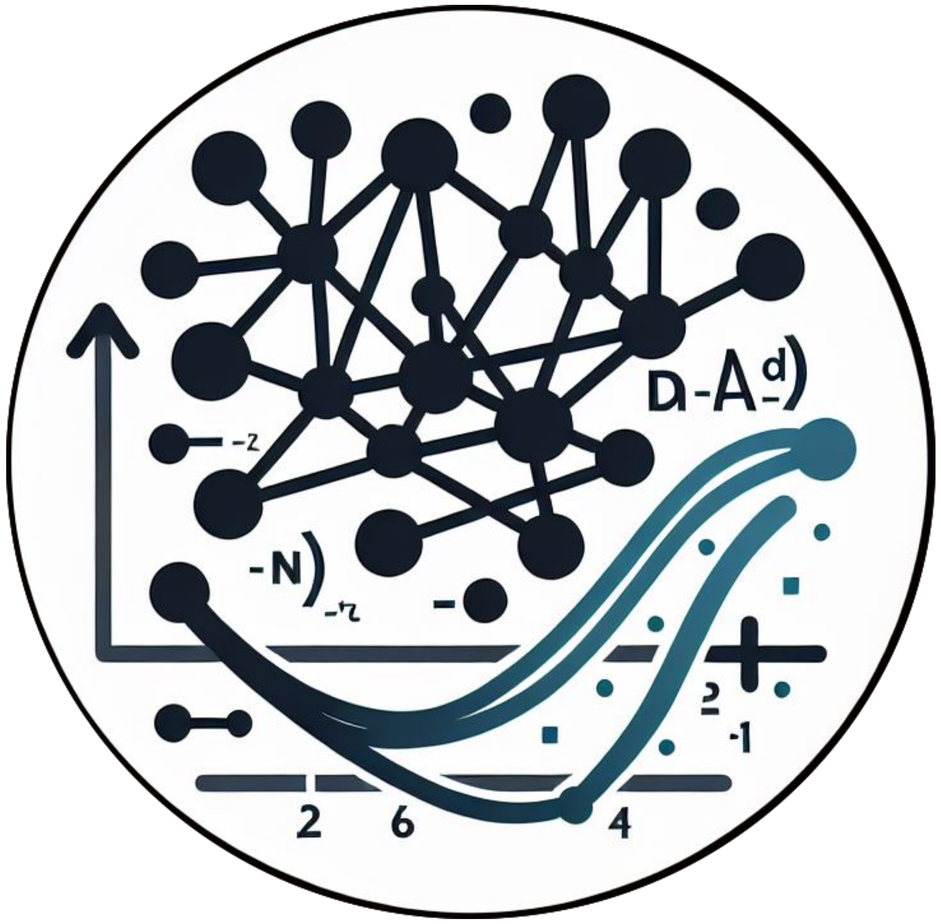
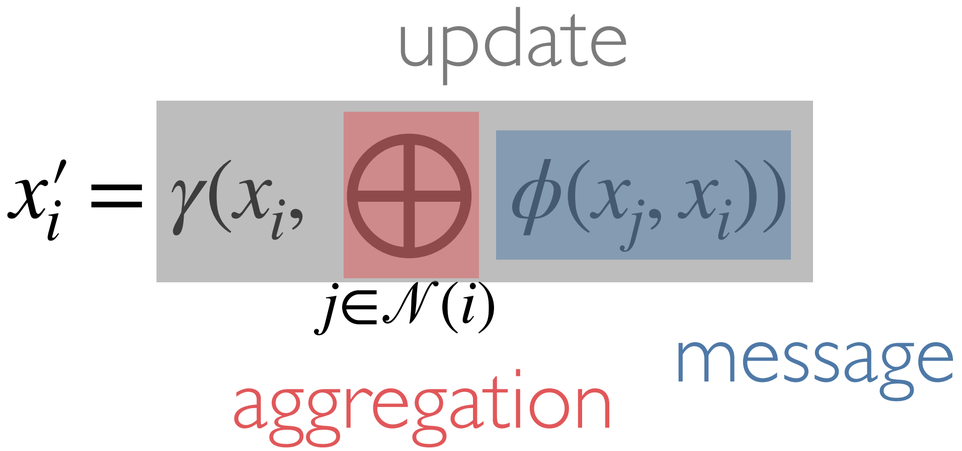
Graph Neural Net Implementation

PyTorch Geometric



Deep Learning for Engineers
Andrew Ning
aning@byu.edu


update
$x_i' = \gamma(x_i, \bigoplus_{j \in \mathcal{N}(i)} \phi(x_j, x_i))$
aggregation
message

Basic Linear Layers: GCNConv, GraphConv, SAGEConv

GCNConv:

$$x_i' = W \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{x_j}{\sqrt{|\mathcal{N}(i)||\mathcal{N}(j)|}} + b$$

GraphConv:

$$x_i' = W_1 x_i + W_2 \sum_{j \in \mathcal{N}(i)} x_j + b$$

SAGEConv:

$$x_i' = W_1 x_i + W_2 \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} x_j + b$$

also has an optional L2 normalization

Edges

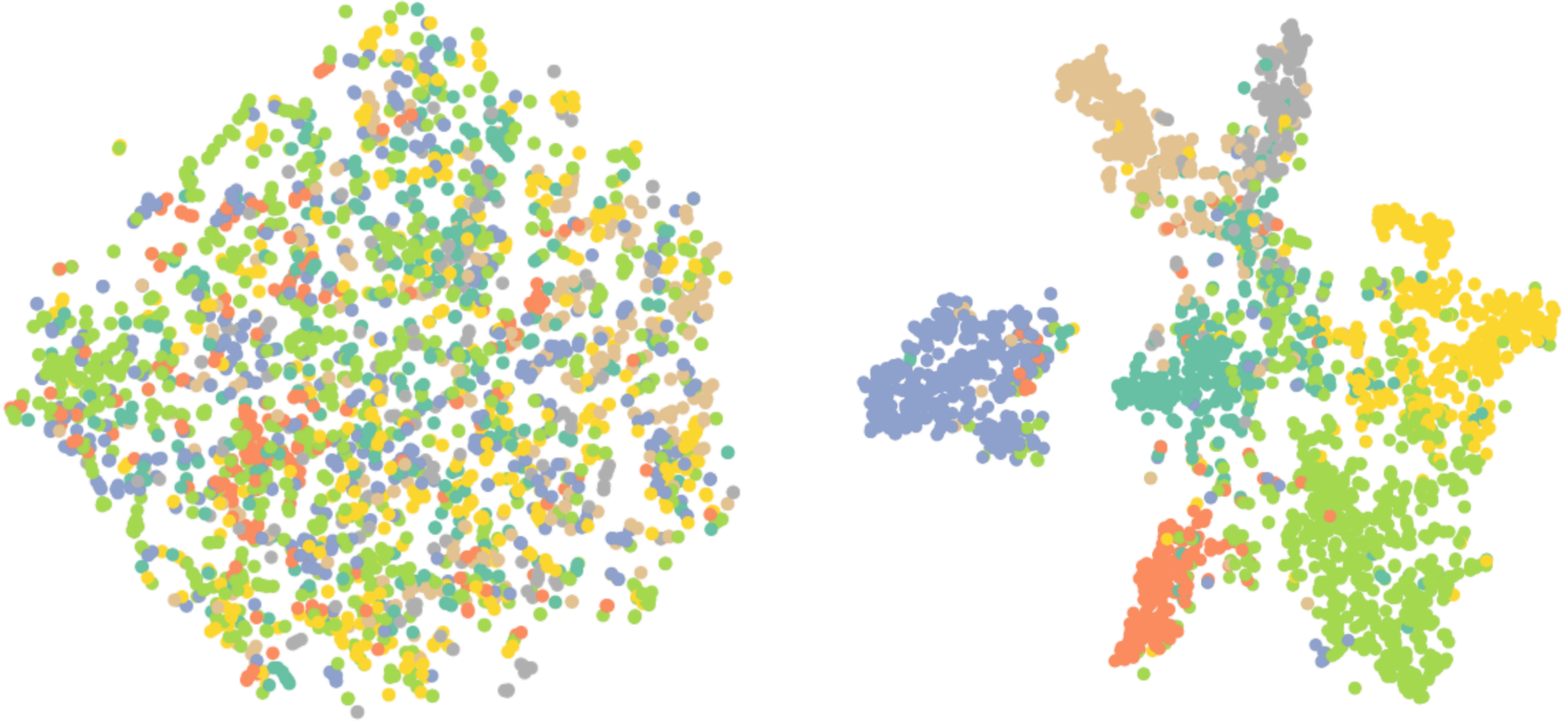
Linear (GCNConv, GraphConv, SAGEConv):
can optionally *provide* an edge weight (scalar)

Attention (GATConv, TransformerConv):
learn edge weights (scalar)

Edge-conditioned layers (NNConv, GINEConv):
use edge *feature vectors* to modify messages

Node Classification with GNN

https://pytorch-geometric.readthedocs.io/en/stable/get_started/colabs.html



Graph Classification

https://pytorch-geometric.readthedocs.io/en/stable/get_started/colabs.html

Develop Your Own Graph Convolutional Layer

update

$$x_i' = \gamma(x_i, \bigoplus_{j \in \mathcal{N}(i)} \phi(x_j, x_i))$$

aggregation

message

Develop Your Own Graph Convolutional Layer

update

$$x_i' = \gamma(x_i, \bigoplus_{j \in \mathcal{N}(i)} \phi(x_j, x_i))$$

aggregation

message

update

$$x_i' = W_1 x_i + W_2 \sum_{j \in \mathcal{N}(i)} x_j + b$$

aggregation

message

Develop Your Own Graph Convolutional Layer

update

$$x_i' = \gamma(x_i, \bigoplus_{j \in \mathcal{N}(i)} \phi(x_j, x_i))$$

aggregation

message

update

$$x_i' = W_1 x_i + W_2 \sum_{j \in \mathcal{N}(i)} x_j + b$$

aggregation

message

```python
import torch.nn as nn
from torch_geometric.nn import MessagePassing

class GraphNetwork(MessagePassing):

    def __init__(self, inchan, outchan, aggr='add'):
        super(GraphNetwork, self).__init__(aggr=aggr)
        self.lin1 = nn.Linear(inchan, outchan, bias=False)
        self.lin2 = nn.Linear(inchan, outchan)

    def forward(self, x, edge_index):
        return self.propagate(edge_index, x=x)

    def message(self, x_i, x_j):
        return x_j

    def update(self, aggr_out, x):
        return self.lin1(aggr_out) + self.lin2(x)
```

https://pytorch-geometric.readthedocs.io/en/stable/tutorial/create_gnn.html

Discovering Symbolic Models from Deep Learning with Inductive Biases

Miles Cranmer[1] **Alvaro Sanchez-Gonzalez**[2] **Peter Battaglia**[2] **Rui Xu**[1]

Kyle Cranmer[3] **David Spergel**[4,1] **Shirley Ho**[4,3,1,5]

[1] Princeton University, Princeton, USA [2] DeepMind, London, UK
[3] New York University, New York City, USA [4] Flatiron Institute, New York City, USA
[5] Carnegie Mellon University, Pittsburgh, USA

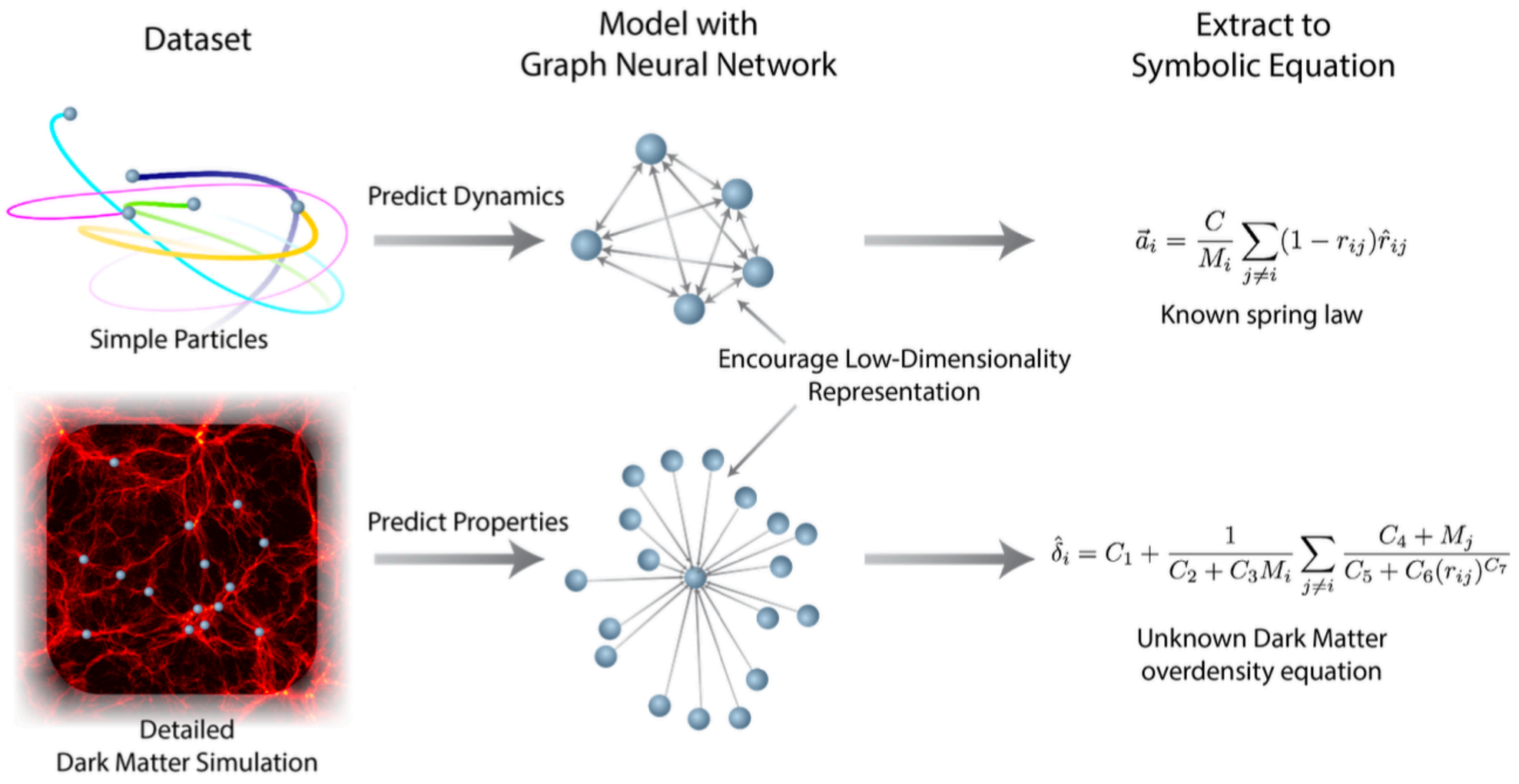
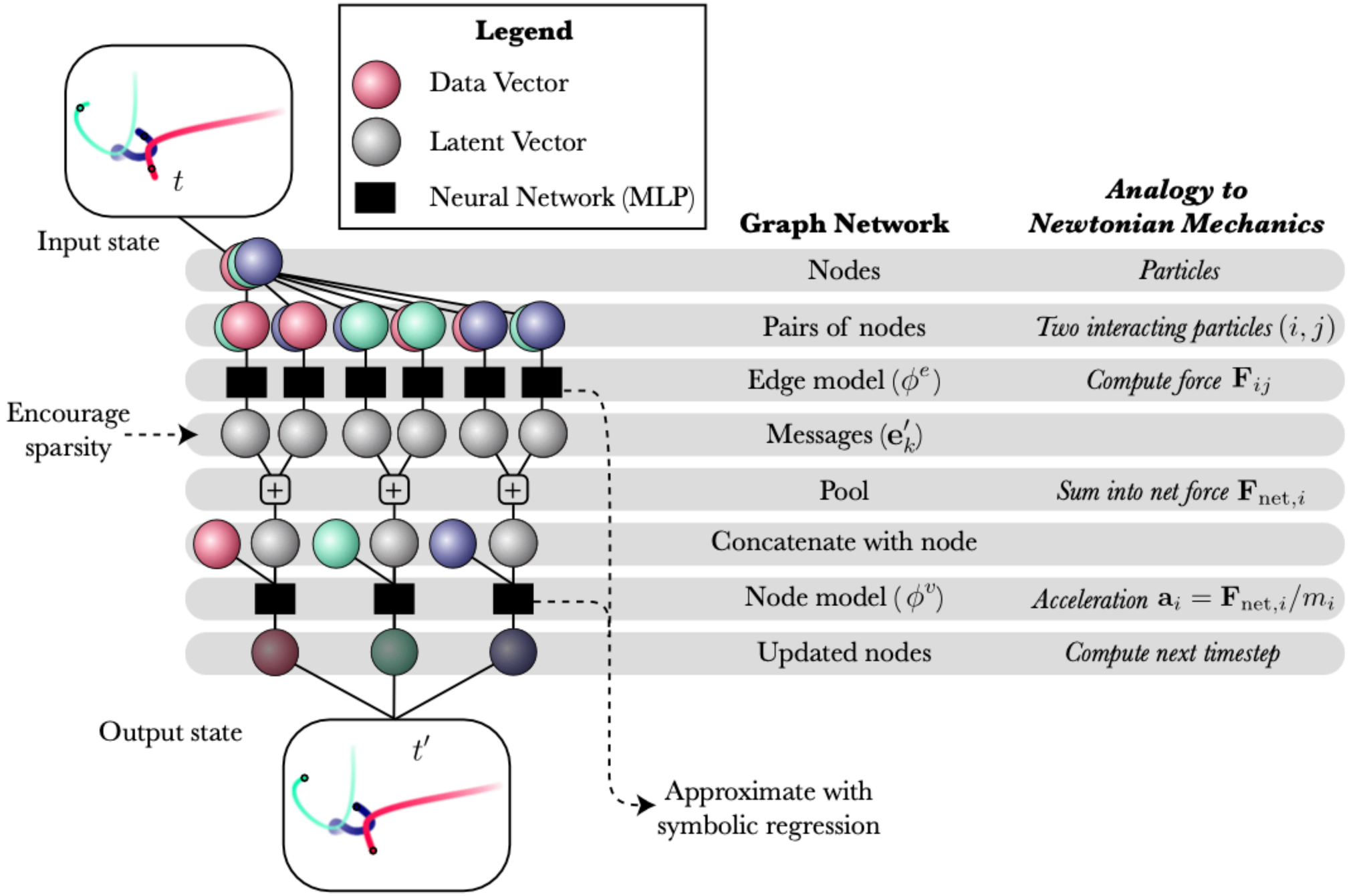
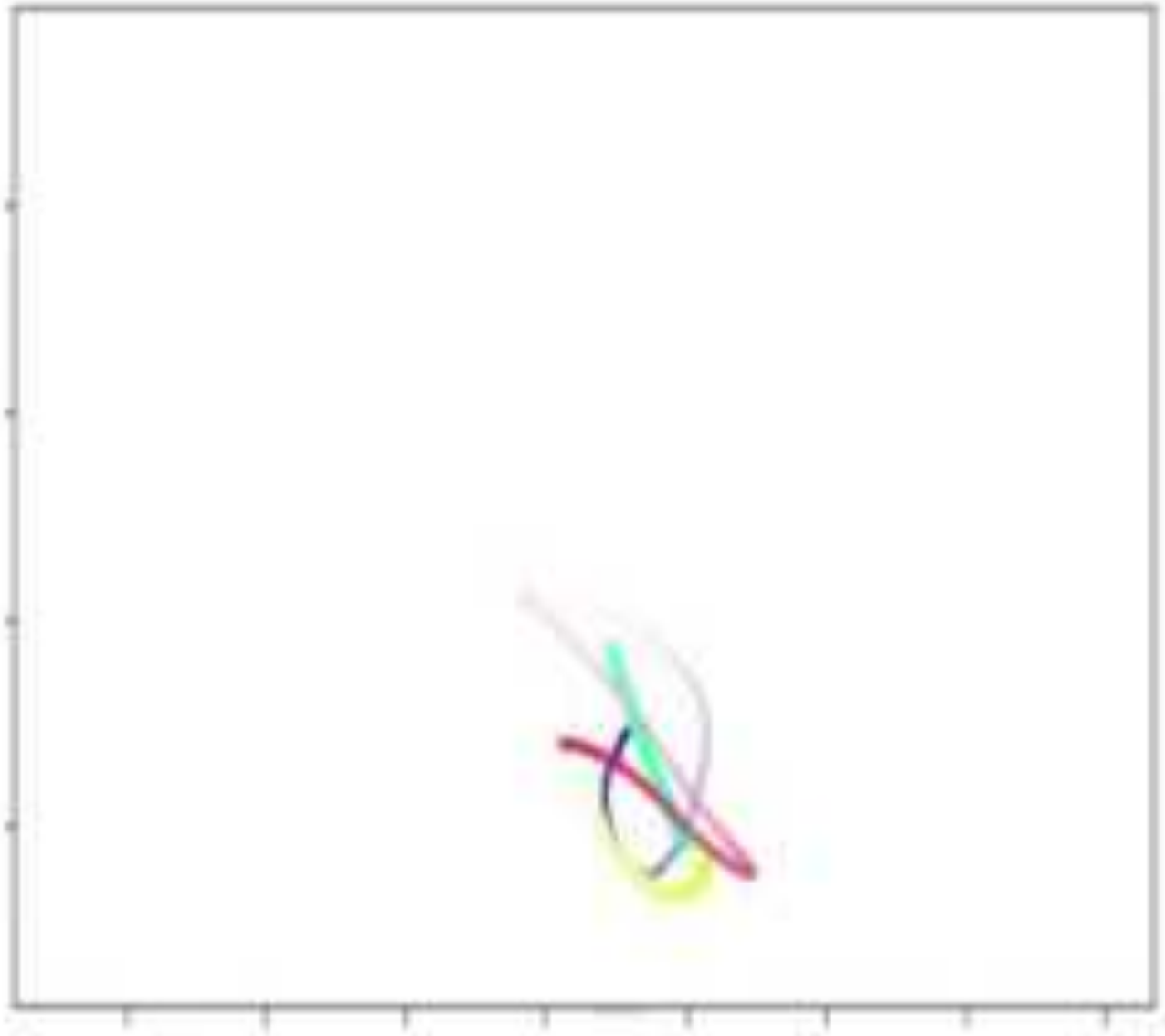
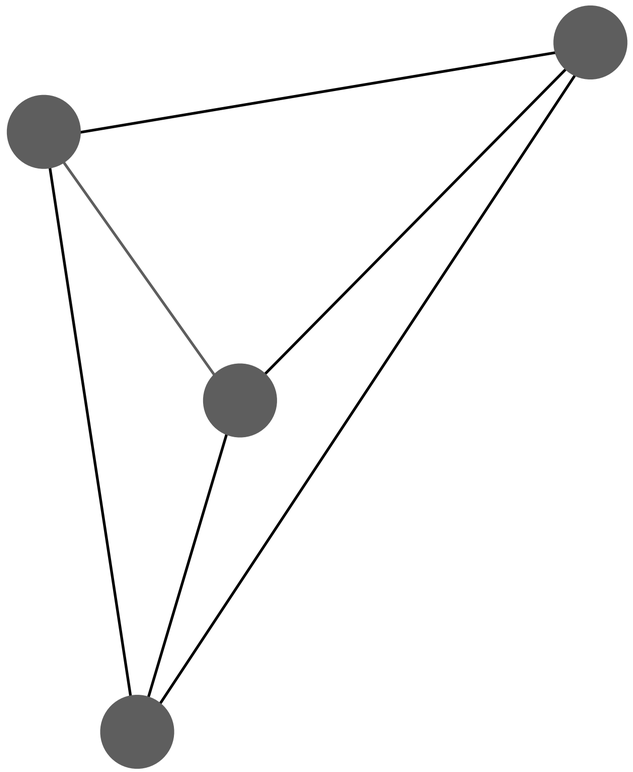


Figure 1: A cartoon depicting how we extract physical equations from a dataset.


Legend
Data Vector
Latent Vector
Neural Network (MLP)
Input state
t
Encourage sparsity
Output state
t'
Approximate with symbolic regression
Graph Network
Analogy to Newtonian Mechanics
Nodes
Particles
Pairs of nodes
Two interacting particles (i, j)
Edge model (ϕ^e)
Compute force $\mathbf{F}_{ij}$
Messages $(\mathbf{e}'_k)$
Pool
Sum into net force $\mathbf{F}_{\text{net},i}$
Concatenate with node
Node model (ϕ^v)
Acceleration $\mathbf{a}_i = \mathbf{F}_{\text{net},i}/m_i$
Updated nodes
Compute next timestep

$$\text{Force} = -(r-1)\hat{r}$$





4 particles, snapshot in time

75,000 training graphs

x = [px, py, Vx, Vy, m]

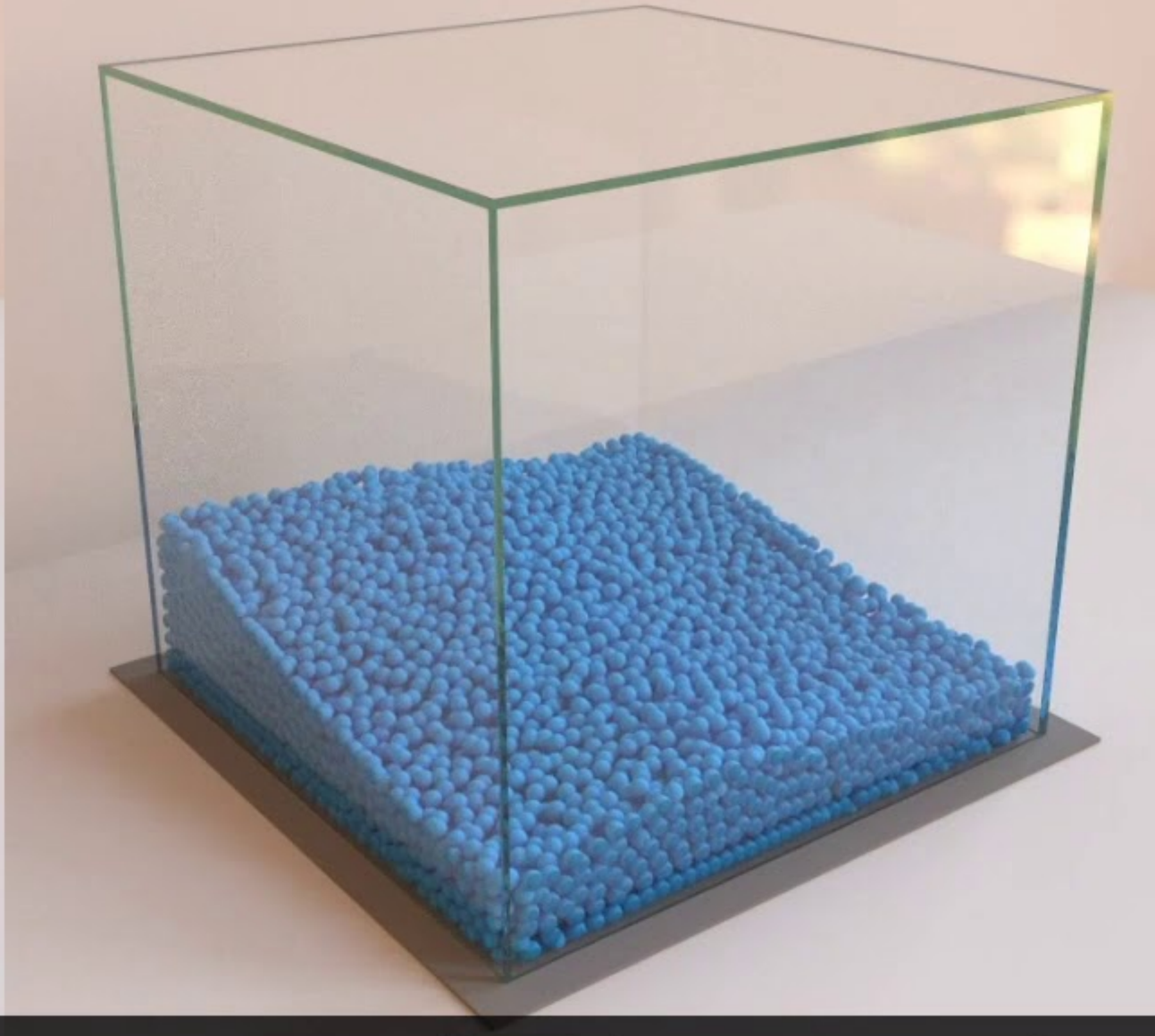
y = [ax, ay]

```python
class GraphNetwork(MessagePassing):

    def __init__(self, aggr='add'):
        super(GraphNetwork, self).__init__(aggr=aggr)

    def forward(self, x, edge_index):
        # x is [num nodes x inchannels]
        return self.propagate(edge_index, x=x)

    def message(self, x_i, x_j):
        # construct messages from all node j's (sources) to all node i's (destinations)
        # (assuming flow="source_to_target")
        # x_i and x_j are both of size [num edges x input channels]
        xcat = torch.cat([x_i, x_j], dim=1)  # pass into NN

    def update(self, aggr_out, x):
        # update my state based on the aggregated messages from my neighbors and my own state
        # x is [num nodes x input channels], xagg is [num nodes x message channels]
        xcat = torch.cat([x, aggr_out], dim=1)  # pass into a different NN
```

Message Passing GNN

Surface mesh

Underlying particles



Particle representation

ETA Prediction with Graph Neural Networks in Google Maps

Austin Derrow-Pinion[1], Jennifer She[1], David Wong[2*], Oliver Lange[3], Todd Hester[4*], Luis Perez[5*], Marc Nunkesser[3], Seongjae Lee[3], Xueying Guo[3], Brett Wiltshire[1], Peter W. Battaglia[1], Vishal Gupta[1], Ang Li[1], Zhongwen Xu[6*], Alvaro Sanchez-Gonzalez[1], Yujia Li[1] and Petar Veličković[1]

[1]DeepMind [2]Waymo [3]Google [4]Amazon [5]Facebook AI [6]Sea AI Lab *work done while at DeepMind

{derrowap,jenshe,wongda,petarv}@google.com

ABSTRACT

Travel-time prediction constitutes a task of high importance in transportation networks, with web mapping services like Google Maps regularly serving vast quantities of travel time queries from users and enterprises alike. Further, such a task requires accounting for complex spatiotemporal interactions (modelling both the topological properties of the road network and anticipating events—such as rush hours—that may occur in the future). Hence, it is an ideal target for graph representation learning at scale. Here we present a graph neural network estimator for estimated time of arrival (ETA) which we have deployed in production at Google Maps. While our main architecture consists of standard GNN building blocks, we