DIS project report; Word Count:

Yihao Liu; CRSid:yl2063

June 2025

Abstract

Contents

1 Introduction

Deriving closed-form analytical expressions for complex physical systems has long been a formidable challenge even when the underlying laws are simple. In recent years, deep learning has offered powerful "black-box" models that predict physical behavior with remarkable accuracy, but their internal representations often provide little indication to classical analytic equations and thus remain largely uninterpretable. Extracting interpretability from these black-box models is now a popular research area of both the Explainable AI and AI for Science communities.

Among deep learning methods, modern graph neural networks (GNNs) have demonstrated remarkable success in modeling complex many-body systems across physical sciences due to its strong inductive biases [1] and structural properties. These inductive priors, such as permutation invariance and local message-passing, align well with physical symmetries and conservation laws. As a result, GNNs not only achieve high predictive performance, but also provide a structured representation space that is particularly well-suited for interpretable deep learning.

At the same time, symbolic regression, as a technique of Interpretable AI or Explainable AI (XAI), offers a method to extract analytical equations from raw scientific data. Recent toolkits like PySR [2] apply evolutionary algorithms to search over a space of symbolic expressions, balancing complexity and accuracy. In addition, when applied to latent representations of a trained model, rather than raw data, symbolic regression can discover interpretable formulas that describe underlying physical laws.

Rediscovery project: Combination of GNN and symbolic regression This project reproduces and extends the work [3], which combines symbolic regression with GNNs to interpret the learned representations of physical systems. Specifically, for a trained GNN, they examined the linear relationship between latent messages and the ground-truth forces acting between particles by fitting a simple linear regression. They also showed that applying symbolic regression to the latent messages of the trained GNN can recover analytical expressions that describe the underlying dynamics of a simulated many-body system. By such a study, we aim to bridge the gap between black-box prediction and interpretable understanding of deep learning.

2 Background

In order to ground our reproduction and extensions, we first review the core theory of a GNN, illustrate its key applications in science and beyond, and summarize methods for Interpretable AI.

2.1 Theory of GNN

GNNs provide a principled framework for learning on graph-structured data by explicitly modeling interactions via edge messages. The key idea of applying GNN in this research consists of three core steps:

1. **Edge model or Message function:**

   For a graph with n nodes, let $x_i \in R^{d_x}$ denote the feature vector of node i, where d_x is a natural number indicating the number of node attributes (typically determined by the underlying data). The message dimension $D_{\text{msg}} \in N$ is a tunable hyperparameter.

   $$m_{ij} = \phi_{\text{edge}}\big(x_i,\, x_j,\, e_{ij}\big)\,, \tag{1}$$

   where $m_{ij} \in R^{D_{\text{msg}}}$ is the message passed from node i to node j, and ϕ_{edge} is a learnable MLP that takes the source feature $x_i \in R^{d_x}$, target feature $x_j \in R^{d_x}$, and optional edge attribute e_{ij} as input.

2. **Aggregation:**

The messages from neighboring nodes are aggregated using a permutation-invariant operator $\oplus$ (e.g., sum, mean, or max):

$$\bar{m}_j = \bigoplus_{i \in \mathcal{N}(j)} m_{ij} \,, \tag{2}$$

where $\bar{m}_j \in R^{D_{\text{msg}}}$ is the aggregated message vector at node j.

3. **Node model or Update function:**

   The updated feature vector x'_j for node j is computed by combining the original node feature x_j and the aggregated message $\bar{m}_j$:

   $$x'_j = \phi_{\text{node}}\big(x_j,\, \bar{m}_j\big) \,, \tag{3}$$

   where $x'_j \in R^{d'_x}$ is the new node embedding, and ϕ_{node} is another MLP. The output dimension d'_x is typically chosen based on the model and task design and can be different from d_x.

These design choices ensure GNNs with:

- *Permutation invariance*, since node ordering does not affect the results.
- *Parameter sharing*, as the same ϕ_{edge} and ϕ_{node} are used across all edges and nodes.

Such relational inductive biases naturally align with the symmetries and conservation laws of physical systems, making GNNs particularly well-suited for modeling many-body dynamics.

2.2 Graph Neural Network for science

Recently significant researches using GNN for natural sciences include `AlphaFold` [4], which is an example of successful combinations of interdisciplinary sciences and GNN-based deep learning. It is a system that predicts a protein's 3D structure from its amino-acid sequence with near-experimental accuracy. The motivation of designing this system is a combination of bioinformatics and physical inductive biases to learn from the data. For social sciences, GNN-based social recommender systems tackle the long-standing data sparsity and cold-start issues in traditional recommenders by unifying user–item interactions with user–user social ties into a single graph model [5]. The systems boost recommendation accuracy even for lightly connected users. GNNs are also used in many other domains, such as traffic forecasting [6] and healthcare [7], where the data exhibit strong structural properties.

2.3 Interpretable AI

Interpretability methods for explainable AI can generally be divided into four main categories [8]. The first focuses on explaining complex black-box models, such as deep neural networks. The second involves designing inherently transparent models, so-called white-box models, such as decision trees or linear models. The third category aims to ensure fairness by reducing or preventing discriminatory behavior in model predictions. Finally, the fourth type of method analyzes how sensitive a model's output is to changes in its input, helping to assess the model's robustness and stability.

Here we briefly point out two methods: *Post-hoc interpretability* provides external approximations or feature attributions after model training. *Neuro-symbolic* methods integrate deep learning with symbolic reasoning to extract concise and human-readable formulations. As a mixture of these two methods, symbolic regression techniques, such as PySR [2], offer the prospect of recovering closed-form mathematical expressions directly from data, thus providing interpretable laws that underpin empirical observations.

3 Methods

In this section, we discuss our reproduction methodology. First, we generate training and test data via JAX-based many-body simulations; next, we train various one-step message-passing GNNs on these trajectories to predict particle accelerations; finally, we extract learned edge-wise message representations and apply symbolic regression to recover human-readable force laws.

3.1 Analogy between Message–Passing GNN and N-Body Dynamics

To highlight why GNNs are particularly well-suited for learning many-body physical systems, we can draw an explicit analogy between each GNN operation and the corresponding step in Newtonian mechanics. Table 1 summarizes this correspondence.

Through this correspondence from computing edge messages to aggregating and updating node features, each step of the GNN mirrors the classical pipeline of computing

Table 1: Analogy between GNN components and many-body mechanics

GNN Component	**Physical Interpretation**
Nodes (v_i)	Particles with state vectors $(x_i, v_i, m_i, q_i, \dots)$.
Directed edges $(i \to j)$	Ordered interacting particle pairs (i, j).
Edge model $\phi^e(x_i, x_j)$	Computes pairwise interaction F_{ij} (e.g. spring, Coulomb, gravity).
Message e_{ij}	Latent "force" learned on edge $i \to j$, $e_{ij} = \phi^e(x_i, x_j)$.
Aggregation $\sum_{j \in \mathcal{N}(i)} e_{ij}$	Net force on particle i: $F_i = \sum_{j \neq i} F_{ij}$.
Node model $\phi^v(x_i, m_i)$	Updates acceleration $a_i = \phi^v(x_i, m_i) \approx F_i/m_i$.
Node update $x_i' = \phi^u(x_i, v_i)$	Integrates to next state.

pairwise forces, summing them to obtain net accelerations, and integrating forward in time.

This analogy suggests that the GNN's edge model and the physical force function perform roughly equivalent functional roles in their respective frameworks, motivating us to relate the black-box message parameters learned by the edge model to true force laws and thereby interpret them. However, we emphasize that this correspondence is intended purely as a conceptual guide for interpretable deep learning, rather than a precise one-to-one mapping. GNN's nodes and edges do not have to carry explicit physical meaning.

For example, although the underlying physical forces (e.g. Coulomb, gravity, spring) act symmetrically between particles, in our GNN implementation each undirected interaction (i, j) is represented by two *directed* edges $i \to j$ and $j \to i$. This is purely an implementation detail of the message-passing API [9]: we must distinguish a "sender" node (whose state is used to compute the message) from a "receiver" node (which aggregates incoming messages and updates its state). By instantiating both directions, we ensure that information can flow both ways and that each node both sends and receives messages in a consistent manner, while still preserving the overall symmetry of the physical interaction.

3.2 Compact Message Representations

Modern GNNs typically learn very high-dimensional edge representations ($m \in R^{D_{\text{msg}}}$) where high flexibility enables state-of-the-art predictive accuracy but obscures interpretability. A key insight from the original work [3] is that, for simple many-body force laws, the true inter-particle forces themselves lie in a much smaller message subspace. In this rediscovery research, we only study 2D physical systems, where $m \in R^{D_{\text{msg}}}, D_{\text{msg}} = 2$. A 2D force vector is supposed to span only a 2D message space. Therefore, the message space of our GNN models should be manually concentrated most of their physical content in the low-dimensional subspace.

3.2.1 Message-space constraints via GNN variants

To encourage exactly the low-dimensional representations, we compare four GNN models that differ only in how they constrain or regularize the edge-message space:

- **Standard GNN:** Uses a full and standard D_{msg} dimenisional message space with no implicit nor explicit regularization. All message channels are free to carry information.

- **Bottleneck GNN:** Directly restricts the message dimension to $D_{\text{msg}} = D$ (here $D = 2$), forcing the model to compress all inter-particle interactions into exactly two channels.

- **ℓ_1-regularized GNN:** Keeps D_{msg} large but adds an ℓ_1 penalty on each message channel's activations,
$$\mathcal{L}_{\ell_1} = \lambda \sum_e \|m_e\|_1 \tag{4}$$
encouraging most channels to go near zero and only a few to remain active.

- **Variational (KL-regularized) GNN:** Treats each edge message as a Gaussian latent variable and adds a KL divergence term to a fixed prior,
$$\mathcal{L}_{\text{KL}} = \beta \sum_e \text{KL}\big[q(m_e) \,\|\, \mathcal{N}(0, I)\big], \tag{5}$$
thereby penalizing excess information and effectively compressing the message distribution.

These four variants provides methods from no constraint (Standard) to tight, explicit compression (Bottleneck), with two soft-constraint approaches (ℓ_1, KL) in between. By inspecting how well each model's compressed message subspace aligns with the true force components, we elucidate the trade-off between predictive expressivity and physical interpretability.

3.2.2 Theoretical Explanation for Linear Relationship between Messages and True Forces

By following the original work [3], we now sketch why, under idealized "perfect prediction" conditions, each edge-message vector must lie in a linear subspace spanned by the true force vector, and hence can be recovered by ordinary least-squares.

1. GNN message–aggregate–update pipeline. For each directed edge $(i \to j)$, the *edge model* computes

$$e'_{i \leftarrow j} \;=\; \phi^{e}\big(x_i,\, x_j\big) \;\in\; R^{D_{\text{msg}}} \tag{6}$$

Node i then aggregates all incoming messages,

$$\bar{e}'_i \;=\; \sum_{j \neq i} e'_{i \leftarrow j} \tag{7}$$

and the *node model* predicts the acceleration via

$$\hat{a}_i \;=\; \phi^{v}\big(v_i,\, \bar{e}'_i\big) \;\approx\; a_i \;=\; \frac{1}{m_i} \sum_{j \neq i} F_{i \leftarrow j} \tag{8}$$

2. Perfect-prediction and local invertibility. Assume the trained GNN yields $\hat{a}_i = a_i$ exactly. If we manually choose $D_{\text{msg}} = 2$ to match the true force dimension, and furthermore $\phi^{v}(v_i, \cdot)$ is (locally) an invertible linear map for each fixed v_i, then

$$\bar{e}'_i \;=\; \big(\phi^{v}(v_i, \cdot)\big)^{-1} \Big(\tfrac{1}{m_i} \sum_{j \neq i} F_{i \leftarrow j}\Big) \;=\; A_i \sum_{j \neq i} F_{i \leftarrow j} \quad (\text{for some } A_i \in R^{2 \times 2}) \tag{9}$$

3. Single-edge degeneration. By the same argument applied to a graph with only one interacting neighbor, each individual message

$$e'_{i \leftarrow j} \;=\; A_i \, \frac{1}{m_i} F_{i \leftarrow j} \;=\; A_i \, F_{ij} \tag{10}$$

is a *linear* function of the true force F_{ij} on that edge.

4. Linear regression recovers the transform. Hence, when we collect all edges and their corresponding message activations e'_k and true forces F_k, fitting

$$e'_k \;\approx\; a\,F_k + b \tag{11}$$

via ordinary least-squares directly estimates the unknown transform $a \in R^{2\times 2}$ and offset $b \in R^2$.

This argument justifies our practice of selecting the top $D = 2$ message channels by variance and performing a separate linear regression of each channel against the two true force components.

3.3 Interpretation of Trained GNN

Once the various GNN models have been trained to predict particle accelerations, we seek to interpret its internal edge representations (the “messages”) in terms of known physical laws. We employ two complementary techniques: (1) linear regression to test whether a simple linear map from true forces to message channels exists as proved in Section, and (2) symbolic regression to search for closed-form analytic expressions that relate inter-particle geometry to message activations.

3.3.1 Linear Regression of Message Channels

We first extract each message channel m_k. The top D_{msg}=2 message channels by highest variance, matching the two spatial dimensions of our system, are supposed to be selected. Then, for each channel m_k separately, we fit a simple linear model

$$m_k \;\approx\; a_{k,x}\,F_x \;+\; a_{k,y}\,F_y \;+\; b_k,$$

where (F_x, F_y) are the true force components on each edge. Fitting via ordinary least-squares yields scalars $a_{k,x}, a_{k,y}, b_k$, and we report the coefficient of determination R_k^2 for each channel. High R_k^2 indicates that channel k has learned to encode a linear combination of the underlying physical force components.

3.3.2 Symbolic Regression of Message Channels

As the linear regression based technique potentially indicates linear relationship between message components, as ”black box” parameters, with the true physical forces, it provides a meaningful prospective of interpreting deep learning models. Furthermore, to uncover non-linear closed-form relations, we apply symbolic regression to each message channel m_k.

Symbolic regression is a data-driven technique for discovering explicit mathematical formulas that best describe a set of observations. Rather than fitting parameters to a pre-specified functional form, symbolic regression searches over a space of candidate expressions—built from variables, elementary functions (e.g. +, −, ×, ÷, ˆ, exp, log), and constants—to simultaneously identify both the structure and coefficients of the underlying law. Modern implementations typically employ genetic or evolutionary algorithms, assigning each candidate expression a fitness score that trades off accuracy against complexity, and iteratively recombine and mutate the fittest formulas to converge on compact, interpretable models [10] [2]. In our workflow, we apply symbolic regression to the learned edge-message activations of a trained GNN, seeking closed-form expressions that reveal the physical dependency on interparticle geometry.

The original study [3] used the *Eureqa* software [11] as its primary symbolic-regression tools. In our work, we instead employ their recently developed *PySR* package [2], which offers a highly optimized evolutionary search, dramatically improving throughput for high-dimensional physics data without sacrificing expression simplicity.

4 Experiments

In this section, we describe our experiments in details. We first simulate four canonical 2D N-body force laws to generate graph snapshots, which are then split training and test sets and make them loaded. Next, we train and compare four GNN variants on the task of predicting particle accelerations. Finally, we extract the learned edge-message embeddings and apply both linear and symbolic regression to evaluate how faithfully they recover the true force laws using the methods described in Methods.

We largely follow the original work's implementation and pipeline [3]. We provide a full discussion of any differences or hyperparameter tuning, including their motivations, in Appendix.

4.1 Physical System Simulation

We study four canonical 2D N-body force laws:

- **Mass–spring force** (Hooke's law): each connected pair of nodes exerts a linear spring force
$$\mathbf{F}_{ij} = k\,(r_{ij} - r_0)\,\hat{\mathbf{r}}_{ij}\,, \quad k = 1,\ r_0 = 1, \tag{12}$$
directed along the unit vector $\hat{\mathbf{r}}_{ij} = (\mathbf{x}_j - \mathbf{x}_i)/r_{ij}$.

- **Inverse-square gravitational force**: every unordered pair interacts via

$$\mathbf{F}_{ij} = \frac{G\, m_i\, m_j}{r_{ij}^2}\,\hat{\mathbf{r}}_{ij}\,, \quad G = 1 \tag{13}$$

- **Inverse-square Coulomb force**: identical form to gravity but with signed charges,

$$\mathbf{F}_{ij} = \frac{q_i\, q_j}{r_{ij}^2}\,\hat{\mathbf{r}}_{ij} \tag{14}$$

- **Inverse-distance force**: a toy "$1/r$" law on every pair,

$$\mathbf{F}_{ij} = \frac{G\, m_i\, m_j}{r_{ij}}\,\hat{\mathbf{r}}_{ij}\,, \quad G = 1 \tag{15}$$

4.2 Data Generation

We simulate 10,000 independent systems for each physical system, each consisting of $N = 4$ particles in an unbounded 2D domain. For each system:

- *Initial conditions*: positions $\mathbf{x}_i(0)$ and velocities $\mathbf{v}_i(0)$ are sampled uniformly in $[-1, 1]^2$. Masses $m_i \sim \mathcal{U}(0, 1)$ and charges $q_i \sim \mathrm{Unif}\{-1, +1\}$ are drawn independently for each particle.

- *Integration*: we integrate Newton's equations

$$m_i\,\ddot{\mathbf{x}}_i \;=\; \sum_{j \neq i} \mathbf{F}_{ij} \tag{16}$$

  using a symplectic Euler scheme with system-specific time steps and a fixed number of steps:

$$\Delta t = \begin{cases} 1 \times 10^{-2}, & \text{spring,} \\ 5 \times 10^{-3}, & 1/r \text{ force,} \\ 1 \times 10^{-3}, & 1/r^2 \text{ forces (gravitational \& Coulomb),} \end{cases} \qquad n_t = 500.$$

- *Edge topology*: For all forces we use the complete graph (every pair interacts).

- *Recorded features*: at each time step we record node-feature vectors

$$x_i = \big(x_i,\, y_i,\, v_{x,i},\, v_{y,i},\, q_i,\, m_i\big)$$

4.3 Data Splitting and Dataloader Construction

The raw simulation output consists of 10 000 independent trajectories, each recorded over $n_t = 500$ timesteps for $N = 4$ particles with $F = 6$ node features $(x, y, v_x, v_y, \text{charge}, \text{mass})$. Hence the feature array X has shape

$$(10\,000,\ 500,\ 4,\ 6),$$

and the corresponding acceleration array Y has shape

$$(10\,000,\ 500,\ 4,\ 2).$$

To reduce temporal redundancy, we keep every 5th frame ($\frac{500}{5} = 100$ snapshots per trajectory) and then collapse the trajectory and time axes into a single flattened dimension along the trajectory (sample) dimensionunlike the original implementation in [3]. The reason behind this modification is provided in Appendix 6.

This yields a total of

$$10\,000 \times 100 \ = \ 1\,000\,000$$

graphs. We then partition these graphs into a 75%/25% train/test split, resulting in

- **Training set:** 750 000 snapshots of shape $(4, 6)$ for X and $(4, 2)$ for Y.
- **Test set:** 250 000 snapshots of the same per-snapshot shapes.

Each snapshot pairs the node-feature matrix $\mathbf{x} \in R^{4\times6}$ with its edge-index specification (fixed for all graphs) and the target acceleration matrix $\mathbf{y} \in R^{4\times2}$. These samples are then organized into data objects and batched using `PyTorch`'s data loading utilities to enable efficient training and evaluation of the GNN. For the training loader, we set `shuffle=True` so that each epoch sees a randomized ordering of the snapshots, which helps prevent overfitting and yields more robust gradient estimates. The training batch size is set as 64 to balance per-batch throughput against GPU memory constraints. For the test loader, we randomly select only 1,024 snapshots out of the 250,000 test samples with one single batch to ensure a deterministic, one-pass evaluation on the held-out set.

4.4 Models

In this section, we describe the various GNN architectures, their constituent components, key hyperparameters, and the overall training procedure employed in this study.

Our models are written under the framework of PyTorch [12] and PyTorch Geometric [9].

4.4.1 Common Architecture and Hyperparameters

All of our models employ the similar single-step message-passing GNN architecture, differing in message dimensionality, regularization and loss scale. The shared hyperparameters and architecture are:

- **Hidden dimension** $h = 300$: each node MLP and edge MLP has two hidden layers of width 300.
- **Message dimension** $D_{\text{msg}} = 100$ (standard): the edge model outputs a 100-dimensional message space per directed edge.
- **Edge MLP**: a three-layer feed-forward network mapping each concatenated source–target feature vector of size $2n_f$ (where $n_f = 6$ for $(x, y, v_x, v_y, charge, mass)$) to a D_{msg}–dimensional message.

  $$\text{Lin}(2n_f,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, D_{\text{msg}})\,. \tag{17}$$

- **Aggregation & Node update**: we sum all incoming messages, concatenate the sum with the node's current state, and pass through the one-layer MLP to compute its new acceleration.
- **Node MLP**: after summing all incoming messages $m_{ij} \in R^{D_{\text{msg}}}$, we concatenate the result with the node's own feature vector of size n_f, yielding input of size $D_{\text{msg}} + n_f$. We then apply

  $$\text{Lin}(D_{\text{msg}}{+}n_f,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, n_{\text{out}}) \tag{18}$$

  where $n_{\text{out}} = 2$ (the predicted (a_x, a_y) per particle).
- **Loss**: mean absolute error (MAE) on predicted accelerations, with L_2 weight-decay (equivalent to L_2 regularization with coefficient $\lambda = 10^{-8}$) applied to all model parameters via an Adam optimizer. Let B be the batch size (number of graphs) and N the number of nodes per graph. Then the loss is

  $$\mathcal{L} = \frac{1}{B}\sum_{b=1}^{B}\sum_{i=1}^{N}\left\|\hat{\mathbf{a}}_i^{(b)} - \mathbf{a}_i^{(b)}\right\|_1 \;+\; \lambda\,\|\boldsymbol{\theta}\|_2^2, \quad \lambda = 10^{-8}\,,$$

where $\boldsymbol{\theta}$ denotes all learnable parameters of the GNN.

4.4.2 Model Variants and Motivation

We compare four variants, as mentioned above, to study how different constraints affect message interpretability. Detailed implementations are provided here:

1. **Standard GNN** ($D_{\text{msg}} = 100$), no extra regularization). Serves as a high-capacity baseline.

2. **Bottleneck GNN** ($D_{\text{msg}} = 2$), no additional loss). This is an implicit regularization.

3. L_1**-regularized GNN** ($D_{\text{msg}} = 100$), $\lambda = 0.02$). Adds an L_1 penalty on every message vector $\mathbf{m}_{ij}$:
$$\mathcal{L} = \mathcal{L}_{\text{MAE}} \;+\; 0.02 \sum_{i,j} \|\mathbf{m}_{ij}\|_1 \,,$$
encouraging sparse channel usage.

4. **Variational GNN (KL)** ($D_{\text{msg}} = 100$).

   In the KL model, we treat each edge message as a random vector rather than a deterministic embedding. Concretely, the edge network ϕ^e now outputs $2D_{\text{msg}}$ features for each directed edge k:
$$\mu'_k \;=\; \phi^e_{1:D_{\text{msg}}}(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}), \quad \log \sigma'^2_k \;=\; \phi^e_{D_{\text{msg}}:2D_{\text{msg}}}(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}),$$
   and we sample
$$\mathbf{e}'_k \;\sim\; \mathcal{N}\big(\mu'_k,\; \text{diag}(\sigma'^2_k)\big).$$
   The sampled messages are then aggregated at each target node i,
$$\bar{\mathbf{e}}'_i \;=\; \sum_{k:r_k=i} \mathbf{e}'_k, \quad \hat{\mathbf{v}}'_i \;=\; \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i),$$
   to predict the next-step acceleration/velocity.

   The training objective combines a reconstruction term (MAE) on the predicted accelerations with a Kullback–Leibler divergence term that regularizes each message distribution toward the standard normal prior $\mathcal{N}(0, I)$:
$$\mathcal{L}_{\text{MAE}} = \frac{1}{N_{\text{particles}}} \sum_{b=1}^{B} \sum_{i=1}^{N} \big\|\hat{\mathbf{a}}_i^{(b)} - \mathbf{a}_i^{(b)}\big\|_1,$$

$$\mathcal{L}_{\mathrm{KL}} = \frac{1}{N^e} \sum_{k=1}^{N^e} \sum_{j=1}^{D_{\mathrm{msg}}} \frac{1}{2} \Big(\mu'^2_{k,j} + \sigma'^2_{k,j} - \log \sigma'^2_{k,j} - 1 \Big).$$

Following the original work, we set the KL weight $\alpha_1 = 1$ (i.e. standard VAE) so that uninformative message dimensions naturally revert to $(\mu = 0, \sigma = 1)$.

Moreover, through hyperparameter tuning we found that scaling the MAE by the total number of particle-level predictions (rather than by full graph or batch size) yields more stable convergence. The overall loss is:

$$\mathcal{L}_{\mathrm{total}} = \mathcal{L}_{\mathrm{MAE}} + \mathcal{L}_{\mathrm{KL}}.$$

B**:** Batch size, the number of graphs processed in one training step.

N**:** Number of particles (nodes) per graph.

$N_{\mathrm{particles}} = B \times N$**:** Total number of particle predictions per batch, used to normalize the MAE.

N^e**:** Total number of directed edges per batch. In a fully connected directed graph, $N^e = B \times N \times (N - 1)$.

$\hat{\mathbf{a}}_i^{(b)}$, $\mathbf{a}_i^{(b)}$**:** Predicted and true acceleration of the i-th particle in the b-th graph.

$\mu'_{k,j}$, $\sigma'^2_{k,j}$**:** Mean and variance of the j-th component of the message distribution on the k-th edge.

Each variant trades off expressive capacity against interpretability: reducing D_{msg}, enforcing sparsity, or imposing a prior can make the learned messages more amenable to symbolic analysis, at the potential cost of predictive accuracy.

4.4.3 Model Training Pipeline

All models are trained for 30 complete passes (30 epochs) over the training set. We optimize using the Adam [13] algorithm with an initial learning rate of 1×10^{3} and an L weight-decay coefficient of 1×10 applied to every parameter. To modulate the learning rate we employ OneCycleLR scheduler [14]: it ramps up from a very small value to the peak (1×10^{3}) and back down over exactly one epoch, repeating this cycle for each of the 30 epochs. Since our training loader contains 11 719 mini-batches, this schedule executes a total of $30 \times 11\ 719 = 351\ 570$ update steps.

Within each step, we first clear accumulated gradients, then perform a forward pass to predict particle accelerations, compute the MAE loss (plus the L penalty), backpropagate, and finally advance both the optimizer and the learning-rate scheduler. By exposing the models to every mini-batch in each epoch and carefully annealing the learning rate, we ensure stable convergence across all four GNN variants.

4.5 Latent Message Extraction

After training, we freeze the GNN weights and run inference on the held-out test set. For each directed edge $(i \to j)$ in each graph, we record:

1. *Node features* x_i and x_j, each comprising

$$(x, y;\ v_x, v_y;\ q;\ m)\,,$$

   i.e. the particle's 2D position, 2D velocity, scalar charge and scalar mass.

2. *Learned message vector*

$$m_{ij}\ \in\ R^{D_{\text{msg}}}\,,$$

   where

$$D_{\text{msg}} = \begin{cases} 100 & \text{(standard model)} \\ 2 & \text{(bottleneck model)} \\ 100 & \text{(L}_1\text{-regularized model)} \\ 200 & \text{(KL-divergence model)} \end{cases}$$

3. *Geometry features*: the relative displacement

$$\Delta x = x_i - x_j, \quad \Delta y = y_i - y_j, \quad r = \sqrt{\Delta x^2 + \Delta y^2}\,.$$

   In downstream analysis we combine $(\Delta x, \Delta y, r)$ with (q_i, q_j, m_i, m_j) to compute the true physical force components.

All of these quantities are concatenated into a single `pandas.DataFrame`, with one row per directed edge. Since each graph has $N = 4$ particles and thus $N(N-1) = 12$ directed edges, and there are 250 000 test graphs, the resulting frame has

$$250\,000 \times 12\ =\ 3\,000\,000$$

rows. Each row contains

$$\underbrace{2 \times 6}_{\substack{\text{source/target}\\ \text{node features}}} \quad + \quad \underbrace{D_{\text{msg}}}_{\text{message channels}} \quad + \quad \underbrace{3}_{(\Delta x, \Delta y, r)} \quad \text{columns in total.}$$

4.6 Analytic Function Recovery

Recall the 2D force laws 12 13 14 15. Writing $\Delta x = x_j - x_i$, $\Delta y = y_j - y_i$ and $r = \sqrt{\Delta x^2 + \Delta y^2}$, each law takes the form

$$\mathbf{F}_{ij} = F(r)\,\frac{(\Delta x,\ \Delta y)}{r}\,,$$

with the following specific choices of the scalar magnitude $F(r)$:

$$\begin{aligned}
&\textbf{(1) Mass–spring (Hooke's law):} && F(r) = k\,(r - r_0) \quad (k = 1,\, r_0 = 1),\\
& && \Longrightarrow \begin{cases} F_x = (r - r_0)\,\dfrac{\Delta x}{r}, \\ F_y = (r - r_0)\,\dfrac{\Delta y}{r}. \end{cases}\\
&\textbf{(2) Inverse-square gravitational:} && F(r) = \frac{G\,m_i\,m_j}{r^2} \quad (G = 1),\\
& && \Longrightarrow \begin{cases} F_x = m_i\,m_j\,\dfrac{\Delta x}{r^3}, \\ F_y = m_i\,m_j\,\dfrac{\Delta y}{r^3}. \end{cases}\\
&\textbf{(3) Inverse-square Coulomb:} && F(r) = \frac{q_i\,q_j}{r^2},\\
& && \Longrightarrow \begin{cases} F_x = q_i\,q_j\,\dfrac{\Delta x}{r^3}, \\ F_y = q_i\,q_j\,\dfrac{\Delta y}{r^3}. \end{cases}\\
&\textbf{(4) Inverse-distance “toy” } 1/r\textbf{:} && F(r) = \frac{k}{r} \quad (k = 1),\\
& && \Longrightarrow \begin{cases} F_x = \dfrac{\Delta x}{r^2}, \\ F_y = \dfrac{\Delta y}{r^2}. \end{cases}
\end{aligned}$$

After extracting the GNN's edge-message channels, we first compute the within-channel variance for each message channel and select the top 2 channels (those with the highest variance, as they empirically carry the most information) for further analysis. We then apply two complementary interpretation methods:

1. Linear-combination fitting Recall the linear model defined in 3.3.1: For each channel m_k, we fit a simple linear model

$$m_k \;\approx\; a_{k,x}\,F_x \;+\; a_{k,y}\,F_y \;+\; b_k,$$

we evaluate its predictive power on held-out edges by feeding the true force components (F_x, F_y) into the learned model to produce

$$\hat{m}_{k,i} = a_{k,x}\,F_{x,i} + a_{k,y}\,F_{y,i} + b_k \quad \text{for each edge } i.$$

We then compute the coefficient of determination

$$R_k^2 = 1 - \frac{\sum_i \bigl(m_{k,i} - \hat{m}_{k,i}\bigr)^2}{\sum_i \bigl(m_{k,i} - \bar{m}_k\bigr)^2}\,, \quad \bar{m}_k = \frac{1}{N}\sum_i m_{k,i}\,,$$

which quantifies how much of the variance in the recorded message m_k is explained by its linear fit from (F_x, F_y).

- $R_k^2 \approx 1$ implies that channel k is nearly a perfect linear encoding of the force components.
- $R_k^2 \approx 0$ indicates almost no linear relationship.
- Intermediate values (e.g. $R_k^2 \in [0.3, 0.5]$) suggest only partial alignment – for instance, that the channel may predominantly capture a single force dimension or that a nonlinear mapping would explain more variance.

Numerical Stabilization To avoid potential division-by-zero errors when two particles coincide, we add a small constant ε to r in the denominator. Specifically, we use $\varepsilon = 10^{-6}$ for the Spring, Inverse-distance, and Inverse-square systems, and $\varepsilon = 10^{-5}$ for the Coulomb system. These values were chosen by a manual parameter tuning to balance numerical stability with negligible perturbation to the true force magnitudes.

2. Symbolic regression (SR) To discover explicit closed-form laws in each message channel, we employ the high-performance *PySR* package, which runs evolutionary symbolic regression fast. Following the original study [3], we restrict our search to the top-variance channel. We draw a random subset of 5,000 edges to speed up training and run PySR for 100 iterations, which is the default. Initially, we allow only the four basic binary operators $\{+,\,-,\,\times,\,/\}$, which means that there is exactly no functional prior provided to the model; more complex operators (e.g. `Cube`) can be added later if needed.

PySR evaluates each candidate by trading off simplicity (expression complexity) against accuracy (mean squared error).

5 Results and Analysis

In this section, we report results for all 16 experiment configurations, combining four GNN variants with four physical N-body force laws. We begin by summarizing each model's training convergence and acceleration-prediction performance on the held-out test set. We then evaluate the alignment between the learned message channels and the true force components via linear-combination fitting. Finally, we present the outcomes of our symbolic regression (PySR) experiments, comparing the recovered closed-form expressions against the ground-truth force laws.

5.1 Model Performance

5.1.1 Training Loss

In Figure 2 we show the training loss curves for the four physical systems. Both the spring and the inverse-distance setups exhibit nearly identical convergence profiles and reach the lowest loss values, reflecting their relatively simple linear and $1/r$ force laws. By contrast, the Coulomb system exhibits a substantially higher loss, and the inverse-square gravitational system is the highest. This ordering aligns with increasing functional complexity: the spring and inverse-distance laws depend only on r, whereas the Coulomb model adds discrete ± 1 charges to the $1/r^2$ kernel, and the gravitational law combines continuous random masses with a singular $1/r^2$ dependence, making it the hardest for the GNN to fit.

Within each physical system the three model variants (Standard, Bottleneck, ℓ_1-regularized) show similar training-loss trajectories. This indicates that constraining the message dimension or adding sparsity/KL penalties does not significantly affect convergence speed or final training loss under our one-step message-passing scheme.

The KL-regularized models use a different overall loss scale (due to the added KL term), so we plot their curves separately in the bottom panel of Figure 2. Nevertheless, their convergence trends mirror those of the other three variants across all four systems.

These curves are not necessarily guaranteed for full convergence. Our priority in this research is on interpretability across 16 model–system combinations rather than squeezing out the last drop of predictive accuracy. A full description of the number of training steps, as a hyperparameter of model performance, will be provided in Appendix.

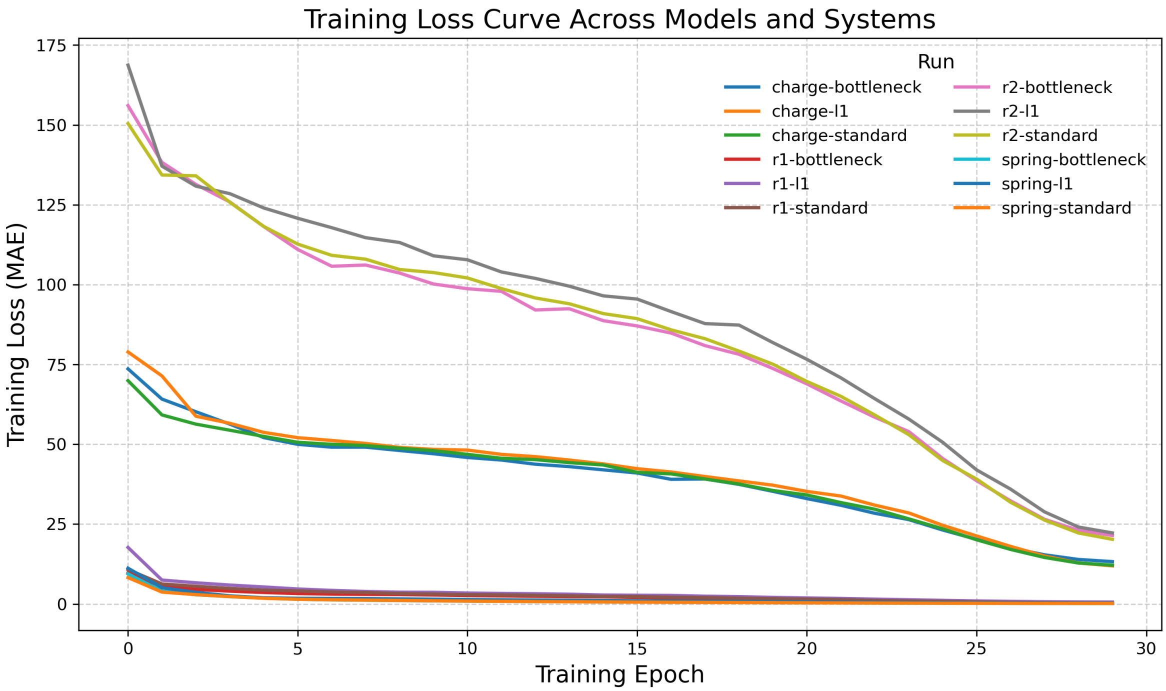

Figure 1: Training loss curve across all model variants (except KL-regularized) and all systems.

5.1.2 Test-set MAE

For completeness, Table 2 reports the particle-level MAE on the test set, i.e. the mean absolute error averaged over individual particles (as opposed to the batch-level MAE used during training). Despite this difference, the ranking of test MAE across the four physical systems—lowest for the spring and $1/r$ laws, higher for Coulomb, and highest for the $1/r^2$ law—closely matches the trends seen in training loss. Since acceleration prediction is not our primary focus, these MAE results are provided for reference only and will not be discussed in depth.

Table 2: Test MAE on the held-out set for each physical system and model variant.

System	Standard	Bottleneck	ℓ_1	KL
Spring	0.024	0.021	0.027	1.445
Coulomb (Charge)	2.619	3.266	0.403	2.860
Inverse-square ($1/r^2$)	8.945	9.268	8.230	12.415
Inverse-distance ($1/r$)	0.040	0.034	0.068	1.995

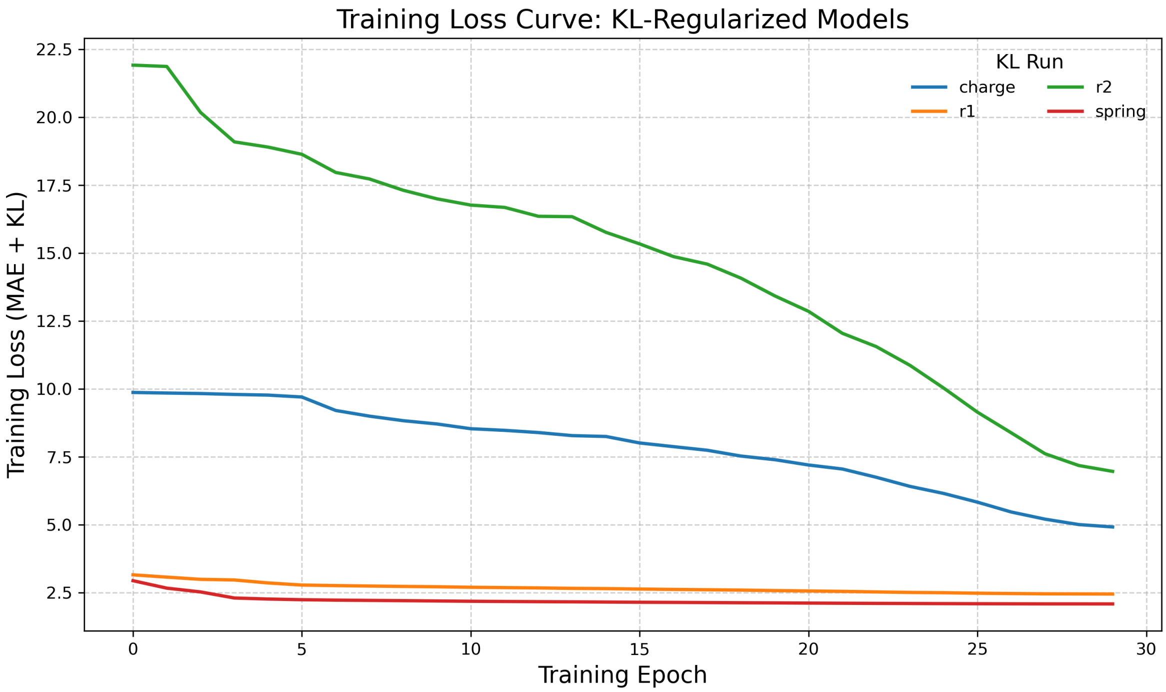

Figure 2: Training loss curve of KL-regularized models and all systems.

5.2 Linear-Combination Fitting

We now evaluate the linear relationship between the GNN's learned edge-message channels and the true force components. We use the Spring system as a detailed case study, and then highlight the most striking findings for the remaining three force laws.

5.2.1 Spring System

Standard Model Figure 3 displays the scatter plots of the two highest-variance message channels from the Standard GNN variant against their best linear combinations of true force components. Remarkably, even without any explicit bottleneck or sparsity constraint, both channels exhibit a clear linear trend ($R^2 = 0.343$ and 0.337, respectively). This indicates that the trained GNN's message activations naturally exhibit a small degree of linear correlation with the underlying simulated spring forces.

Bottleneck Model Figure 4 shows the two message channels from the Bottleneck GNN (with $D_{\text{msg}} = 2$) plotted against their best linear combinations of true spring forces. Both channels achieve near-perfect fit ($R^2 = 0.997$ and 0.998), with almost all points clustering tightly along the $y = x$ reference line.

Compared to the Standard GNN (Figure 3), the Bottleneck variant achieves nearly perfect linear relationship between message channels and true forces. This dramatic

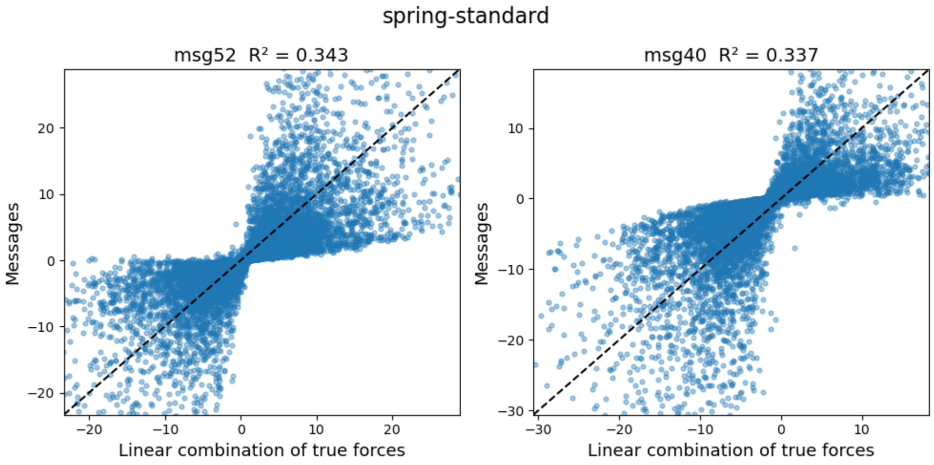

Figure 3: Spring system – Standard GNN variant. The dashed line shows $y = x$.

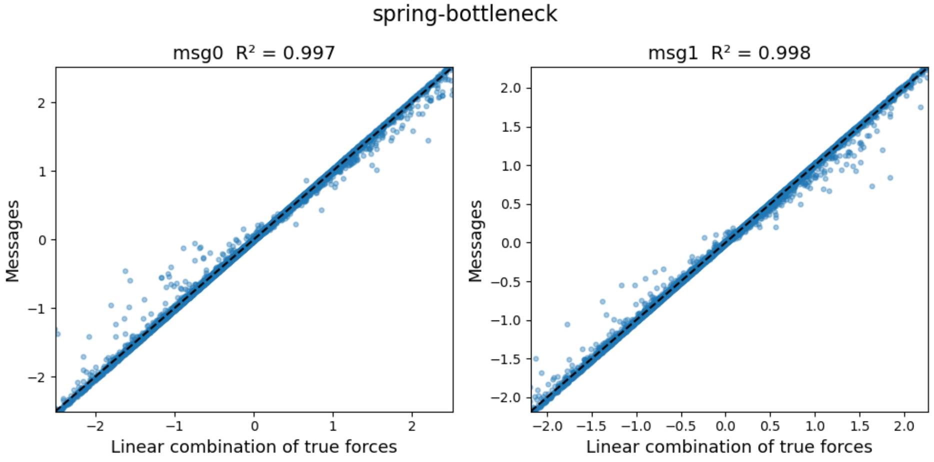

Figure 4: Spring system – Bottleneck GNN variant.

improvement confirms our theoretical claim that fixing the message dimension to the actual force dimensionality, specifically by enforcing a two-channel bottleneck, is highly effective, where the GNN has to encode the full interaction information into exactly those two channels.

L_1-regularized Model Having seen the extreme case of an explicit bottleneck, we next examine softer sparsity constraints. Figure 5 presents the results for two highest-variance message channels from the L_1-regularized GNN. Both channels exhibit stronger linear alignment than the unconstrained Standard model, but the linear relationship is not as strong as the Bottleneck case. This improvement demonstrates that adding an ℓ_1 penalty encourages the model to concentrate physical information into fewer active channels, yielding message activations that more closely track the underlying force subspace.

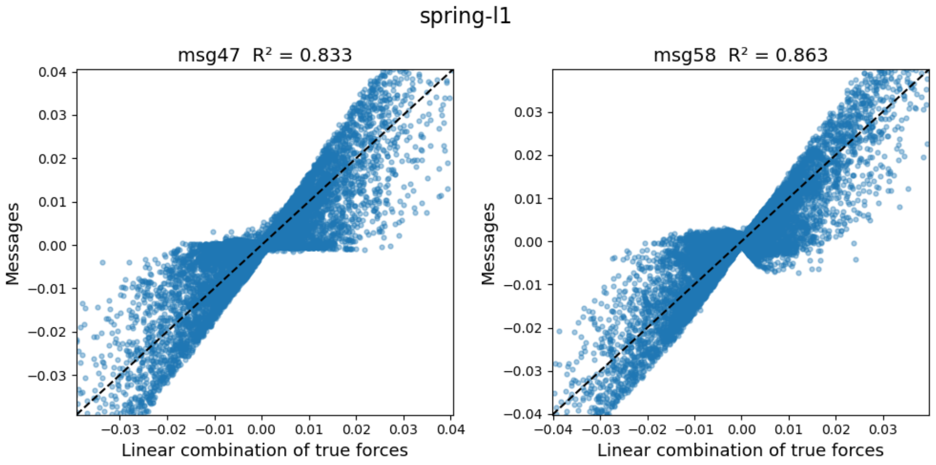

Figure 5: Spring system – L_1-regularized GNN.

KL-regularized Model Figure 6 shows results for the KL-regularized GNN. Both channels exhibit moderate linear alignment. Beyond the linear relationship, it is worth mentioning that unlike the other variants, an obvious fraction of message activations collapse to zero.

This zero-mass of points at $m = 0$ arises from the KL penalty: any message dimension that does not carry sufficient predictive information is driven toward the Gaussian prior $\mathcal{N}(0, 1)$, whose mean is zero. As a result, uninformative channels become effectively "switched off," leaving only the most salient subspace to encode the force law. This behavior confirms the KL model's tendency to sparsify its representation by pruning weak channels, albeit at the cost of somewhat reduced linear fit compared to the L_1 and Bottleneck variants.

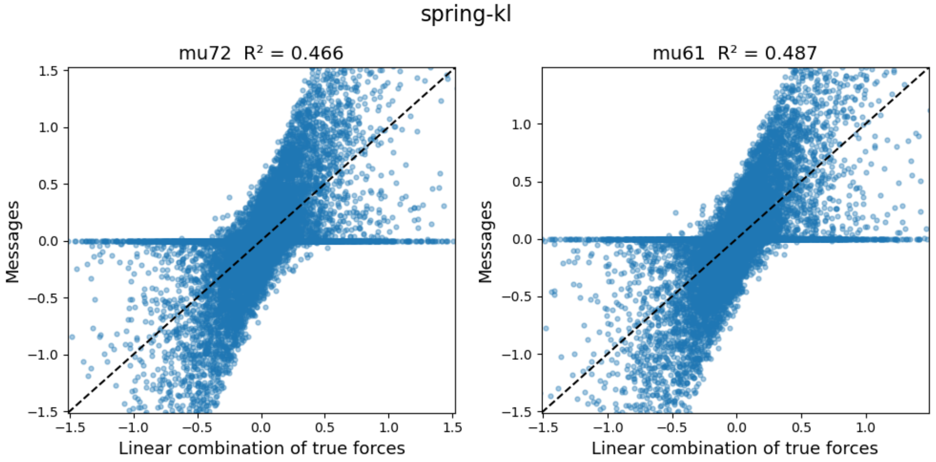

Figure 6: Spring system – KL-regularized GNN. Note the The KL penalty drives many messages to zero mean.

Message-Space Scaling and Normalization In the original work [3], all extracted message channels were standardized to zero mean and unit variance before any downstream analysis. We deliberately do not rescale or normalize the learned messages, so that each model's native output range is preserved.

Under this scheme, the four GNN variants exhibit markedly different activation scales:

- **Standard GNN:** With $D_{\text{msg}} = 100$ unconstrained channels and no activation penalty, messages freely grow to minimize MAE, often spanning a large range (e.g. $[-20, 20]$) determined solely by the MLP weights and training dynamics.
- **Bottleneck GNN:** By fixing $D_{\text{msg}} = 2$, the network must compress all pairwise information into two dimensions, yielding much tighter activations (e.g. $[-2, 2]$) that directly align with the true force components.
- **ℓ_1-regularized GNN:** Adding an ℓ_1 penalty on each message's absolute value drives most channels toward zero, so that only a handful of channels remain active in a very narrow band around zero (e.g. ± 0.01).
- **KL-regularized GNN:** Treating messages as Gaussian latents with a $\mathrm{KL}(q \| \mathcal{N}(0, I))$ term pushes uninformative channels to the zero-mean prior, producing many exactly zero activations and confining the remaining ones roughly within $[-1.5, 1.5]$.

This discussion provides a confirmation on how each constraint strategy, including dimensional, sparsity, or probabilistic, implicitly calibrates the magnitude of the learned physical kernel.

5.2.2 Other Systems

We now turn to the remaining results of other systems, as displayed in Table 3. We select the most illustrative message channels and discuss the key behaviors.

System	**Standard**	**Bottleneck**	ℓ_1	**KL**
Spring	0.343, 0.337	0.997, 0.998	0.833, 0.863	0.466, 0.487
Coulomb	0.011, 0.033	0.789, 0.140	0.480, 0.684	0.589, 0.543
Inverse-square	0.020, 0.043	0.668, 0.525	0.201, 0.206	0.163, 0.162
Inverse-distance	0.431, 0.425	0.409, 0.530	0.470, 0.508	0.391, 0.302

Table 3: Linear-combination fitting R^2 values for the two highest-variance message channels, across four GNN variants and four classical force laws.

Intrinsic Capture by Unconstrained GNNs Even without any explicit constraints, the Standard GNN already achieves relatively high R^2 on the Spring and Inverse-distance systems. This indicates that these two force laws, being the simplest natural laws, are already partially interpretable by a high-capacity, unconstrained message space.

Channel-Specific Information Concentration In some of the Coulomb and Inverse-square systems, we observe large discrepancies between the two top-variance channels (e.g. Bottleneck Coulomb: 0.789 vs. 0.140) within the same model. Such imbalances suggest that most of the physical kernel is possibly encoded in a single dominant channel, while the second channel carries only residual or noise-level information.

Constraint Ranking and Irreproducibility In contrast to the original study's claim that the ℓ_1-regularized variant yields the best interpretability [3], our results consistently follow the ordering

$$\text{Bottleneck} > \ell_1 > \text{KL} > \text{Standard}$$

in terms of linear-fit quality. This mismatch underscores a reproducibility gap, even with the same conceptual and similar experimental pipeline.

Resistance of Inverse-Distance Law For the $1/r$ force, all four GNN variants achieve similar R^2 values, indicating that neither dimensional bottlenecks nor sparsity priors yield marked gains. The Inverse-distance kernel thus appears inherently less amenable to further compression beyond the unconstrained baseline.

Continuous Results vs. Extreme Results Overall, we report a continuum of mid-range R^2 values (0.2–0.8) across many systems and models, in contrast to the original paper's extreme "near-0" or "near-1" fits [3].

5.2.3 Overall Reproducibility of Linear-Fit Analysis

In general, our experiments confirm that the core idea of applying a compact internal representation via constrained message dimensions, as a type of strong inductive biases, is useful and successful for making the message components of a trained GNN to contain physical information. However, we also observe that finer details, including whether the Bottleneck or ℓ_1 variant consistently outperforms the other, and the presence of extreme near-zero or near-one R^2 values reported in the original work, do not replicate perfectly.

5.3 Symbolic Regression Outcomes

We now provide and analyze results of applying PySR-based symbolic regression to the GNN's highest-variance message channels.

References

[1] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," 2018.

[2] M. Cranmer, "Interpretable machine learning for science with pysr and symbolicregression.jl," 2023.

[3] M. Cranmer, A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, "Discovering symbolic models from deep learning with inductive biases," 2020.

[4] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[5] K. Sharma, Y.-C. Lee, S. Nambi, A. Salian, S. Shah, S.-W. Kim, and S. Kumar, "A survey of graph neural networks for social recommender systems," vol. 56, June 2024.

[6] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert Systems with Applications*, vol. 207, p. 117921, 2022.

[7] S. G. Paul, A. Saha, M. Z. Hasan, S. R. H. Noori, and A. Moustafa, "A systematic review of graph neural network in healthcare-based applications: Recent advances, trends, and future directions," *IEEE Access*, vol. 12, pp. 15145–15170, 2024.

[8] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, “Explainable AI: A review of machine learning interpretability methods,” *Entropy*, vol. 23, no. 1, p. 18, 2021.

[9] M. Fey and J. E. Lenssen, “Fast graph representation learning with pytorch geometric,” 2019.

[10] N. Makke and S. Chawla, “Interpretable scientific discovery with symbolic regression: a review,” *Artificial Intelligence Review*, vol. 57, Jan. 2024.

[11] M. Schmidt and H. Lipson, “Distilling free-form natural laws from experimental data,” *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” 2019.

[13] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

[14] L. N. Smith and N. Topin, “Super-convergence: Very fast training of neural networks using large learning rates,” 2018.

6 Appendix