Symbolic Distillation of Neural Networks

Supervised by Dr Miles Cranmer

Word Count: 6990 words

Yihao Liu; CRSid:yl2063

June 2025

Abstract

We reproduce and extend the framework of [1], which combines Graph Neural Networks (GNNs) with symbolic regression to recover analytical force laws from learned message representations. It is also an alternative to interpret deep neural networks. Specifically, by studying four 2D many-body systems and four GNN variants, we show that constraining the edge-message space yields representations that align linearly to ground-true physical laws and produce closed-form expressions by symbolic regression. We also provide two extensions illustrating that the in-depth mechanism of how message channels encode physical information is not fully transparent. Our results confirm that the original work [1] offer a largely reproducible route to interpretable deep learning in scientific research.

Contents

1 Introduction

Deriving closed-form analytic 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 (XAI) 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 [2] 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 or Explainable AI, offers a method to extract analytical equations from raw scientific data. Recent toolkits like PySR [3] 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 [1], 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 analytic 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

A GNN processes a graph $G = (V, E)$ by passing learned messages along its edges and updating node features accordingly [2]. Let each node $i \in V$ have an initial feature vector, where d_x is a natural number indicating the number of node attributes (typically determined by the underlying data):

$$x_i^{(0)} \in \mathbb{R}^{d_x}\,, \tag{1}$$

and each edge $(i, j) \in E$ an attribute e_{ij}. The core of one *message-passing* step t consists of three stages:

1. **Edge model (message function):**

$$m_{ij}^{(t)} \;=\; \phi_{\text{edge}}\big(x_i^{(t)},\, x_j^{(t)},\, e_{ij}\big) \;\in\; \mathbb{R}^{D_{\text{msg}}}. \tag{2}$$

   Here D_{msg} is the message dimension, which is the number of activation channels output by the edge MLP. A message channel is the collection of the component from the same dimension taken from every edge's message embedding across the entire set of edges.

2. **Aggregation:**

$$\tilde{m}_j^{(t)} \;=\; \bigoplus_{i \in \mathcal{N}(j)} m_{ij}^{(t)}, \tag{3}$$

   where $\oplus$ is a permutation-invariant reducer (e.g. sum or mean) over the neighborhood $\mathcal{N}(j)$ of node j.

3. **Node model (update function):**

$$x_j^{(t+1)} \;=\; \phi_{\text{node}}\big(x_j^{(t)},\, \tilde{m}_j^{(t)}\big). \tag{4}$$

   The updated vector $x_j^{(t+1)} \in \mathbb{R}^{d_x}$ then serves as input for the next step.

Single-Step Message Passing (T=1) In this work, we use exactly one message-passing layer ($T = 1$). That is, each node's next-feature $x_j^{(1)}$ depends only on its immediate neighbors in the graph, rather than multi-hop interactions. Constraining $T = 1$ simplifies our analysis: the only learned latent variables of interest are the edge messages $m_{ij}^{(0)} \in \mathbb{R}^{D_{\text{msg}}}$ produced by the initial edge model.

Inductive Biases of GNNs Graph Neural Networks naturally encode several key physical symmetries [4]:

- **Permutation invariance:** The aggregation operator $\oplus$ is insensitive to the ordering of neighbors, matching the indistinguishability of particles.
- **Locality:** Messages are exchanged only between connected nodes, enforcing that interactions are local.
- **Parameter sharing:** The same MLPs ϕ_{edge} and ϕ_{node} are reused across all edges and nodes, reducing overfitting and promoting transferability.

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` [5], 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 [6]. The systems boost recommendation accuracy even for lightly connected users. GNNs are also used in many other domains, such as traffic forecasting [7] and healthcare [8], where the data exhibit strong structural properties.

2.3 Interpretable AI

Interpretability methods for explainable AI can generally be divided into four main categories [9]. 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 from the first and the second categories respectively: *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 [3], 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 [10] many-body simulations; next, we train various one-step message-passing GNNs on these trajectories to predict particle accelerations; then we extract learned edge-wise message representations; finally we study their linear relationship to the underlying force laws and apply symbolic regression to recover these human-readable force equations.

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 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

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.

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 [11]: 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 ($m \in \mathbb{R}^{D_{\text{msg}}}$) where high flexibility enables state-of-the-art predictive accuracy but may obscures interpretability. A key insight from the original work [1] is that, for simple many-body force laws, the true inter-particle forces themselves lie in a much smaller message subspace, specifically the same dimensionality of the many-body systems. In this rediscovery research, we study 2D physical systems, indicating a 2D message space. Therefore, the message space of our GNN models should be manually concentrated most of their physical content in this 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} dimensional 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,
$$\mathcal{L}_{\ell_1} = \lambda \sum_e \|m_e\|_1 \tag{5}$$
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,
$$\mathcal{L}_{\text{KL}} = \beta \sum_e \text{KL}\big[q(m_e) \,\|\, \mathcal{N}(0, I)\big], \tag{6}$$
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.

3.2.2 Theoretical Explanation for Linear Relationship between Messages and True Forces

By following the original work [1], 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.

1. Edge-to-Node aggregation For every directed interaction $(i \to j)$, the edge model computes
$$e'_{i\to j} \;=\; \phi^e\big(x_i,\, x_j\big) \;\in^{D_{\text{msg}}}, \tag{7}$$

and node j pools all incoming messages by summation:

$$\bar{e}'_j \;=\; \sum_{i\to j} e'_{i\to j} \;\in\; {}^{D_{\text{msg}}}. \tag{8}$$

2. Perfect-prediction assumption Assume the node model then predicts the exact acceleration,

$$\hat{a}_j \;=\; \phi^v\big(v_j, \bar{e}'_j\big) \;=\; \frac{1}{m_j}\sum_{i\to j} F_{i\to j}\,, \tag{9}$$

so that "aggregate then predict" reproduces Newton's law.

3. Single-edge degeneration Restricting to a graph with a single neighbor enforces

$$\phi^v\big(v_j,\, e'_{i\to j}\big) \;=\; \frac{1}{m_j} F_{i\to j}\,, \tag{10}$$

which by linearity of summation implies the same holds when summing before prediction, i.e.

$$\phi^v\Big(v_j, \sum_{i:i\to j} e'_{i\to j}\Big) \;=\; \sum_{i:i\to j} \phi^v(v_j, e'_{i\to j})\,. \tag{11}$$

4. Commutativity implies linearity The only way for $\phi^v(v_j,\cdot)$ to commute with arbitrary sums of messages is for it to be a linear map in its second argument. Denote this linear operator by $A_j : {}^{D_{\text{msg}}} \!\to {}^{D}$. Then

$$A_j\big(e'_{i\to j}\big) = \frac{1}{m_j} F_{i\to j}, \quad A_j\big(\bar{e}'_j\big) = \frac{1}{m_j}\sum_{i\to j} F_{i\to j}. \tag{12}$$

where D denotes the dimension of the force. If A_j is (approximately) invertible, which requires $D_{\text{msg}} \geq D$, we obtain

$$e'_{i\to j} = A_j^{-1}\Big(\tfrac{1}{m_j} F_{i\to j}\Big), \tag{13}$$

i.e. each edge-message is exactly a linear transformation of the true force.

Enforcing the invertibility condition In practice we choose a *bottleneck* message size $D_{\text{msg}} = D$, or by applying sparsity/KL penalties that drive extra channels to zero, the network is encouraged to use exactly the minimal subspace needed.

This argument justifies our practice of selecting the top $D = 2$ message channels by variance and performing a separate linear regression (as described in details later) 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 message representations 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 above, and (2) symbolic regression to search for closed-form analytic expressions that relate interparticle 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

$$m_k \approx a_{k,x} F_x + a_{k,y} F_y + b_k, \tag{14}$$

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

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, which are built from variables, elementary functions (e.g. $+$, $-$, $\times$, $\div$, $\hat{}$, 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 [12] [3]. 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 [1] used the *Eureqa* software [13] as its primary symbolic-regression tools. In our work, we instead employ their recently developed *PySR* package [3], which offers a highly optimized evolutionary search, dramatically improving throughput for high-dimensional physics data without sacrificing expression simplicity.

Figure 1 illustrates the overall workflow of our pipeline, using the Spring-Bottleneck variant as a representative example.

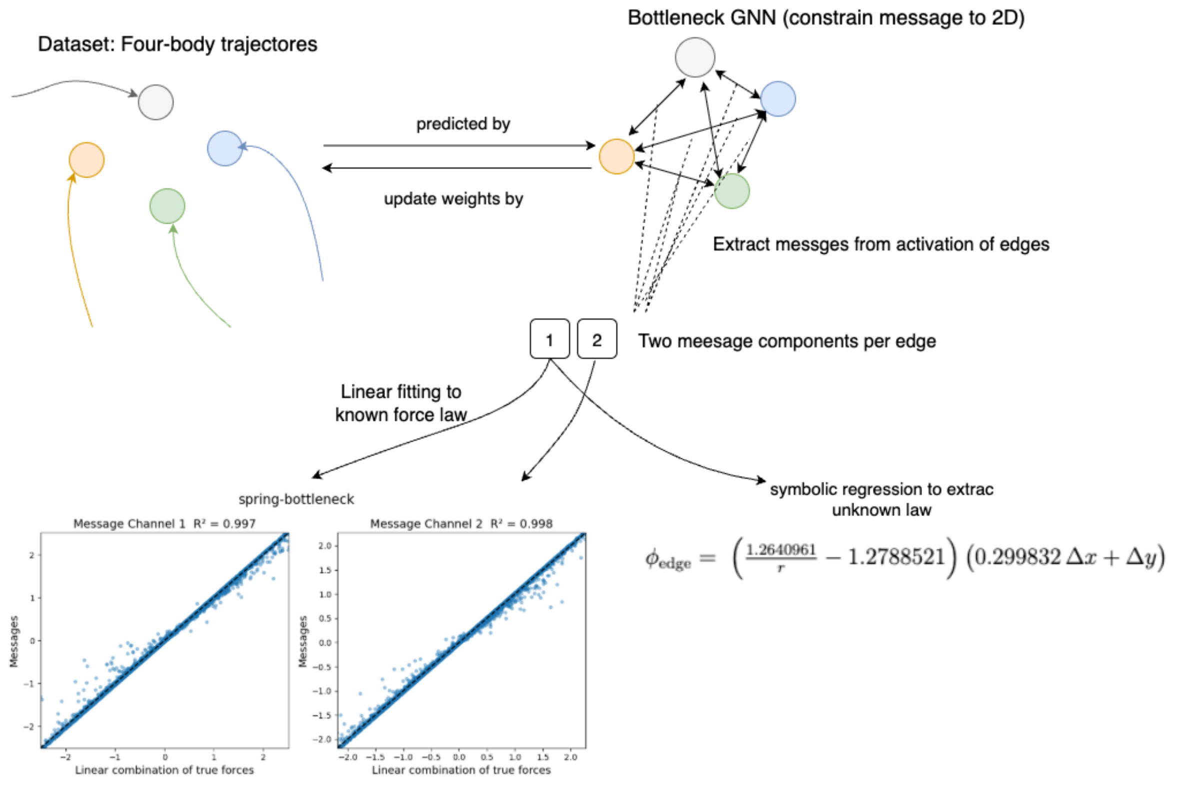
Dataset: Four-body trajectores
predicted by
update weights by
Bottleneck GNN (constrain message to 2D)
Extract messges from activation of edges
1
2
Two meesage components per edge
Linear fitting to known force law
symbolic regression to extrac unknown law
spring-bottleneck
Message Channel 1 R² = 0.997
Message Channel 2 R² = 0.998
Messages
Linear combination of true forces
$\phi_{\text{edge}} = \left(\frac{1.2640961}{r} - 1.2788521\right)(0.299832\,\Delta x + \Delta y)$

Figure 1: Overall Methodology (Spring-Bottleneck variant as a representative example).

4 Experiments

In this section, we describe our experiments in details. We first simulate four canonical 2D four-body systems to generate graph snapshots, which are then split to training and test sets. 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 [1]. We provide a full discussion of any differences, 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
$$\mathbf{F}_{ij} = k\,(r_{ij} - r_0)\,\hat{\mathbf{r}}_{ij}\,, \quad k = 1,\ r_0 = 1, \tag{15}$$
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
$$\mathbf{F}_{ij} = \frac{G\,m_i\,m_j}{r_{ij}^2}\,\hat{\mathbf{r}}_{ij}\,, \quad G = 1 \tag{16}$$

- **Inverse-square Coulomb force**: identical form to gravity but with signed charges,
$$\mathbf{F}_{ij} = \frac{q_i\,q_j}{r_{ij}^2}\,\hat{\mathbf{r}}_{ij} \tag{17}$$

- **Inverse-distance force**: a toy "$1/r$" law on every pair,
$$\mathbf{F}_{ij} = \frac{G\,m_i\,m_j}{r_{ij}}\,\hat{\mathbf{r}}_{ij}\,, \quad G = 1 \tag{18}$$

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
$$m_i\,\ddot{\mathbf{x}}_i \;=\; \sum_{j \neq i} \mathbf{F}_{ij} \tag{19}$$
using a symplectic Euler scheme with system-specific time steps and a fixed num-

ber 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

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. 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 (100 snapshots per trajectory) and then collapse the trajectory and time axes into a single flattened dimension along the trajectory (sample) dimension, unlike the original implementation in [1]. 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 $\mathbf{x} \in \mathbb{R}^{4\times 6}$ with its edge-index specification (fixed for all graphs) and the target acceleration matrix $\mathbf{y} \in \mathbb{R}^{4\times 2}$. These samples are then organized into data objects and batched using `PyTorch` [14]. 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 for 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 test 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 [14] and PyTorch Geometric [11].

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.

$$\text{Lin}(2n_f,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, D_{\text{msg}})\,. \quad (20)$$

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

$$\text{Lin}(D_{\text{msg}}+n_f,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, h) \xrightarrow{\text{ReLU}} \text{Lin}(h,\, n_{\text{out}}) \quad (21)$$

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 [15]. 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. The edge network ϕ^e now outputs $2D_{\text{msg}}$ features for each directed edge k:

$$\mu'_k \; = \; \phi^e_{1:D_{\text{msg}}}(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}), \quad \log \sigma'^2_k \; = \; \phi^e_{D_{\text{msg}}:2D_{\text{msg}}}(\mathbf{v}_{r_k}, \mathbf{v}_{s_k}), \tag{22}$$

   and we sample
$$\mathbf{e}'_k \; \sim \; \mathcal{N}\big(\mu'_k, \; \text{diag}(\sigma'^2_k)\big). \tag{23}$$

The sampled messages are then aggregated at each target node i,

$$\bar{\mathbf{e}}'_i \;=\; \sum_{k:r_k=i} \mathbf{e}'_k, \quad \hat{\mathbf{v}}'_i \;=\; \phi^v(\mathbf{v}_i, \bar{\mathbf{e}}'_i), \tag{24}$$

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 $\mathcal{N}(0, I)$:

$$\mathcal{L}_{\text{MAE}} = \frac{1}{N_{\text{particles}}} \sum_{b=1}^{B} \sum_{i=1}^{N} \left\| \hat{\mathbf{a}}_i^{(b)} - \mathbf{a}_i^{(b)} \right\|_1, \tag{25}$$

$$\mathcal{L}_{\text{KL}} = \frac{1}{N^e} \sum_{k=1}^{N^e} \sum_{j=1}^{D_{\text{msg}}} \frac{1}{2} \Big(\mu'^2_{k,j} + \sigma'^2_{k,j} - \log \sigma'^2_{k,j} - 1 \Big). \tag{26}$$

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:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{MAE}} + \mathcal{L}_{\text{KL}}. \tag{27}$$

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

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

$N_{\text{particles}} = B \times N$**:** 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.

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 [15] algorithm with an initial learning rate of 1×10^{-3} and an L_2 weight-decay coefficient of 1×10^{-8} applied to every parameter. To modulate the learning rate we employ OneCycleLR scheduler [16]: 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 loss, backpropagate, and finally advance both the optimizer and the learning-rate scheduler.

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*

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

   where

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

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 edge-level dataset, with one sample per directed edge. Since each graph has $N = 4$ particles and thus $N(N-1) = 12$ directed edges, and there are 1024 test graphs, the resulting frame has

$$12 \times 1024 \;=\; 12\,288$$

sample. Each sample contains

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

4.6 Analytic Function Recovery

Recall the 2D force laws 15 16 17 18. 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 $F(r)$:

$$\textbf{(1) Mass–spring (Hooke's law):}\quad F(r) = k\,(r - r_0) \quad (k = 1,\, r_0 = 1),$$

$$\Longrightarrow \begin{cases} F_x = (r - r_0)\,\dfrac{\Delta x}{r}, \\ F_y = (r - r_0)\,\dfrac{\Delta y}{r}. \end{cases}$$

$$\textbf{(2) Inverse-square gravitational:}\quad F(r) = \frac{G\,m_i\,m_j}{r^2} \quad (G = 1),$$

$$\Longrightarrow \begin{cases} F_x = m_i\,m_j\,\dfrac{\Delta x}{r^3}, \\ F_y = m_i\,m_j\,\dfrac{\Delta y}{r^3}. \end{cases}$$

$$\textbf{(3) Inverse-square Coulomb:}\quad F(r) = \frac{q_i\,q_j}{r^2},$$

$$\Longrightarrow \begin{cases} F_x = q_i\,q_j\,\dfrac{\Delta x}{r^3}, \\ F_y = q_i\,q_j\,\dfrac{\Delta y}{r^3}. \end{cases}$$

$$\textbf{(4) Inverse-distance ``toy'' } 1/r\textbf{:}\quad F(r) = \frac{k}{r} \quad (k = 1),$$

$$\implies \begin{cases} F_x = \dfrac{\Delta x}{r^2}, \\ F_y = \dfrac{\Delta y}{r^2}. \end{cases}$$

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 Equation 14. We evaluate its predictive power on edges of test set by feeding the true force components (F_x, F_y) into the learned model to produce

$$\hat{m}_{k,i} = a_{k,x}\, F_{x,i} + a_{k,y}\, F_{y,i} + b_k \quad \text{for each edge } i. \tag{28}$$

We then compute the coefficient of determination

$$R_k^2 = 1 - \frac{\sum_i \big(m_{k,i} - \hat{m}_{k,i}\big)^2}{\sum_i \big(m_{k,i} - \bar{m}_k\big)^2}, \quad \bar{m}_k = \frac{1}{N}\sum_i m_{k,i}\,, \tag{29}$$

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 [1], 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. After a preliminary hyperparameter sweep, we settled on the following operator sets per system:

- **Spring:** only binary $\{+, -, \times, /\}$ operators are sufficient to recover Hooke's law.
- **Inverse-Distance:** add another binary power operator ($\wedge$).
- **Coulomb & Inverse-Square:** include a cubic (`cube`) operator but without explicitly telling PySR that it should be applied to which variable. The search must discover on its own that a cubic inverse of the distance best fits the data.

PySR evaluates each candidate by trading off simplicity (expression complexity) against accuracy (mean squared error). After each generation, it constructs a Pareto front of candidate equations. Once the evolutionary run completes, PySR applies its default `model_selection="best"` policy to pick a single equation with the best error-complexity trade-off.

4.7 Reproducibility of our research

All randomness was fixed to seed 0 to ensure reproducibility:

- **NumPy** [17] for numerical operations
- **JAX** [10] for data generation
- **PyTorch** [14] for model training
- **PySR** [3] for symbolic regression
- **pandas** [18] for subsampling

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 behavior and acceleration-prediction performance on the 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

PySR-based symbolic regression experiments, comparing the recovered closed-form expressions against the ground-truth force laws.

5.1 Model Performance

Note that the acceleration-prediction accuracy is not our primary focus. We discuss the model performance to illustrate the complexity of the physical models.

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 the lowest loss values, reflecting their relatively simple linear and $1/r$ force laws. By contrast, the Coulomb system shows 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.

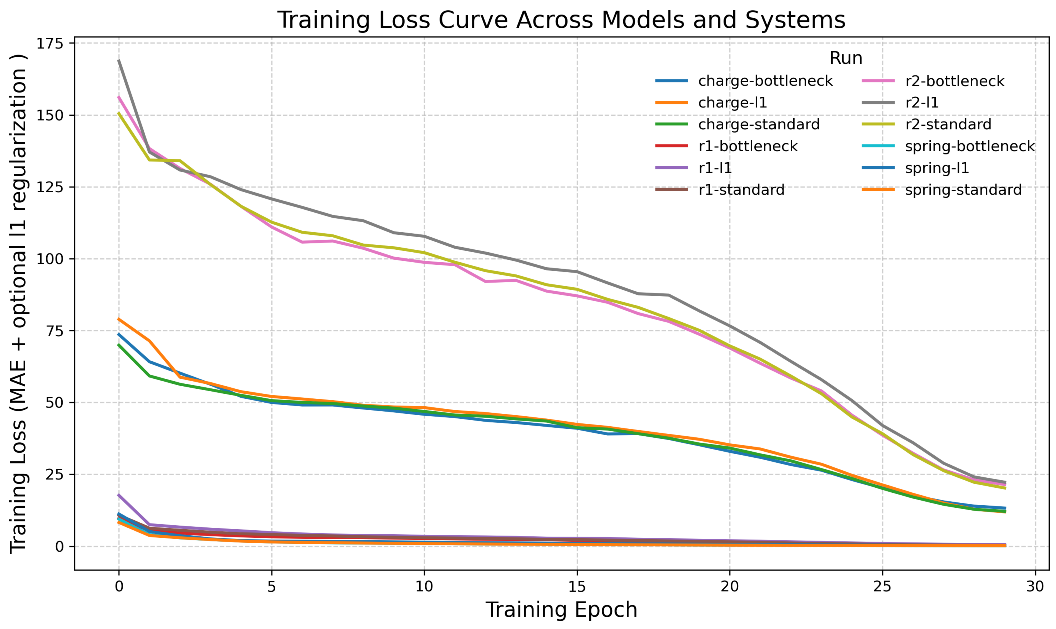

Figure 2: Training loss curve across all model variants (except KL-regularized) and all systems.

Within each physical system the three model variants show similar training-loss trajectories. This indicates that constraining messages does not significantly affect the training loss behavior.

Figure 3 shows the KL-regularized models. Their behavior mirror those of the other three variants across all four systems.

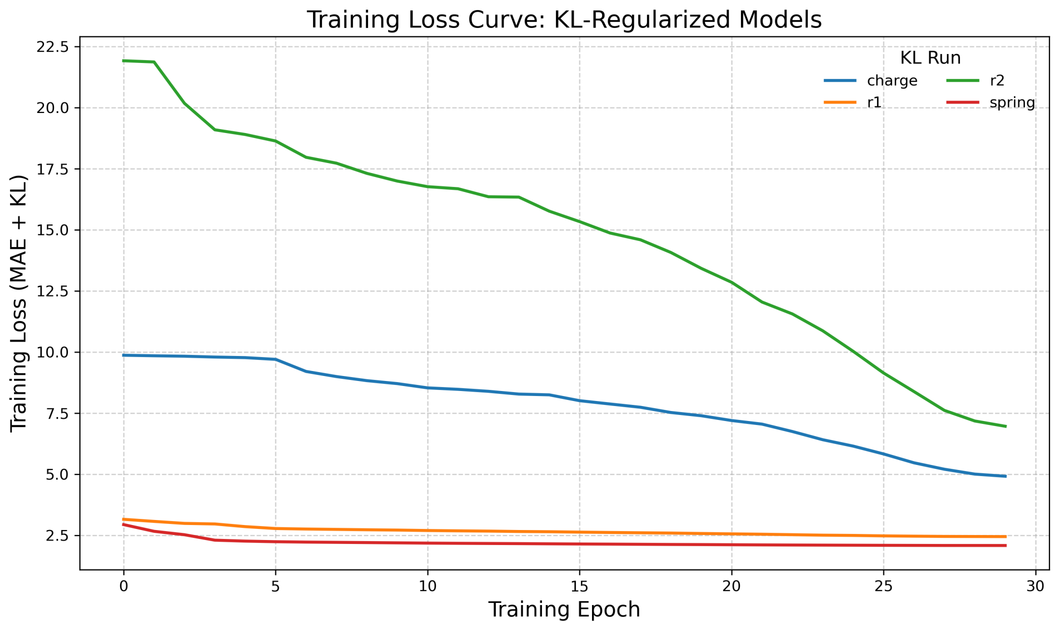

Figure 3: Training loss curve of KL-regularized models and all systems.

5.1.2 Test-set MAE

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). The ranking of test MAE across the four physical systems closely matches the trends seen in training loss.

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

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 4 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 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.

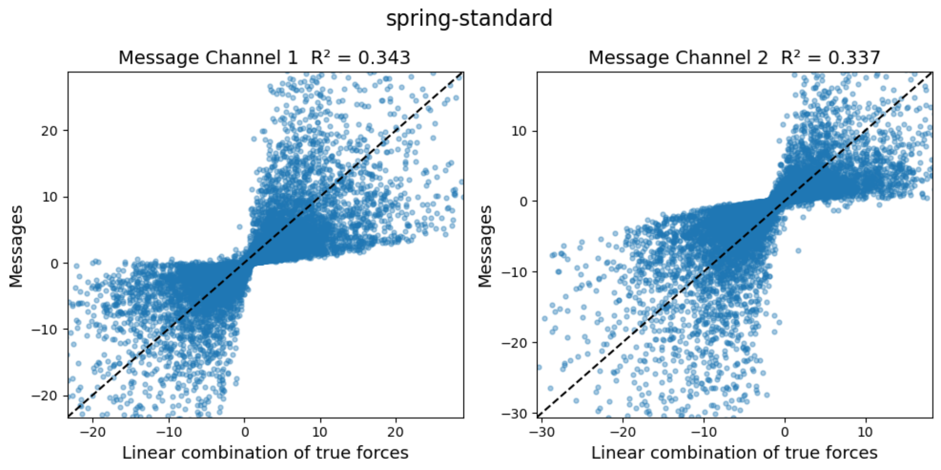

Figure 4: Spring system – Standard GNN variant. The dashed line shows $y = x$.

Bottleneck Model Figure 5 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.

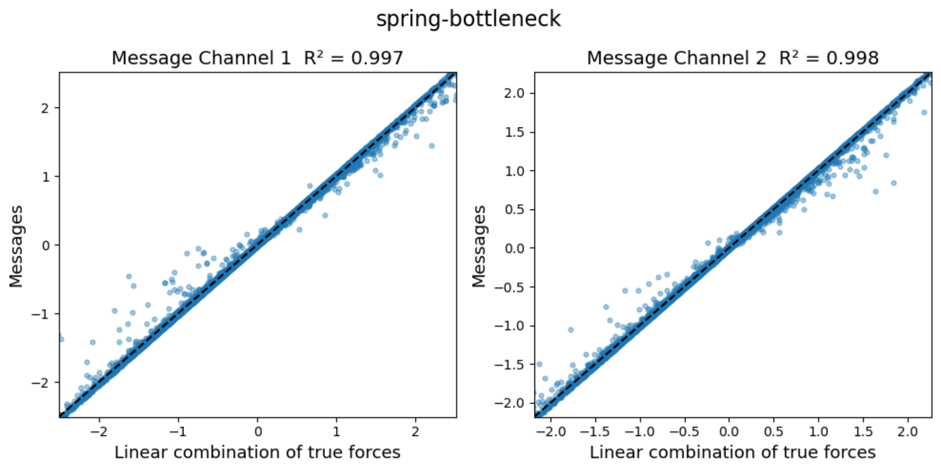

Figure 5: Spring system – Bottleneck GNN variant.

Compared to the Standard GNN, the Bottleneck variant achieves nearly perfect linear relationship between message channels and true forces. This dramatic 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 6 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.

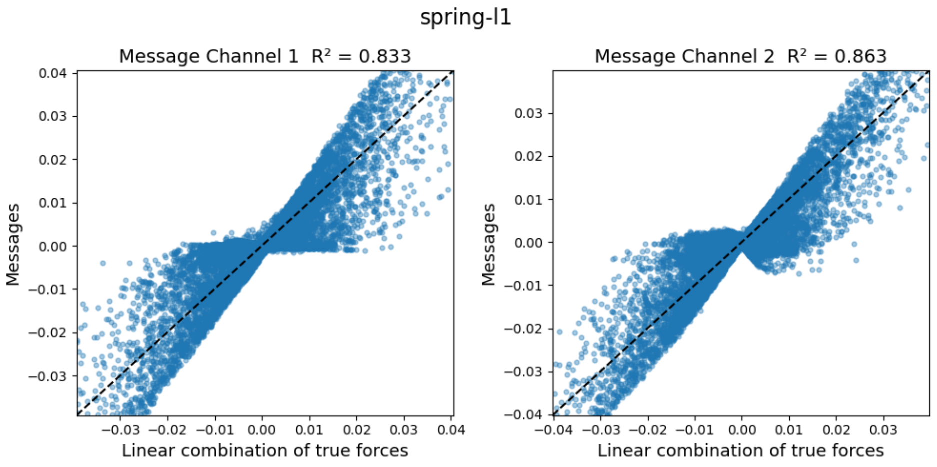

Figure 6: Spring system – L_1-regularized GNN.

KL-regularized Model Figure 7 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.

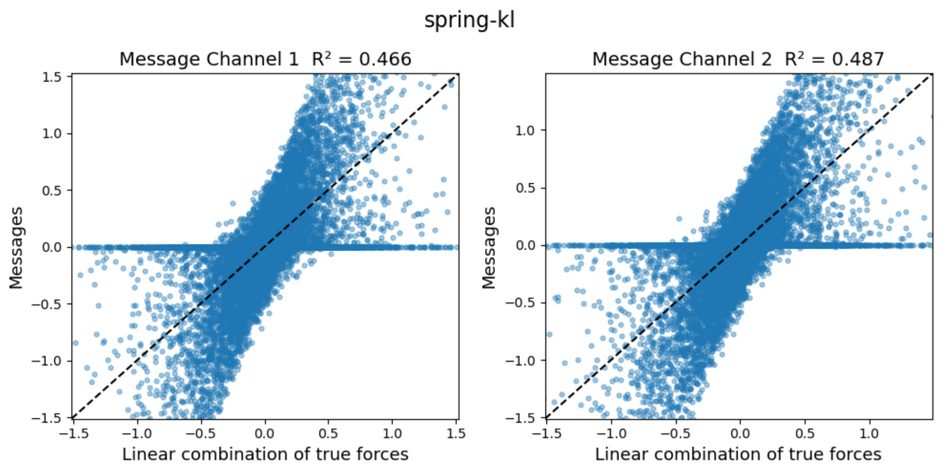

Figure 7: Spring system – KL-regularized GNN. Note the The KL penalty drives many messages to zero mean.

Message-Space Scaling and Normalization In the original work [1], 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 $\text{KL}(q \| \mathcal{N}(0, I))$ 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 discuss 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	0.763, 0.162	0.443, 0.158	0.584, 0.350
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 the Coulomb systems, we observe large discrepancies between the two top-variance channels 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 [1], 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 sensitive 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 [1].

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, 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 [1], 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. Table 4 displays the complete set of symbolic-regression outcomes for all four force laws and GNN variants. Beyond binary success/failure markers, we further inspect each "partial" case to identify whether the recovered formula omitted physical factors (e.g. masses, charges or one dimension of the force) or just failed to be selected as the best despite appearing in the Pareto front.

We then compare these SR trends to our earlier linear-fit R^2 results to see if high linear alignment reliably predicts symbolic discoverability, and we discuss the trends with the original paper [1]'s reported outcomes to assess reproducibility. Finally, we highlight several representative examples to show the full recovered expressions and detailed analysis.

Table 4: Symbolic regression outcome markers for each system and GNN variant.

System	**Standard**	**Bottleneck**	ℓ_1	**KL**
Spring	$\times$	✓	$\circ\star$	$\star$
Coulomb	$\times$	✓	$\times$	✓
Inverse-square	$\times$	$\circ\star$	$\times$	$\circ\star$
Inverse-distance	$\times$	$\circ$	$\circ$	$\circ$

✓	perfect discovery matching the true law.
$\times$	no valid formula recovered.
$\circ$	partial discovery (kernel found, but missing factors and/or dimensions).
$\star$	correct formula exists in the Pareto front but was not selected as "best" by PySR.
$\circ\star$	both partial and not top-ranked.

Additional Key Observations Several patterns stand out in the "partial" and "starred" cases:

- **Spring–ℓ_1 ($\circ\star$):** PySR recovers an equation which only has one spatial component of Hooke's law, omitting the orthogonal dimension entirely.
- **Inverse-square–Bottleneck ($\circ\star$):** An equation captures the correct $1/r^3$ kernel but drops both mass factors $m_i m_j$ and one force dimension, recovering only the geometric shape.
- **Inverse-square–KL ($\circ\star$):** A similar omission of both mass factors occurs, even though the geometric dependence is correctly found.
- **Inverse-distance systems ($\circ$):** Every variant except Standard consistently recovers the $1/r$ kernel but fails to include the source-node mass m_1.

Standard GNN Variant As expected, the unconstrained Standard model fails to recover any of the true analytic force laws ($\times$ in all four systems), despite the fact that its messages exhibited non-zero linear correlations with the spring and r_1 forces. This indicates that a moderate linear alignment alone does not suffice for SR to rediscover closed-form expressions when the message space remains high-dimensional and noisy.

Bottleneck GNN By fixing the message dimension to two, the Bottleneck variant either perfectly rediscovers the true force law or else returns a clear geometric kernel (e.g. $\Delta x/r^3$) with missing scalar factors, but it never completely fails. This demonstrates that constraining the latent space to match the physical force dimensionality greatly aids symbolic recovery.

ℓ_1–regularized GNN Adding an ℓ_1 penalty concentrates information into a few channels. This sometimes allowing PySR to discover the physical and geometrical kernel shape of the underlying equation, but overall the performance of ℓ_1–regularization with respect to discovering physical law by using symbolic regression is not good.

KL–regularized GNN It sometimes recovers a partial kernel but also occasionally succeeds.

5.3.1 Overall SR Trend and Consistency with Linear-Combination Fitting

Our symbolic-regression experiments further confirm that constraining the message space is crucial for recovering physical laws. In particular, the Bottleneck variant outperforms all others, regularly yielding perfect or near-perfect closed-form expressions. The KL-regularized model comes next, managing to recover the correct geometric kernels more often than the ℓ_1 variant, which itself still exceeds the unconstrained Standard baseline. Although the relative ordering of ℓ_1 versus KL differs slightly from our linear-fit results, both sets of experiments consistently demonstrate the same core insight: stronger compression or dimensionality constraints on the GNN's internal messages markedly improve the interpretability and recoverability of the underlying analytic force laws.

5.3.2 Comparison with the Original Study

Overall, our PySR-based symbolic regression outcomes exhibit similar qualitative ordering as reported by the original study [1]. However, since the original paper only presents binary success/failure outcomes for all the variants and systems, it remains unclear whether every individual case is reproducible. Notably, in both of their 2D examples for the inverse-distance (r^{-1}) and inverse-square (r^{-2}) laws, the recovered formulas also omit the source-node mass m_1, exactly as we observe—indicating that at least this particular factor-dropping behavior is consistently reproducible across implementations.

5.3.3 Representative Examples

Spring System (Bottleneck) The Bottleneck variant on the Spring system exhibits perfect results in both linear fitting and symbolic regression. PySR recovers the following closed-form expression for one message channel:

$$\phi_{\text{edge}} = \left(\tfrac{1.2640961}{r} - 1.2788521\right)\left(0.299832\,\Delta x + \Delta y\right) \tag{30}$$

Here $\Delta x = x_j - x_i$, $\Delta y = y_j - y_i$, and $r = \sqrt{\Delta x^2 + \Delta y^2}$. The coefficients 1.2640961 and 1.2788521 are both very close to 1, and the small cross-term $0.299832\,\Delta y$ merely reflects a rotated basis in the two-dimensional space. This near-perfect match confirms that the Bottleneck model forces the GNN to rediscover the true Hooke's law directly into its message channels.

Inverse-Square System (Bottleneck) In the Bottleneck variant on the inverse-square system, PySR recovers the following form for one message channel:

$$\phi_{\text{edge}}(\Delta x, \Delta y, r) \;=\; \frac{0.0021036337\,\Delta x}{r^3}. \tag{31}$$

This expression matches the geometric kernel of the true gravitational law, $\Delta x/r^3$, up to the constant factor 0.0021 which absorb the average product of masses $m_i m_j$ without treating them as variables. Additionally, only the x-component of the force is recovered in this channel. This illustrates that a two-dimensional bottleneck has an ability to at least partially make the GNN encode spatial projection of the $1/r^2$ force law.

Inverse-distance System (Bottleneck) Below is the case of the Bottleneck variant on the inverse-distance system:

$$\phi_{\text{edge}} = \left(\tfrac{\Delta y + \Delta x/0.90422577}{r + 0.009675505} + 0.0037858116\right) \times \frac{-0.53762525\,m_2}{r} + 0.025902085. \tag{32}$$

Here m_2 is the target-node mass. By ignoring the $\varepsilon_1 = 0.0037858116$ which provides numerical stability and $\varepsilon_2 = 0.009675505$, it turns to:

$$\phi_{\text{edge}} \;\approx\; -0.5376\,m_2\,\frac{\Delta y + 1.106\,\Delta x}{r^2} + \mathit{constant}. \tag{33}$$

5.4 Short Conclusion of Core Part and Motivation of Extension

Overall, both our linear-combination fitting and PySR-based symbolic regression experiments reproduce the key findings of the original work [1], although certain details, such as the relative ordering of KL versus ℓ_1 in linear R^2 values, or the presence of mid-range fits instead of extreme near-0/near-1 scores, do not perfectly reproduce every case. Moreover, the two experiments reinforce each other: systems with high linear alignment (R^2) systematically produce successful symbolic rediscoveries, while low R^2 corresponds to SR failures or partial recoveries.

The symbolic-regression outcomes also suggest two possible directions for further investigation:

1. Acceleration-Based Regression for r^{-1} Systems In several $1/r$ experiments, SR consistently recovers the geometric kernel but omits the source-node mass m_1. One possible explanation is that ϕ_{edge} is encoding not the raw force $F_{i\to j}$ but the acceleration contribution $a_{i\to j} = F_{i\to j}/m_i$. To test this hypothesis, we will re-run linear-combination fitting on the r^{-1} messages using the true acceleration contributions $a_{i\to j}$ as targets, and compare the resulting R^2 against the force-based fits. A significant increase in R^2 would confirm that edge messages learn acceleration kernels rather than forces.

2. Component-Wise Channel Regression We also observe that some SR-recovered equations project onto only one spatial dimension (e.g. $\Delta x/r^3$ alone). This raises the conjecture that the GNN may prefer to assign each message channel to a single force component, rather than encoding the full 2D vector in each. To explore this, we will perform separate linear regressions of each channel against the individual force components F_x and F_y, rather than the combined linear transformation. Comparing these component-wise R^2 scores will reveal whether channels indeed specialize to one dimension of the physical interaction.

6 Extension i

We discuss whether the GNN's messages in the Inverse-distance systems more directly encode acceleration rather than raw force by using the same methodology and experimental setup in the core part, except that for each channel m_k, we fit

$$m_k \;\approx\; \alpha_x\, a_x + \alpha_y\, a_y + b \tag{34}$$

where acceleration components replace force components.

Results Figure 8 9 10 shows the results for the Bottleneck, ℓ_1–regularized and KL-regularized variants of the Inverse-distance system respectively.

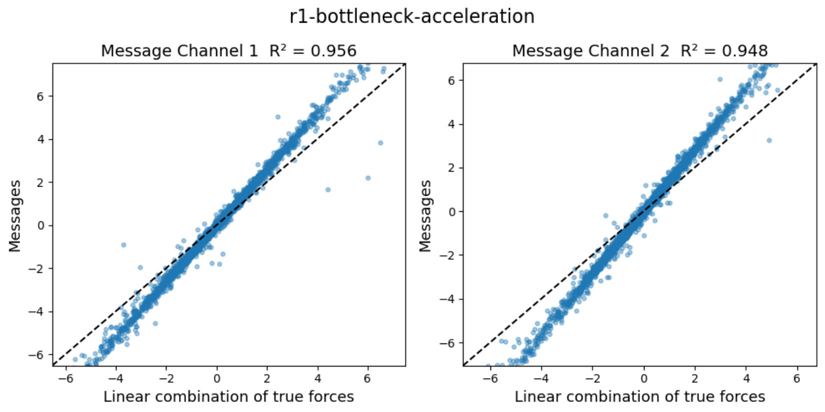

Figure 8: Inverse-distance system – Bottleneck GNN variant. Fitting target is the linear combination of acceleration components.

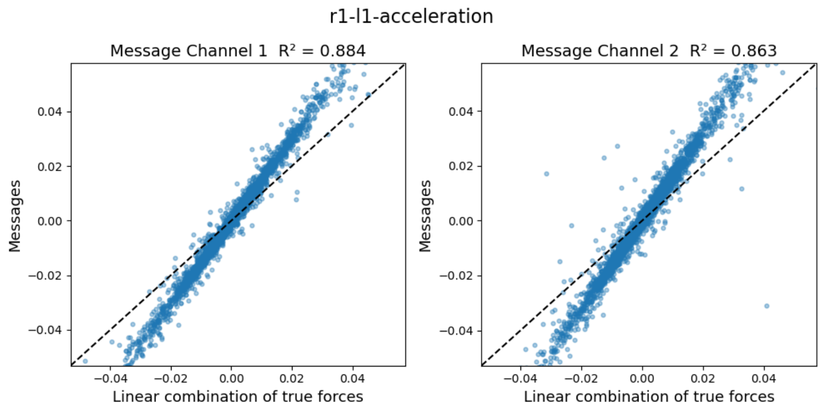

Figure 9: Inverse-distance system – ℓ_1–regularized GNN variant.

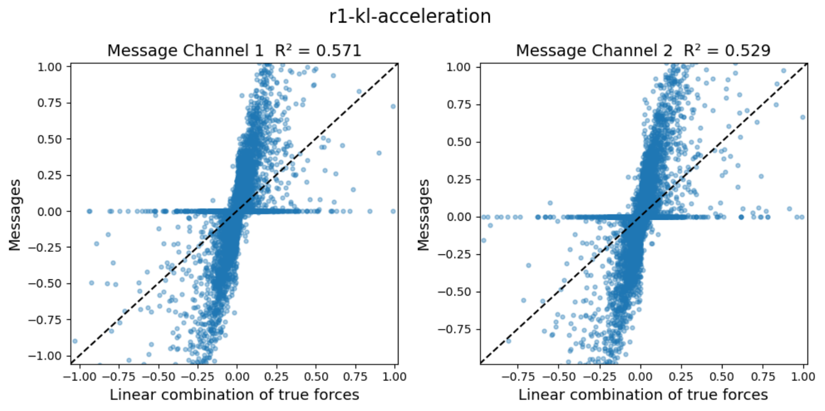

Figure 10: Inverse-distance system – KL-regularized GNN variant.

Interpreting SR "Missing" Factors as True Signal When we compare these acceleration-based R^2 scores to the original force-based fits (see Table 3), we observe a dramatic increase in all three cases. Most significantly, the Bottleneck GNN's channels climb to over 0.95, indicating near-perfect linear alignment with the true acceleration law. This confirms that the learned message embeddings more directly encode acceleration contributions than raw forces.

This extension highlights a key advantage of PySR: when the recovered symbolic formula omits an expected variable or factor, it does not necessarily indicate a failure of this tool to "see" that variable. Rather, it often reflects the actual quantity encoded by the GNN's message channels. This showcases PySR's reliability in extracting whatever physical law the model has learned. Consequently, the consistency between symbolic regression outcomes and our follow-up acceleration fits provides strong evidence for the overall validity of the symbolic extraction pipeline.

7 Extension ii

To investigate whether individual message channels specialize in a single force component, we perform separate univariate linear regressions of each of the top-D message channels against the true force components F_x and F_y. Recall Equation 28, here we fit two independent models per channel:

$$m_{k,i} \approx a_{k,x}\, F_{x,i} + b_{k,x}, \tag{35}$$

$$m_{k,i} \approx a_{k,y}\, F_{y,i} + b_{k,y}, \tag{36}$$

For each fit we compute the coefficient of determination. High $R^2_{k,x}$ (resp. $R^2_{k,y}$) indicates that channel k predominantly encodes the x- (resp. y-) component of the physical force.

Results The results of several representative cases are displayed in Table 5. The cases with only x-dimension discovered showcase that they only have linear relationship with F_x components. Surprisingly, the "best" Spring–Bottleneck model also splits cleanly into a channel specialized learning mechanism.

System	**Channel**	$\boldsymbol{R^2(F_x)}$	$\boldsymbol{R^2(F_y)}$
Spring – Bottleneck	**1**	**0.089**	**0.916**
	2	0.914	0.075
Spring – ℓ_1 only x-dim discovered in SR	1	0.336	0.510
	2	**0.835**	**0.023**
Inverse-square – Bottleneck only x-dim discovered in SR	**1**	**0.655**	**0.058**
	2	0.035	0.519
Inverse-distance (acceleration) – Bottleneck	**1**	**0.536**	**0.434**
	2	0.620	0.315

Table 5: Component-wise channel regression R^2 scores for selected cases. Boldface indicates the channel selected for symbolic-regression analysis.

Further Interpretation First, the overall methodology in this research, constraining message information to the dimensionality of the underlying force law, does *not* guarantee that each channel will align exclusively with one Cartesian component. Instead, the GNN is free to learn any invertible linear transform of the (F_x, F_y) vector, yielding rotated axes in the message space. As a result, some channels may mix both force components to varying degrees, others prefer to map one force component to one message channel respectively.

Second, comparing Equations 30 and 33, we see that in the Spring–Bottleneck case the coefficient on the Δx term is substantially smaller than that on Δy. This basis-change highlights that PySR has extracted primarily the y-component of the spring force, matching the high $R^2_{F_y}$ and low $R^2_{F_x}$ reported in Table 5. By contrast, in the inverse-distance–Bottleneck model the Δx coefficient slightly exceeds the Δy coefficient, and likewise $R^2_{F_x}$ is marginally higher than $R^2_{F_y}$.

This consistency between symbolic regression and component-wise channel regression demonstrates that PySR's evolutionary search is sensitive enough to sometimes recover even minor projections onto secondary directions. When one force component dominates a channel's activation, a simple univariate regression treats the weaker component as noise (yielding $R^2 \approx 0$), but PySR will still include that small contribution in the final "best" equation with a much smaller coefficient, which reflects the true rotated basis learned by the edge model. This showcases PySR's ability to capture the exact underlying physical laws.

Note that this observation is not meant to imply that the linear-regression and symbolic-regression results are themselves linearly correlated, but rather to emphasize PySR's sensitivity to weak projections in the learned basis.

To further support this point, we also apply PySR to the second-highest-variance message channel in the Spring–Bottleneck and Inverse-distance–Bottleneck variants respectively:

$$\phi_{\text{edge}}(\Delta x, \Delta y, r, m_1, m_2) = \left(-0.36478665\,\frac{1}{r} + 0.38995925\right)(\Delta x + \Delta y)\,, \tag{37}$$

$$\phi_{\text{edge}}(\Delta x, \Delta y, r, m_1, m_2) = 0.63738245\,(\Delta y - \Delta x)\,\frac{m_2}{r^{1.8972397}}\,. \tag{38}$$

Although the component-wise R^2 fits can differ substantially, Equations 37 and 38 both assign comparable weights to Δx and Δy. This tells that, symbolic regression results and the linear-regression R^2 results, are not necessarily linearly related.

8 Conclusion and Further Research Directions

Conclusion In this work, we have reproduced the core experiments of [1], validating their pipeline that combines Graph Neural Networks (GNNs) with symbolic regression to extract interpretable physical laws. We chose GNNs for their strong, physics-motivated inductive biases [2], and showed that by constraining the edge-message representation in the message passing layer one obtains latent embeddings that are interpretable in two complementary ways:

1. Apply a linear fit to the true force laws, demonstrating strong linear alignment with ground-truth physics.

2. Apply symbolic regression to extract the underlying physical laws.

Moreover, we confirmed that these two interpretability methods produce consistent and reproducible results with each other across sixteen model–system combinations generally.

Finally, we proposed and evaluated two extensions, acceleration-based linear regression and component-wise linear regression, that deepen our understanding of how GNN message functions encode physical quantities. The results demonstrate that the exact mechanism of how the message information encodes the underlying physical laws remains unclear. This suggests further research directions.

Future Research Directions To further validate and deepen our findings on interpretable GNNs with constrained message representations, we propose the following directions:

- **Diverse Scientific Systems.** Apply the pipeline to a broader domains and scale up to larger dataset. This will test the generality of the core methodology across other scientific domains.
- **Complexity–Driven Mechanism Analysis.** Leverage inherent measures of system complexity, such as the non-linearity of a system, to study how message-space constraints behave. In particular, investigate whether constrained embeddings capture full vectorial information or collapse to single components at different complexity levels.
- **SR–LR Correlation Study.** Systematically quantify the relationship between symbolic-regression and linear regression results. Key questions include:
  - Does a high linear-fit R^2 always predict a reliable symbolic equation?
  - Can SR uncover nonlinearity that linear regression cannot capture?
  - Are there regimes where SR and LR results diverge, and what does that reveal about the learned message basis?

References

[1] 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.

[2] 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.

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

[4] S. J. Prince, *Understanding Deep Learning.* The MIT Press, 2023.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.

[11] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," 2019.

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

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

[14] 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.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

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

[17] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[18] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference* (Stéfan van der Walt and Jarrod Millman, eds.), pp. 56 – 61, 2010.

Appendices

A Difference from Original Train/Test Split

In the original work [1], training and test snapshots are drawn from the same trajectories by splitting along the temporal axis (e.g. reserving later time-steps for testing). Consequently, each test point is a future frame of a trajectory whose earlier frames were included in the training set, and complete independence between train and test cannot be achieved even when subsampling every fifth frame.

By contrast, we perform a split where entire trajectories are assigned wholly to either the training or the test set. This ensures that no time series contributes to both sets. Such a split provides a stronger guarantee of independence and a more rigorous evaluation of out-of-distribution performance.

B Declaration of AI generation tools

Declaration of AI generation tools for the report part is made here:

Although some of the advise is not accepted, ChatGPT was used to help this report:
1. The format of the mathematical calculations was helped by it to make the process in a publication quality.
2. The format of plots, tables and itemized expression was helped to make things in a publication quality.
3. It suggested some alternative wording and other academic language issues throughout the whole report.
4. It provided proofreading for some of text, including the description of the theory of GNN, the description of methodology and the experimental setup, the discussion of results and the overall logic flow among paragraphs.