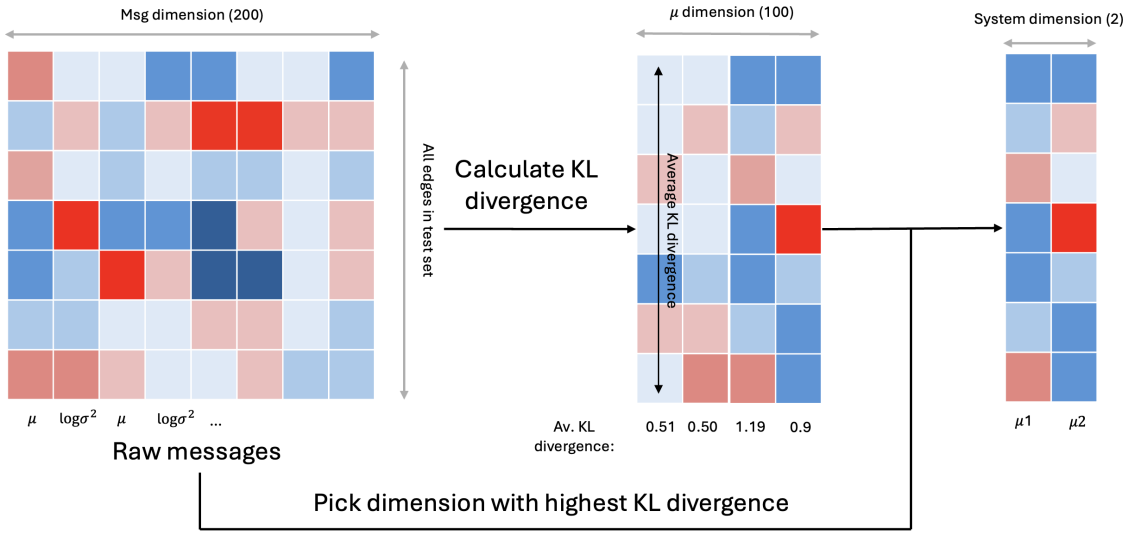
2.5.2 Fitting the Forces to the Messages

The expected forces for the our systems are (ignoring constant coefficients):

- Charge Force
$$\mathbf{F} = \frac{q_1 q_2}{r^2}\hat{\mathbf{r}}$$
- r^{-1} Force
$$\mathbf{F} = -\frac{m_1 m_2}{r}\hat{\mathbf{r}}$$
- r^{-2} Force
$$\mathbf{F} = -\frac{m_1 m_2}{r^2}\hat{\mathbf{r}}$$
- Spring Force
$$\mathbf{F} = -(r-1)\hat{\mathbf{r}}$$



(a) **Standard and L1 model.** The edge model outputs the raw messages. We calculate the standard deviations of the message elements over all the edges in the test set. The best messages are those with the highest standard deviation.



(b) **KL model.** The edge model outputs both the means and the log-variances. We calculate the KL divergence between the message probability distribution and a standard normal distribution (see Equation 7) and average these over all edges. The best messages are those with the highest KL divergence.

Figure 2: Choosing the best message dimensions to be fitted by a linear combination of the true forces. The grey grids depict a tensor containing the edge messages for all of the edges in the test set.

which can be found by taking the negative derivative of the potentials in Equations 8-11. We need to perform a linear regression to fit the message elements as a linear combination of force components:

$$\text{msg}_1 = a_1 + b_1 F_x + c_1 F_y \tag{20}$$

$$\text{msg}_2 = a_2 + b_2 F_x + c_2 F_y \tag{21}$$

2.5.3 Robust Linear Regression and R^2 Calculation

Recapping Section 2.5, if the GNN has learned to predict accelerations from particle data, then the edge model has learned the true underlying interaction force of the system as the messages outputted from the edge model are just rotations of the true force. The edge model, however, is just a piece-wise linear learned function which can produce erratic outputs, especially for out-of-distribution data [11]. We just want to test whether the model has generally learned the correct underlying force function, so we performed a robust linear regression, where we ignored the worst 10% of points when fitting the coefficients in Equations 20 and 21. Finally, to find the best coefficients, we used the SciPy minimize function with the Powell algorithm [12] to minimise the mean-squared error between the messages and linear combination of forces.

To evaluate the goodness of fit of the linear regression to the message dimensions, we calculated the R^2 score, again, ignoring the worst 10% performing points in this calculation. If the edge model has learned the true forces, then the messages will be a transformation of the forces producing a R^2 score close to 1. A visual depiction of how the entire process of plotting the linear combination of forces to the messages is described in Figure 3.

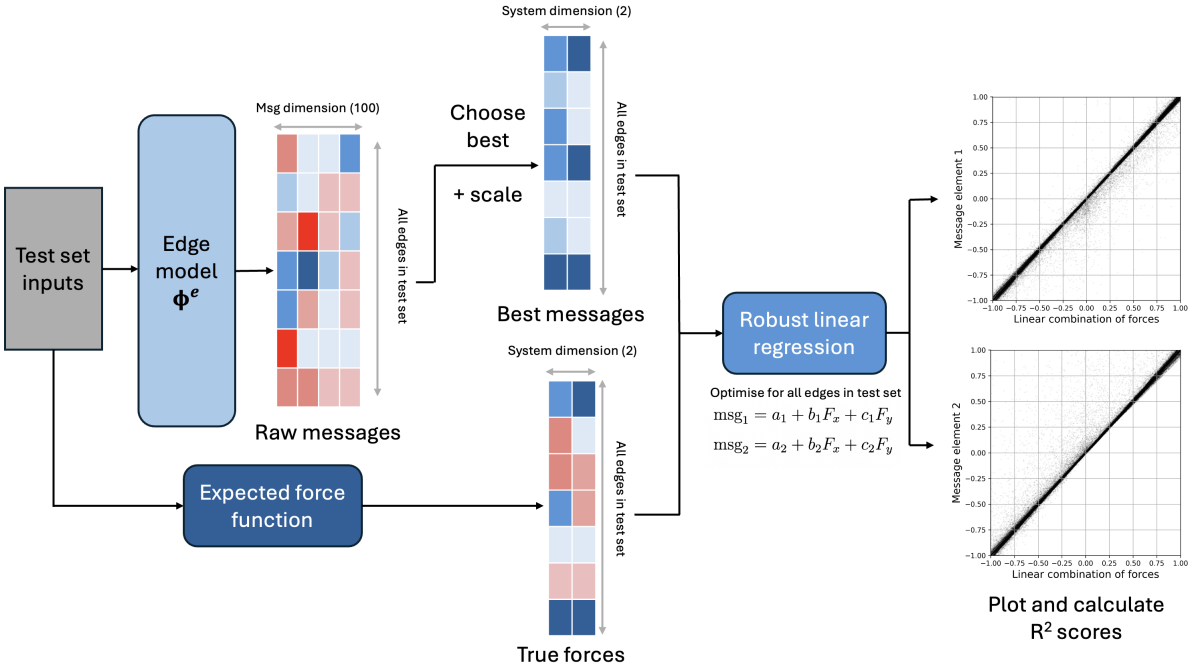


Figure 3: How we plotted the the message elements as a linear combination of true forces acting on the particles. The edge model takes the features of the source and receiver nodes as inputs, and outputs a message. The message is of dimension 100 for the standard and L1 model variations, as shown in the figure. The edge model was evaluated on the data from the test set and the messages were extracted for every edge. We choose the best message elements by calculating the standard deviation across the edges, and then scale them. As we know the true interaction forces between the particles, we can can calculate the forces acting between particles from the node features in the test set. We perform a robust linear regression to fit the message elements as a linear combination of the true forces in the $x-$ and $y-$ direction and plot these. If the model has learned the true forces, then the R^2 score should be near 1. Not depicted here: For the KL model variation, the edge model outputs a message dimension of 200, and we choose the best messages using the KL divergence instead of the standard deviation. The bottleneck and pruning variations do not require us to choose the best message dimension as the message dimension is already equivalent to the dimensionality of the system.

2.6 Pruning Hyperparameter Tuning

As an extension to the original paper, we introduce a new model variation - 'pruning' - where we zero-mask message dimensions as training progresses such that we are only left with two active dimensions at the end of training. The rate at which the pruning occurs, and the epoch where pruning finishes, are hyperparameters which we have tuned via a grid search procedure. In all cases, we begin pruning after the first epoch (approximately 11,000 optimiser steps).

There were three different pruning schedules that we trialed in the hyperparameter search:

1. **Linear**. The number of active dimensions decreases linearly as training progresses. Letting t be the current epoch, T be the epoch at which pruning finishes, D_m be the total number of message dimensions, and ΔD be the total number of dimensions that we need to prune, the number of active dimensions at epoch $t \leq T$, D_a are

$$D_a(t) = D_m - \texttt{ceil}\left[\Delta D \frac{t}{T}\right] \tag{22}$$

   `ceil` is the ceiling operator that rounds up to the nearest integer to keep the number of pruned dimensions an integer value.

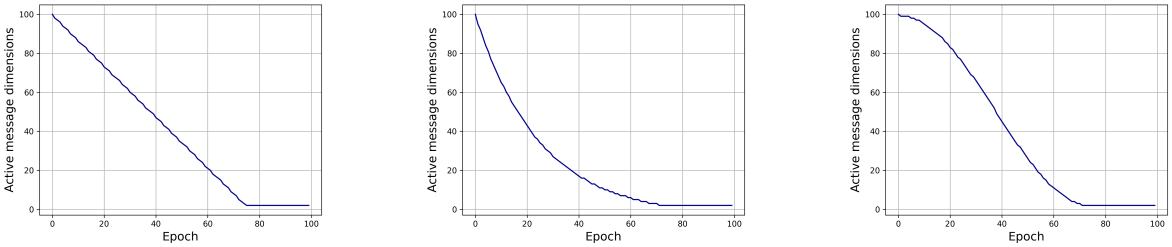
2. **Exponential**. The number of active dimensions at epoch $t \leq T$, D_a are

$$D_a(t) = D_m - \texttt{ceil}\left[\Delta D \frac{1 - e^{kt/T}}{1 - e^{-k}}\right] \tag{23}$$

   where $k = 3$ is a decay constant.

3. **Cosine**. The number of active dimensions at epoch $t \leq T$, D_a are

$$D_a(t) = D_m - \texttt{ceil}\left[\Delta D \left(1 - 0.5\left(1 + \cos\left(\frac{\pi t}{T}\right)\right)\right)\right] \tag{24}$$



(a) Linear. (b) Exponential. (c) Cosine.

Figure 4: A plot of the active message dimensions as training progresses for the different pruning schedules. For each of these plots, pruning finishes on epoch $T = 75$. The total number of message dimensions are $D_m = 100$ and the number of dimensions that need to be pruned are $\Delta D = 98$, as is the case in our pruning experiments.

We also varied the point in training at which pruning was completed as part of our hyperparameter search, trialing pruning end points at 65%, 75%, and 85% of the total

training duration. We trained the nine different pruning models with the same configuration and length as the other model variations (100 epochs, Adam optimiser, Cosine Annealing Scheduler etc.) so we could directly compare the results. To choose the best hyperparameters, we evaluated the model on the validation set, extracted the messages, and calculated the R^2 score of the linear fit of the true forces to the messages (the process described in Section 2.5). This was because the model with the lowest mean-absolute error (MAE) loss on the target variable was not necessarily the one that learned the force function the best. We know this empirically from the training and evaluating the other model variations. The hyperparameter search was performed on the charge dataset as this was found to have the worst R^2 values for the other model variations.

2.7 Symbolic Regression

By demonstrating that the edge model outputs messages that are linear combinations of the true force, we show that the edge model has learned the interaction force between particles. MLPs, however, are not interpretable as their internal representations are high-dimensional and distributed, making it difficult to extract explicit physical relations. Symbolic regression is a method of distilling analytical equations from these learned representations by fitting compact, closed-form expressions to the inputs and outputs of specific components of the network—such as the edge model—thereby enabling the recovery of interpretable physical laws that govern the underlying system.

2.7.1 Combining Deep Learning with Symbolic Regression

A key motivation for using deep learning to assist symbolic regression is that direct application of symbolic regression to high-dimensional physical datasets is computationally intractable. For example, consider our n-body simulation data, where each sample consists of multiple particles with properties such as positions, velocities, masses, and charges. If we aim to discover the scalar acceleration of particle given its interaction with all other particles, the full input would then include features of every interacting particle pair. Applying symbolic regression directly would require searching over an enormous function space.

Instead, we factorise the problem by first training a GNN with inductive biases that localise the computation — splitting the overall mapping into an edge model that learns pairwise interactions (messages), and a node model that aggregates them. Symbolic regression is then applied only to the low-dimensional outputs of the edge model, drastically reducing the dimensionality of the regression problem and making recovery of interpretable force

laws computationally tractable.

2.7.2 *PySR*

We performed symbolic regression on the edge model using the *PySR* package [13]. *PySR* uses a genetic algorithm to search for the best equations to describe a dataset (target variable) given some input variables and operators. The target variables, in our case, are the two best edge messages, which were found in the same way as described in Section 2.5.1. The input variables are m_1, m_2, q_1, q_2 as well as the relative positions to make the symbolic regression more efficient: $\Delta x = x_1 - x_2$, $\Delta y = y_1 - y_2$, and $r = \sqrt{\Delta x^2 + \Delta y^2} + 10^{-2}$. We added a small constant to the distance r to match the form used in the original potential equations when generating the dataset.

PySR performs symbolic regression by maintaining a large population of candidate equations, each represented as a tree of mathematical operations. These equations evolve over time through random modifications known as mutations (e.g. changing an operator or replacing a variable) and crossovers (swapping parts between equations). At each generation, *PySR* selects the most promising equations using a fitness function that balances prediction error (in this case MAE) and equation simplicity. We are motivated to seek simplicity in our equations as simple equations tend to generalise well and are easier to interpret. The complexity of an equation is the number of nodes in the equation tree. Periodically, the equations are simplified, such that redundancies are removed, and constants are optimised using a gradient-based optimisation algorithm. *PySR* outputs a Pareto front of equations, providing the most accurate equation at different complexity levels.

PySR selects the best equation by maximising the fractional drop in log mean absolute error (MAE) relative to an increase in model complexity. Specifically, it chooses the point on the Pareto front that maximises the score given by

$$\text{score} = -\frac{\log\left(\text{loss}_i/\text{loss}_{i-1}\right)}{\text{complexity}_i - \text{complexity}_{i-1}} \tag{25}$$

which reflects the greatest gain in accuracy per unit of additional complexity [14]. However, often times the best equation is not the one that successfully recovers the force law, so we considered all equations on the Pareto front. With regard to scientific discovery, this approach is reasonable since one can explore all candidate functional forms returned by *PySR* and evaluate which best captures the underlying physical law, with constants fine-tuned separately as needed.

2.7.3 Configuration

The operators that were allowed in the symbolic regression were $+, -, \times, \texttt{inv}(\cdot)$, which had complexity of one, as well as `exp`, `log` and `^`, which we set to have complexity of three, as they are more complex operators. The complexity of constants and input variables were set to be one.

We chose a random set of 5,000 examples from the test set for the symbolic regression. For all of the tests, we ran the symbolic regression for enough iterations for the Pareto front of best equations to remain mostly stable, as genetic algorithms do not tend to converge. This was usually 6,000 iterations, except for the KL model on the r^{-1} dataset, and all models on the charge dataset, which required 7,000 iterations.

PySR can take in a *parsimony* argument, which is a multiplicative factor for how much complexity is punished in equations, and this was set to 0.05. Lastly, *PySR* also allowed us to change the maximum size of equations permitted (measured in terms of complexity), which we set to 23 for all datasets, except the charge dataset where this was set to 25. There needed to be more flexibility for the algorithm to find the appropriate equation for the charge dataset.

The remaining *PySR* hyperparameters were kept at their default values. Specifically, the number of populations was set to 31, with 27 individuals in each population. *PySR* evolves these populations in parallel, and after each iteration the equations can migrate between populations. The number of mutations per 10 individuals per iteration was set to 380.

2.7.4 Successful Recovery of Equations

Below are the forms we should expect if we have successfully distilled the true force law from the GNN:

- Charge
$$\text{msg} = \frac{\mathbf{a}(\Delta x, \Delta y)}{r} \cdot \frac{q_1 q_2}{r^2} + b$$
- r^{-1}
$$\text{msg} = \frac{\mathbf{a}(\Delta x, \Delta y)}{r} \cdot \frac{m_1 m_2}{r} + b$$
- r^{-2}
$$\text{msg} = \frac{\mathbf{a}(\Delta x, \Delta y)}{r} \cdot \frac{m_1 m_2}{r^2} + b$$
- Spring
$$\text{msg} = \frac{\mathbf{a}(\Delta x, \Delta y)}{r} \cdot (r - 1)$$

where $\mathbf{a}(\Delta x, \Delta y) = a_1 \Delta x + a_2 \Delta y$ is an arbitrary linear transformation of relative displacements Δx and Δy, and b is an arbitrary constant. In principle, the form of the equations should include both r and $r + \epsilon$ terms to reflect the true form of the potential used during dataset generation. For example with the r^{-1} force, the recovered equation should be $\frac{\mathbf{a}(\Delta x, \Delta y)}{r} \cdot \frac{m_1 m_2}{r+\epsilon} + b$. However, this discrepancy was negligible in practice and did not significantly affect the results of symbolic regression.

3 Results and Comparison

3.1 Methodology Differences

For the purposes of reproducibility, we aimed to follow the same configurations as the original paper when training our models, including the choice of optimiser and hyperparameters. However, certain aspects of the training pipeline were not clearly specified, and our implementation may differ in the following ways:

- It is unclear from the original paper the learning rate scheduler that was used, but from the paper's accompanying repository [6], we believe that the OneCycleLR scheduler [15] was used. We used the Cosine Annealing scheduler instead, as OneCycleLR is more commonly used for shorter training regimes [16].

- It was implied that models were trained for 200 epochs in the original work, which was equivalent to 1 million steps for their batch size and dataset size. We trained our models a similar amount - 1.1 million steps, but had a larger batch size of 700,000 instead of 500,000. It should be noted that in the paper the training set size, batch size and training length was not mentioned, so knowledge of this was inferred from the demonstration Colab notebook.

- The original work *eureqa* [17] for the symbolic regression of messages, whereas we used *PySR* as it is open-source and allows for more flexibility in altering the hyperparameters for symbolic regression. We used a reduced set of operators during symbolic regression compared to the original paper, in order to improve efficiency.

- The original work applied symbolic regression only to the single most important message element, whereas we applied it to the top two message elements to match the dimensionality of the system.

3.2 Pruning Experiments

To find the best pruning hyperparameters, we computed the R^2 values of the linear combination of forces fitted to the two most important messages across different pruning hyperparameter settings. The results are shown in Table 1.

Decay Schedule	**End Point**	**Average R^2 Value**
Linear	0.65	0.9871
	0.75	0.9936
	0.85	0.4707
Exp	0.65	0.9912
	0.75	0.9953
	0.85	0.9730
Cosine	0.65	0.9977
	0.75	0.9949
	0.85	0.9705

Table 1: Results from the pruning experiments. We averaged over the R^2 value for both messages.

The best-performing hyperparameter configuration was a cosine decay schedule with pruning ending at 65% of training. This likely worked well because the gradual nature of cosine decay allowed the model to adjust smoothly to decreasing dimensionality, while halting pruning relatively early gave it sufficient time to optimise within the final dimensional space. Notably, the linear decay schedule caused a sharp spike in training loss at the end of pruning—likely due to its abrupt dimensionality reduction compared to the more gradual transitions in the other schedules (see Figure 4).
Hence, we used this pruning configuration for the remaining datasets. The original paper did not include this model variation, so this is an extension to the original work.

3.3 Model Training

The prediction losses on the test set for each simulation and model type are shown in Table 2, and the results from the original paper are shown in Table 3. The losses were calculated using MAE as shown in Equation 12 (regularisation terms were not included in the loss). For the KL model variation, we did not sample the messages from the distribution defined in Equation 6 as during training. Instead, we used the mean values of the distributions

directly as the message elements.

Simulation	**Standard**	**Bottleneck**	**L1**	**KL**	**Pruning**	***Variance***
Charge	18.20	19.12	18.06	39.80	20.35	56417.66
r^{-1}	0.26	0.25	0.32	15.40	0.30	87.69
r^{-2}	24.07	25.25	21.67	57.80	25.71	98641.90
Spring	0.24	0.16	0.20	7.29	0.23	55.84

Table 2: Prediction losses of the model types on the test sets of the different simulations and the variance of the test set.

Simulation	**Standard**	**Bottleneck**	**L1**	**KL**
Charge	49	50	52	60
r^{-1}	0.077	0.069	0.079	3.5
r^{-2}	1.6	1.6	1.2	9.3
Spring	0.047	0.046	0.045	1.7

Table 3: Prediction losses of the model types on the different simulations from the original paper.

Comparison to Original Work

The prediction losses are different to the original paper, which is to be expected as neural network training is inherently stochastic, but ours are an order of 10 higher than except for the charge simulation, where our losses were lower. It is unclear why this is the case. Reasons could include different scaling for the test predictions or differences in the training pipeline (see Section 3.1 for methodology differences). However, the precise test loss is inconsequential to the framework, as we are more interested in distilling the force laws from the data.

3.3.1 Challenges in GNN Learning

The GNN finds it more difficult to learn the dynamics for the r^{-2} and charge systems. This is likely due to the highly sensitive nature of the underlying force laws, which diverge as $r \to 0$. In particular, both the r^{-2} and charge forces scale with $1/r^{-2}$ meaning that small variations in particle separation lead to disproportionately large changes in force

magnitude. This creates steep gradients that are difficult for the neural network to approximate, especially when using standard MLP-based edge models. Moreover, the charge force introduces additional complexity due to its dependence on the product $q_1 q_2$ which varies in sign and leads to discontinuous shifts in the direction of the force. This nonlinearity and potential for sign flipping further increase the difficulty of accurately representing the force using a smooth, differentiable model.

In addition to these modelling challenges, the charge and r^{-2} datasets exhibit significantly higher variance in the target acceleration values compared to the r^{-1} and spring datasets, naturally leading to larger absolute prediction errors unless the model can fully capture the system's complexity However, even after accounting for this by normalising the MAE by the standard deviation of the test set (i.e., $(\text{MAE}/\sqrt{\text{variance}})$), the charge and r^{-2} systems still perform noticeably worse than the others. This indicates that the reduced performance cannot be fully explained by the scale of the target values alone, and instead reflects intrinsic difficulties in learning the dynamics of systems with sharp singularities and sign-sensitive interactions.

3.4 R^2 of Linear Combination of Forces

The R^2 values of the linear combination of forces fitted to the two most important messages for the different model types and simulations are shown in Table 4, and the results from the original work are shown in Table 5.

Simulation	**Standard**	**Bottleneck**	**L1**	**KL**	**Pruning**
Charge	0.466	0.995	0.692	0.029	0.998
r^{-1}	0.414	0.839	0.844	0.594	0.838
r^{-2}	0.833	0.967	0.830	0.866	0.973
Spring	0.920	1.000	0.998	0.801	0.952

Table 4: Averaged R^2 values of the linear fits of the forces to the most important messages for the different simulations and model types. The 10% outliers were not included in the linear fit or R^2 calculation.

Simulation	Standard	Bottleneck	L1	KL
Charge	0.016	0.947	0.004	0.185
r^{-1}	0.000	1.000	1.000	0.796
r^{-2}	0.004	0.993	0.990	0.770
Spring	0.032	1.000	1.000	0.883

Table 5: R^2 values of the linear fit of the true forces to the learned messages from the original paper.

Comparison to Original Work

Our results broadly align with those reported in the original paper, with the exception of the standard model variation, for which we observed significantly higher R^2 values. The bottleneck, L1, and pruning model variations consistently performed best, demonstrating the greatest success in capturing the underlying interaction forces. There was also a difference in the fit for the L1 model on the charge dataset, as this model variation performed better in our reproduction than in the original work.

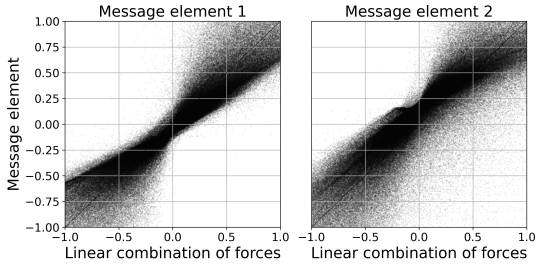
3.4.1 Data Handling

In Section 2.5.3, we described the procedure used to ignore outliers when fitting the true forces to the message vectors and computing the corresponding R^2 values such that these values were robust to outliers. The original paper, however, does not mention any data handling. Although the demonstration Colab notebook in the original repository includes a method for robust linear fitting, it does not provide code for robust R^2 calculation. Without applying this robust handling, we observed significantly lower R^2 scores compared to those reported in the original work, as shown in Table 6. Therefore, we believe that the original authors ignored outliers when performing the linear fit of true forces to the message elements and when calculating the R^2 values, but left these details out in the paper.

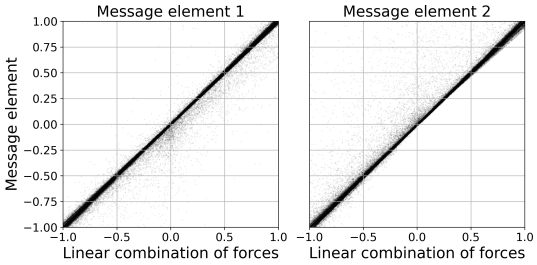
Figure 5 shows plots of the true forces fitted to the important messages for the different model variations for the spring simulation, without the robust fitting. Data points that are closer to the $y = x$ line signify that the linear combination of forces fit the message elements well, and thus would have a better R^2 value. These are similar to the results in the original paper, however our standard model variation produces a plot with data points more clustered to the $y = x$ line.

Simulation	Standard	Bottleneck	L1	KL	Pruning
Charge	0.0428	0.5777	0.2294	0.2550	0.6868
r^{-1}	0.1462	0.3496	0.3500	0.1581	0.3570
r^{-2}	0.0013	0.3842	0.0115	0.2066	0.4379
Spring	0.6980	0.9936	0.9932	0.6487	0.6845

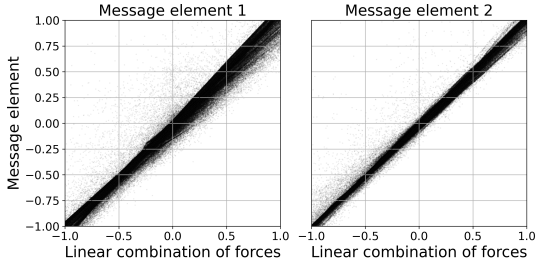
Table 6: Averaged R^2 values of the linear fits of the forces to the most important messages for the different simulations and model types. All datapoints were included in the fitting of the true forces to the messages, and the calculation of R^2.



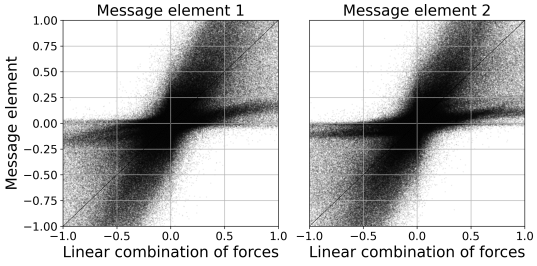
(a) Standard



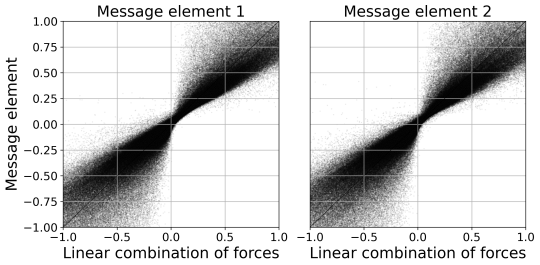
(b) Bottleneck



(c) L1



(d) KL

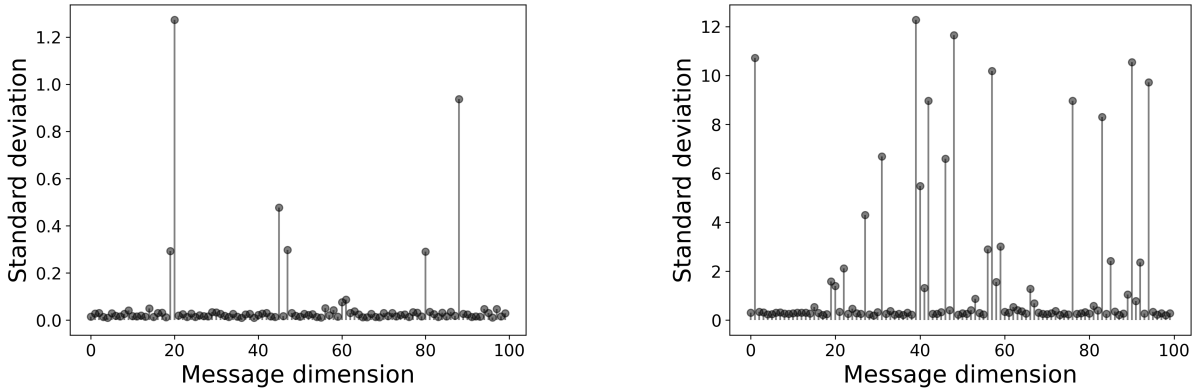


(e) Pruning

Figure 5: Linear fits of the true forces to the messages components for the spring dataset. All datapoints are included in the fit. The diagonal black line shows a $y = x$ line, as points that lie close to this line signify that the true forces fit well to the messages. Only a subset of 50,000 points from the test set were used to plot these, to reduce overcrowding. The R^2 values of these plots are shown in Table 4.

3.4.2 Spread of Representation

In some cases, particularly for the standard model variation, the most important message had a much higher R^2 score than the second most important message. We believe this is because the model is spreading out the representation to higher dimensions. We can see this directly from Figure 6, which is a plot of the standard deviations of the message elements from the L1 and standard model variations evaluated on the r^{-1} test set. It is apparent that the L1 model uses a much sparser latent representation than the standard variation, as only a small number of message elements have a high standard deviation. Message elements with low standard deviations do not vary across different inputs hence are not used by the model.



(a) L1 (b) Standard

Figure 6: The standard deviation of the message elements across the test set for the standard and L1 models trained on the r^{-1} dataset.

3.5 Reconstructing Force Equations

Table 7 summarises the success of symbolic regression in recovering the underlying force laws across various systems and model variations, while Table 8 presents the corresponding results from the original study. Unlike the original authors, who applied symbolic regression only to the most significant message component, we extended the analysis to include the top two message components.

Simulation	Message	Standard	Bottleneck	L1	KL	Pruning
Charge	1	✗	✓§	✓§*	✗	✓‡*
	2	✗	✓	✗	✗	✓‡*
r^{-1}	1	✓	✓	✓	✗	✓
	2	✓	✓	✓	✗	✓
r^{-2}	1	✗	✓	✗	✓§	✓
	2	✗	✓	✗	✗	✓
Spring	1	✓†	✓	✓	✗	✓‡
	2	✓†	✓	✓§	✗	✓‡

Table 7: Symbolic regression results for each message component.
✓= correct form of force law recovered; ✗= failure.
* Correct form, but with a small constant added a term (eg. $1/(r + \text{const.})$).
† Correct form, but only Δy apparent in both messages.
‡ Correct form with Δx in one message and Δy in the other.
§ Correct form with only Δx or Δy in at least one message.

Simulation	Standard	Bottleneck	L1	KL
Charge	✗	✓	✗	✗
r^{-1}	✗	✓	✓	✓
r^{-2}	✗	✓	✓	✗
Spring	✗	✓	✓	✓

Table 8: Symbolic regression results from the original work. Symbolic regression was only done on the top message component from each of the model variations.
✓= correct form of force law recovered; ✗= failure.

In the standard model variation trained on the spring system, both top message components primarily captured Δy, with Δx information dispersed across other message elements. The standard model variation has a higher tendency to disperse information across higher dimensions as it does not have any regularisation encouraging sparse representation, making it more difficult to recover interpretable physical equations.
The pruning model variation performed well, successfully recovering the true force equations across all systems. By progressively reducing the message dimensionality during training, the model was forced to compress information into a minimal, task-relevant subspace. This encouraged disentangled and interpretable representations, making symbolic regression

more effective. Its consistent success across different force laws demonstrates pruning as a robust strategy for extracting physically meaningful structures from learned GNN representations.

Comparison to Original Work

As in the original paper, the bottleneck model was the most effective at recovering the true force laws. Notably, we were able to reconstruct the charge force using the L1 model — an outcome not achieved in the original work. Conversely, we were unable to recover the r^{-2} force law using the L1 model, despite its successful reconstruction in the original study. Our standard model variation outperformed that of the original paper, which reported no successful recoveries with the standard model across any dataset. However, our KL model yielded poorer results by comparison.

3.5.1 Examples of *PySR* Reconstructions

Below are some examples of successful reconstructions of the true forces:

- Spring; bottleneck

$$\texttt{msg1} = \left(\frac{1}{\texttt{r}} - 0.99950946\right) \cdot (0.8855752\Delta\texttt{y} + 1.8560125\Delta\texttt{x}) + 0.031805687$$

- r^{-2}; L1

$$\texttt{msg1} = \texttt{m}_2((\Delta\texttt{y} + 0.43513915) + \Delta\texttt{x}) \cdot \frac{1}{\texttt{r}^3}$$

- Charge; pruning

$$\texttt{msg2} = \frac{1}{(\texttt{r} + 0.036421545)\texttt{r}^2} \cdot \texttt{q}_1\texttt{q}_2 \cdot (\Delta\texttt{x} - 0.0331669) + 0.08641323$$

  (shows the correct functional form except there is a small constant added to one of the $1/r$ terms and a Δy term is not present).

Table 9 presents the Pareto front of candidate equations discovered by *PySR* for the second most significant message from the pruning model trained on the r^{-2} dataset. The highlighted equation shows the best equation as chosen by *PySR* as it produces the largest drop in log(loss) per unit of additional complexity (maximises the score as given in Equation 25). While the best equation identified by *PySR* in this case accurately reconstructs the underlying relationship, such success is not consistently observed across all datasets and model variations.

Complexity	Equation	Loss
1	$\phi_2^e = 0.08$	0.0888
5	$\phi_2^e = (\Delta x \cdot 0.00) + 0.08$	0.0885
7	$\phi_2^e = ((\Delta x + \Delta y) \cdot 0.00) + 0.08$	0.0882
8	$\phi_2^e = ((\Delta x \cdot \text{inv}(r)) \cdot 0.00) + 0.08$	0.0880
9	$\phi_2^e = (m_2 \cdot (\Delta y + \Delta x) \cdot 0.00) + 0.08$	0.0877
10	$\phi_2^e = (\text{inv}((r + \Delta x) \cdot r) \cdot -0.00) + 0.08$	0.0843
12	$\phi_2^e = (\text{inv}(r^3) \cdot (\Delta x \cdot 0.02)) + 0.08$	0.0687
14	$\phi_2^e = ((m_2 \cdot 0.01) \cdot (\Delta x \cdot \text{inv}(r^3))) + 0.08$	0.0513
16	$\phi_2^e = ((\Delta y + \Delta x) \cdot ((m_2 \cdot \text{inv}(r^3)) \cdot -0.01)) + 0.08$	**0.0260**
18	$\phi_2^e = ((\text{inv}(r^3) \cdot (\Delta x + \Delta y \cdot 0.59)) \cdot m_2) + 0.08$	0.0128
20	$\phi_2^e = (\text{inv}(r^3) \cdot (((m_2 + 0.06) \cdot 0.01) \cdot (\Delta x + \Delta y \cdot 0.59))) + 0.08$	0.0125

Table 9: Pareto front from *PySR* results on message 2 of pruning model on r^{-2} dataset. The highlighted equation shows the 'best equation' chosen by *PySR* according to the metric given in Equation 25. The constants have been truncated to two decimal points.

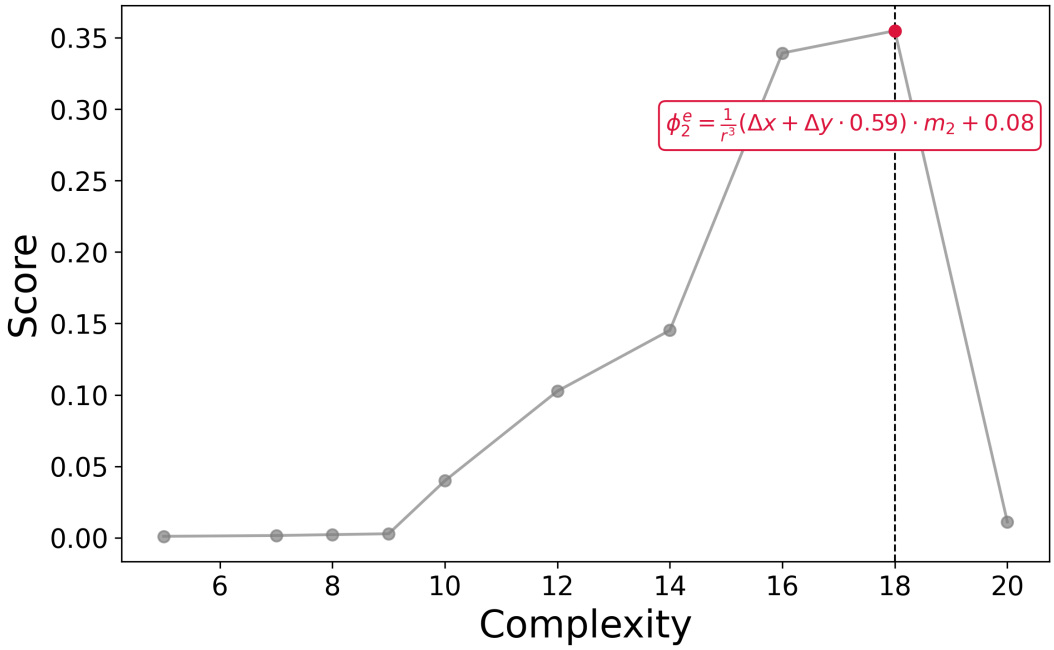


Figure 7: The score of the equations found from *PySR* for the pruning model trained on the r^{-2} as given in Table 9. The score is calculated from Equation 25. The best equation is highlighted in red as it produces the largest drop in log(loss) per unit of additional complexity.

In the reconstructions for the r^{-1} and r^{-2} force laws, the mass of the receiving node, m_1, is absent. As described in Equation 2, the edge model is intended to learn the interaction forces between particles, with the node model aggregating these messages and outputting

the resulting accelerations, as shown in Equation 26. However, this setup is mathematically equivalent to having the edge model learn accelerations directly, with the node model simply outputting the aggregated accelerations, as shown in Equation 27.

Edge model learns forces:

$$\begin{aligned}&\phi^e(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}) \approx \text{Force on } r_k \text{ by } s_k \\ &\phi^v\left(\mathbf{v}_{r_k}, \sum_{j \neq r_k} \phi^e(\mathbf{v}_{r_k}, \mathbf{v}_j)\right) \approx \phi^v(\mathbf{v}_{r_k}, \text{Resultant force on } r_k) \approx \text{Acceleration of } r_k\end{aligned} \tag{26}$$

Edge model learns accelerations:

$$\begin{aligned}&\phi^e(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}) \approx \text{Acceleration on } r_k \text{ caused by interation with } s_k \\ &\phi^v\left(\mathbf{v}_{r_k}, \sum_{j \neq r_k} \phi^e(\mathbf{v}_{r_k}, \mathbf{v}_j)\right) \approx \phi^v(\mathbf{v}_{r_k}, \text{Resultant acceleration of } r_k) \approx \text{Acceleration of } r_k\end{aligned} \tag{27}$$

4 Discussion and Conclusion

4.1 Reproducibility

We were successful in reproducing the main framework of the original paper, ie. training GNNs on particle simulation data, verifying that the model has generally learned the true force law, and distilling the true force law from the edge messages using symbolic regression. We also found similar trends in the results: for example, the bottleneck model seemed to perform the best for all datasets. However, we obtained much better results from the standard model variation than the original authors both in the R^2 value of the linear combination of forces (see Table 4 and 5) and in the symbolic regression of edge messages (see Table 7 and 8).

4.1.1 Stochasticity in GNN Training

The training of deep networks is inherently random — it is not possible to determine precisely the learned internal representations or outcomes from run to run due to stochastic elements such as weight initialisation, batch sampling, and optimiser dynamics. As a result, symbolic regression applied to a trained model will produce different expressions even when the model performance is consistent.

Because of this inherent stochasticity, it is also uncertain whether a given instance of the GNN will reliably recover the correct functional form of the true force law. An interesting extension to this project would be to train multiple instances of each GNN model variation on the same dataset and evaluate the proportion of runs in which symbolic regression successfully reconstructs the underlying force law. This would provide a more robust assessment of each model's reliability and the consistency of symbolic recovery under varying training conditions.

4.1.2 Colab Demonstration Notebook

We aimed to improve the reproducibility of the framework. The original code repository only contained code to initialise the graph neural networks, and an accompanying Colab notebook for demonstration purposes.
We have created a demonstration Colab notebook located in this project's code repository. This notebook is an improvement of the original in the following ways:

- Allows you to train any model variation (standard, bottleneck, L1, KL and pruning) on any dataset (charge, r^{-1}, r^{-2} and spring), whereas the original Colab notebook only had support for L1 and KL models on the spring dataset.
- Mounts your Google Drive to the notebook allowing you to save and load models easily.
- Runs the whole pipeline from data generation to the symbolic regression procedure, whereas the original Colab notebook did not contain the calculation of R^2 values and the symbolic regression procedure.

4.2 Limitations

In Section 3.5.1, we showed that the reconstructed equations for the r^{-1} and r^{-2} systems did not include m_1, implying that the edge model was learning accelerations instead of forces. The target variable for the GNN training is the accelerations of particles, so it is mathematically equivalent for the edge model to learn the accelerations instead.
This can be taken one step further as it would also be mathematically equivalent for the edge model to learn the force applied to the particle multiplied by any function of the features of the receiving node,

$$\phi^e(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}) \approx \text{Force on } r_k \text{ by } s_k \times h(\mathbf{v}_{r_k}) \tag{28}$$

where h is some arbitrary function, provided that the node model 'undos' this function,

$$\phi^v\left(\mathbf{v}_{r_k}, \sum_{j \neq r_k} \phi^e(\mathbf{v}_{r_k}, \mathbf{v}_j)\right) \approx \text{Acceleration of } r_k \times (h(\mathbf{v}_{r_k}))^{-1} \tag{29}$$

as the node model has node information of particle r_k. This is a limitation of the framework it introduces ambiguity in the physical interpretation of the learned edge messages — multiple mathematically equivalent but physically distinct representations can yield the same acceleration predictions. Although this is possible, it tends not to happen during training.

4.3 Conclusion

We successfully reproduced the framework of the original paper, demonstrating that empirical equations can be recovered from high-dimensional data by combining a GNN with inductive biases and symbolic regression. Our results align with the original findings in that the bottleneck model consistently performed best across all datasets. However, we observed notable differences in the performance of the standard and KL models: the standard model outperformed expectations in our experiments, while the KL model performed worse than reported in the original study.

Because of the innate stochasticity of training deep networks, it is not possible to determine the exact learned internal representation of the GNN, hence the precise reconstructed equation may differ across training runs.

Finally, we extended the original work by introducing a new pruning model variation, which dynamically reduces the dimensionality of the edge messages during training. This approach performed comparably to the bottleneck model—as expected, given that both impose constraints on message dimensionality—but offers the key advantage of not requiring the target dimensionality to be specified beforehand.

5 Software

Software used in this project:

- Numpy [18], Matplotlib [19], Pandas [20]: data analysis and plotting
- PyTorch [7], PyTorch Geomtric [8]: creating the GNN architecture and training.
- Accelerate [21]: managing GPU training with minimal code changes.
- PySR [13]: symbolic regression.
- Scipy [12]: linear regression.

5.1 Autogeneration Tools

Claude Sonnet 4 and ChatGPT-4o were used in both the code and report. We have listed below precisely what we used these autogeneration tools for.
Code:

- Writing the docstrings for the functions and classes in the repository.
- Writing comments in the code.
- Checking for bugs (eg. 'INSERT ERROR MESSAGE what is causing my code to break?')
- Plotting aid (eg. 'How do I make the axis labels and the ticks larger in Matplotlib?')

Report:

- Drafting sections of the report (eg. 'Help me draft an abstract. Look in particular at the introduction and conclusion INSERT UNFINISHED REPORT')
- Rephrasing wording (eg. 'Make this paragraph read better INSERT PARAGRAPH')
- Creating results tables in Latex (eg. 'Help me create a results table in Latex with these results INSERT RESULTS').
- Proof-reading (eg. 'Point out mistakes I made in my paragraph INSERT REPORT PARAGRAPH').

Bibliography

[1] Miles D. Cranmer, Alvaro Sanchez-Gonzalez, Peter W. Battaglia, Rui Xu, Kyle Cranmer, David N. Spergel, and Shirley Ho. "Discovering Symbolic Models from Deep Learning with Inductive Biases". In: *CoRR* abs/2006.11287 (2020). arXiv: 2006.11287. URL: https://arxiv.org/abs/2006.11287.

[2] J.L.E. Dreyer. *A History of Astronomy from Thales to Kepler*. 1953. URL: https://www.mathship.com/900/Astronomy/DreyerAHistoryOfAstronomyFromThalesToKepler1953.pdf.

[3] Georg Simon Ohm. *Die galvanische Kette*. 1827. URL: https://library.si.edu/digital-library/book/diegalvanischeke00ohmg.

[4] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. *Relational inductive biases, deep learning, and graph networks*. 2018. arXiv: 1806.01261 [cs.LG]. URL: https://arxiv.org/abs/1806.01261.

[5] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. "Graph Neural Networks: A Review of Methods and Applications". In: (2021). arXiv: 1812.08434 [cs.LG]. URL: https://arxiv.org/abs/1812.08434.

[6] *Discovering Symbolic Models from Deep Learning with Inductive Biases Code Repository*. URL: https://github.com/MilesCranmer/symbolic_deep_learning.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith

Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG]. URL: https://arxiv.org/abs/1912.01703.

[8] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. arXiv: 1903.02428 [cs.LG]. URL: https://arxiv.org/abs/1903.02428.

[9] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[10] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[11] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023. URL: http://udlbook.com.

[12] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[13] Miles Cranmer. *Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl*. arXiv:2305.01582 [astro-ph, physics:physics]. May 2023. DOI: 10.48550/arXiv.2305.01582. URL: http://arxiv.org/abs/2305.01582 (visited on 07/17/2023).

[14] *PySR Code Repository*. URL: https://github.com/MilesCranmer/PySR/blob/master/examples/pysr_demo.ipynb.

[15] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2018. arXiv: 1708.07120 [cs.LG]. URL: https://arxiv.org/abs/1708.07120.

[16] Leslie N. Smith. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*. 2018. arXiv: 1803.09820 [cs.LG]. URL: https://arxiv.org/abs/1803.09820.

[17] Michael Schmidt and Hod Lipson. "Distilling Free-Form Natural Laws from Experimental Data". In: *Science* 324.5923 (2009), pp. 81–85. DOI: 10.1126/science.1165893. eprint: https://www.science.org/doi/pdf/10.1126/science.1165893. URL: https://www.science.org/doi/abs/10.1126/science.1165893.

[18] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[19] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[20] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[21] Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. *Accelerate: Training and inference at scale made simple, efficient and adaptable.* https://github.com/huggingface/accelerate. 2022.