Accelerating Graph Networks for Real-Time Physics Simulation

Mohammad Jasim Usmani

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

January 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mohammad Jasim Usmani**

Entitled: **Accelerating Graph Networks for Real-Time Physics Simulation**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________ Internal Examiner and Chair
Dr. Maria Amer

______________________ External Examiner
Dr. Charalambos Poullis

______________________ Supervisor
Dr. Krzysztof Skonieczny

Approved by ______________________
Yousef R. Shayan, Chair
Department of Electrical and Computer Engineering

__________ 2024 ______________________
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Accelerating Graph Networks for Real-Time Physics Simulation

Mohammad Jasim Usmani

Physics-based simulations analyze interactions between physical objects, especially in areas such as terramechanics to study the interaction between soil particles. By modern standards, most computer simulations run at 60 Hz or 120 Hz refresh rates, equating to real-time step size requirements of 16 ms or 8 ms, respectively. The discrete element method (DEM) models the interactions between the soil particles accurately, but it requires a lot of computational power. Graph Network Simulator (GNS) presents a promising alternative where it uses graph neural networks to learn and approximate the dynamics of the physical system fed into it. Although this method is faster than conventional models, the high dimensionality of the dataset means that the inference happens on a very large graph and rollouts do not occur in real-time. To accelerate the simulation, Haeri and Skonieczny (2021) developed a method that uses dimensionality reduction techniques such as principal component analysis (PCA). This method identifies the top principal components or PCA modes in the dataset that carry the highest variance. The next step is to feed these modes into the GNS along with the rigid body to train the model. Even though the reduced-order dataset results in a much smaller graph than the original dataset, the modified real-time GNS does not use algorithms such as nearest neighbours during adjacency graph construction for training and inference because the interaction between the rigid body and PCA modes may not be proximity-based. The graph fed into the network is naively fully connected. The contribution of this research is to propose a partial graph framework for the GNS to further accelerate this subspace framework without compromising the performance. To identify the redundant connections in the adjacency graph, Neural Relational Inference (NRI) is used. The NRI model, based on a variational autoencoder, uses the encoder to extract a partial graph between the PCA modes. This graph contains the most important edges that can still be used to perform inference on the GNS without any significant loss in accuracy. The effectiveness of this approach has been tested on blade cutting as well as wheel datasets to generate simulation at 60 Hz. This framework has reduced the inference time for excavation blade-driven granular flow from (0.34 - 0.48) seconds per second of simulation to (0.12 - 0.15) seconds and from 0.37 seconds to 0.18 seconds for wheel driven granular flow, achieving approximately 3x and 2x speed up respectively.

Acknowledgments

I would like to extend my heartfelt gratitude to Dr. Krzysztof Skonieczny, my supervisor, for his exceptional guidance and unwavering support throughout the completion of this thesis. Dr. Skonieczny's expertise and mentorship have been instrumental in shaping the direction of this research, and I am truly fortunate to have had the opportunity to work under his guidance.

I am also deeply indebted to my parents Rashid Alam and Zeba Mehfooz for their support, encouragement, and love. Their belief in my abilities and constant encouragement have been a driving force behind my academic journey, and I am grateful for the sacrifices they have made to ensure my success.

I would also like to express my gratitude to my labmates in the Aerospace Robotics Lab. The collaborative and stimulating environment within the lab has not only facilitated the exchange of ideas but has also contributed to the overall success of this research project. The shared passion for innovation and the collective pursuit of knowledge have made the research process both fulfilling and intellectually rewarding.

A special acknowledgment is also due to CM Labs Simulation for their generous research funding and invaluable technical guidance. The support provided by CM Labs Simulation have significantly enriched the depth and scope of this research, and their commitment to advancing innovation in the field has been a driving force behind the success of this thesis.

This thesis is a testament to the collective support and encouragement I have received from Dr. Krzysztof Skonieczny, my family, CM Labs Simulation, and my labmates. Each has played a crucial role in shaping my academic and personal growth, and for that, I am truly thankful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Space exploration missions to celestial bodies such as moon and mars require planetary rovers for exploration. These rovers operate on an unusual terrain with different gravity levels to understand and map the terrain. Understanding the wheel-terrain interaction on these celestial bodies is essential as this helps in designing the rover wheels and installing the right tools to carry out experiments for exploration. In terrestrial applications, understanding the dynamic interaction between construction machinery wheels and the soil is essential for the development of real-time simulators. This type of interaction model is deployed in the construction of simulators by CM Labs Simulations which are designed for training operators. To model the interaction between the martian and lunar soil, Discrete Element Method is a promising approach with high accuracy. However, simulating granular flow and physics-based simulations are generally computationally intensive and expensive. These simulations necessitate a prior understanding of the underlying physics of the environment and objects involved, posing challenges in developing real-time simulators with both accuracy and efficiency. Specifically, modelling situations like robot-terrain interaction and soil interaction with excavators is particularly difficult. Real-time simulators are crucial for visualizing the interaction of granular particles with various objects, however, achieving computational efficiency remains a significant obstacle.

The Mars Rover Spirit encountered a challenging situation as it became immobilized and trapped in a soft regolith terrain. When faced with the entrapment of vehicles in extraterrestrial environments, the application of real-time simulators emerges as a valuable tool for informed decision-making. These simulators can also be used for the dynamic recreation of scenarios wherein the rover's wheels can be steered at varying angles, aiding in the exploration of optimal strategies to steer the rover out of its entrapment.

In the case of planetary exploration and rover design, modelling the impact of gravitational forces on granular flow dynamics, particularly the interaction with rigid structures like wheels or excavation blade is very essential in understanding and developing wheel designs. Nonetheless, the validation of these design concepts through physical experiments is challenging which requires parabolic flights to simulate microgravity conditions. Previously, Skonieczny, Niksirat, and Nassiraei (2017) have extensively worked on developing and testing the effect of gravity on wheel using a single wheel test bed in a parabolic flight to study the effect of gravity (Niksirat, Daca, & Skonieczny, 2020; Skonieczny, Niksirat, & Nassiraei, 2019). In this context, real-time simulators emerge as a highly promising solution for evaluating the performance of various wheel designs in challenging terrains akin to those encountered on celestial bodies such as Mars or the Moon. These real-time simulators can facilitate the systematic exploration of diverse wheel configurations and their response to complex terrain conditions while avoiding the need for repeated physical experimentation.

Construction machinery frequently interacts with granular substances through their wheels and excavation tools. The simulation of these interactions for operator training is also a very important application of real-time simulators. This research is facilitated by an industrial partner, CM Labs Simulations, a company that creates training simulators for a wide range of heavy machinery, including various types of construction vehicles. The precision and real-time simulation of the interaction between machines, terrain and granular particles is vital for the visual accuracy of these simulators for training operators.

1.2 Problem Statement

Modelling granular flow interactions for large scale particles is computationally expensive and often time consuming. With advancements in neural network architectures, models can learn complex dynamics and interpret intricate patterns within a dataset. In the case of granular flow, interactions can be modelled as a graph where each particle represents a node. Graph network simulator (GNS) (Sanchez-Gonzalez et al., 2020) uses graph neural networks to learn and approximate the dynamics of the physical system fed into it. Even though this is faster, due to the high dimensionality of the dataset, the inference is performed on a very large graph and rollouts are not in real-time. To accelerate the simulation, one way developed recently in our laboratory is to use dimensionality reduction techniques such as principal components analysis to identify the modes in the dataset that carry the highest variance (Haeri & Skonieczny, 2021). These reduced-dimension modes are then fed into the GNS along with the rigid body to train the model. Although the reduced-order dataset results in a much smaller graph than the original dataset, the graph fed into the network is still naively fully connected because the relationship between the particles representing the rigid body in full-space and reduced soil particles in sub-space (PCA modes) is unknown. In the development of real-time simulators, which are powered by machine learning techniques and operate at a refresh rate of 60 Hz, the primary criterion is to achieve a minimal response time of 16 milliseconds and a low positional error when compared to the ground truth. Additionally, the force metric is of significant importance, particularly in scenarios where the simulator is required to provide haptic feedback to the user. This research aims to further **accelerate** this subspace framework **by identifying a partial graph framework** for excavation and wheel driven granular flows using the graph network simulator without any significant compromise on the positional accuracy of the particle predictions.

1.3 Related Work

Terramechanics-based models are computationally efficient due to their simplicity, however, they are not considered as of the highest fidelity due to some of their assumptions. In the case of Bekker-Wong model, soil dynamics or wheel dynamics are not considered which limits its ability to capture the complexities of the wheel-soil interaction (McCullough, Jayakumar, Preston-Thomas, Hodges, & Shoop, 2017). These models do not offer a consistent mechanism for accurately representing how terrains respond to varying gravitational conditions, despite some attempts over the years (Kovács et al., 2019; Wong, 2012). In the Discrete Element Method (DEM) approach, soil is represented as a group of discrete particles that interact with each other and can also model the interaction with a rigid body such as a wheel, directed by the principles of physics. DEM is widely acknowledged for its superior accuracy in terramechanics modeling, but has a higher computational demand (Karpman, Kövecses, Holz, & Skonieczny, 2020). Gravity is explicitly incorporated within its formulation, which makes it well-suited for modelling the soil behaviour in environments characterized by gravity (Nakashima, 2011).

Material point method (MPM) uses a hybrid computational approach that combines elements of both Eulerian and Lagrangian frameworks. These materials can be represented as points within a stationary computational grid which has the capacity to simulate physical motion and shape deformation based on the laws of physics. Haeri and Skonieczny (2022) implemented a 3D material point method with non-local granular fluidity (NGF-MPM) which precisely computes the internal forces and was also validated using lab experiments. This high fidelity model is able to generate rigid body interaction with granular particles such as soil with very high accuracy. Specifically, it focussed on generating simulation for excavation driven granular flow at various angles and depths as well as wheel driven granular flow with varying wheel diameters and with varying gravity levels. Although DEM- and MPM-based methods offer high accuracy they are generally not real-time. For the case of NGF-MPM, it took around 270 sec to generate one second of simulation which clearly does not make it usable for real-time applications (Haeri & Skonieczny, 2021).

Accelerating simulations using machine learning. With the advancements in neural network architectures, the generalization abilities of deep learning models have improved exponentially. These architectures have the ability to extract and learn the dynamics of the input data. These neural networks are capable of learning and simulating physical interactions many orders of magnitude quicker than conventional methods as the machine learning algorithm comprises of model architecture and the trained weights of the network. The training process involves the utilization of a substantial volume of simulation data, which is fed into the network to facilitate learning. Through this training, the network acquires an understanding of the complex dynamics embedded in the simulation data. During the inference phase, the trained neural network serves as a fast emulator, approximating the physics solver and predicting future interactions within the environment. In a study by Kasim et al. (2021), rapid emulators were developed using convolutional neural networks for diverse applications, including astrophysics, seismology, and biogeochemistry. Employing a novel method based on neural architecture search, the authors achieved a runtime acceleration of 2 billion times when running the inference step using graphics processing units (GPUs). To investigate real-time generation of physical interaction between various objects such as cloth based interactions in a gaming environment, Holden, Duong, Datta, and Nowrouzezahrai (2019) utilized a multi-layer perceptron (MLP) which was trained on synthetic data generated through a physics solver. They reduced the dimensionality of the dataset using principal component analysis (PCA) to mitigate high space complexity of the model. The model was trained to predict the next time frame based on a window of the dataset, resulting in acceleration ranging from approximately 300 to 5000 times for various physical interactions such as object collisions and cloth movement. To accelerate the computations within MPM, J. Li et al. (2023) introduced a new approach for learning the parameters using neural networks to predict pressure fields between the particles and the grids within MPM. MPMNet architecture contains three blocks in their architecture, an encoder followed by a convolutional long short-term memory block and a decoder which led to an acceleration by 28 times. A novel formulation of the transformer architecture specifically for generating particle based simluations was developed by Y. Shao, Loy, and Dai (2022). The core idea of transformer with implicit edges (TIE) is the decentralization of paired particle interaction calculations, which is achieved by means of per-particle updates. By adapting the self-attention module so that it imitates the update method used for graph edges in Graph Neural Networks (GNNs), the model decentralizes the predictions. TIE stands out as a potentially useful method for modelling intricate particle interactions without the need for clearly defined edges which makes it computationally efficient when generating rollouts on large number of particles and reported lower mean squared error values as compared to graph network simulator (Sanchez-Gonzalez et al., 2020).

Graph networks. Architectures based on geometric machine learning (Gilmer, Schoenholz, Riley, Vinyals, & Dahl, 2017) have been successful in modelling the particle-particle interaction by making the model learn the underlying physics by using graph neural networks using message passing. Here, the information from the neighbouring nodes in the graph are aggregated together which then used to update the state of each node in the graph. The graph network algorithm for message passing which was developed by Battaglia et al. (2018) can capture the dynamics with higher accuracy as it updates the global features along with the node and edge attributes. The graph network simulator (GNS) as proposed by Sanchez-Gonzalez et al. (2020) is based on the graph network architecture for capturing the physical dynamics of examples such as sand collision and water splash. The GNS uses a segment of the particle positions as a time series input and the GNS learns to predict the position of the particles in the next time step. Z. Li and Farimani (2022) modified the graph network formulation to simulate fluid by using two different GNN blocks namely node-focussed networks which are used to learn the advection while the edge-focussed networks learn the elastic collision. This results in a lower number of parameters in the network, thus making them around 8 times faster in generating predictions as compared to GNS. H. Shao et al. (2021) improved the performance of the position-based dynamic solver where the correction step is solved iteratively for simulating rod dynamics and requires initial guesses. To improve the accuracy, the graph network is employed to make initial guesses to generate the simulation which led to a better run time performance. In order to better simulate physical system, a graph network with neural ordinary equation was developed as tested on gravity and coulomb based particles which was compared against GNS where it performed slightly better with lower RMSE values (Shi, Zhang, Jin, Pan, & Yu, 2023).

Pfaff, Fortunato, Sanchez-Gonzalez, and Battaglia (2021) proposed modifications to graph neural network to adapt them to learning mesh-based simulations such as moving cloth, flag and even aerodynamics. Using message passing, the network successfully extracts the dynamics using encode-process-decode architecture (Battaglia et al., 2018) and beat all the baselines in terms of accuracy and rollout performance. The study by Lino, Fotiadis, Bharath, and Cantwell (2022) introduces two novel graph neural network (GNN) architectures multi-scale-GNN (MS-GNN) and rotation-equivariant-multi-scale-GNN (RE-MS-GNN) designed explicitly for learning fluid flow dynamics to predict future interactions. The RE-MS-GNN incorporates rotational equivariance in its model design, significantly improving the network's capacity to learn underlying dynamics and accelerating simulations by four to five times compared to conventional simulation methods. A study by Allen et al. (2023) debunked the assumption that graph network based models do not have the ability to learn the rigid body dynamics without the aid of some contact model. This assertion is examined through experiments conducted on well-established real and simulated datasets, revealing that general-purpose graph network simulators, without any type of contact-specific assumptions still possess the capability to generalize and predict accurate contact discontinuities. Interestingly, compared to finely tuned robotics simulators, the contact dynamics obtained by graph network simulators show a slightly higher degree of precision in the case when replicating actual cube throwing trajectories.

While graph neural networks offer the potential to learn complex systems and accelerate conventional simulation methods, there remains a computational challenge associated with graph formation for modelling dynamics. This challenge becomes evident when simulating systems with a substantial number of particles, resulting in a notably large graph structure. This, in turn, requires higher computational power when employing the network for inference. To address this (Haeri & Skonieczny, 2021; Holden et al., 2019) used principal component analysis (PCA) to reduce the size of the dataset fed to neural network using PCA to accelerate the inference time. Haeri and Skonieczny (2021) used the NGF-MPM (Haeri & Skonieczny, 2022) to generate the training dataset for wheel and excavation driven granular flow. With PCA, nearly 6000 particles were reduced to 8 PCA modes which captured the maximum variance in the dataset. The position of the PCA modes and the particles representing the rigid-body along with the force acting on the rigid-body was fed to the GNS during training. Thus, the smaller graph structure led to a real-time rollout which was almost 700 times faster than the original NGF-MPM implementation.

However, the graph formation in this modified version does not have any specific formulation and uses a fully connected graph. As the particle interaction in the subspace may not be based on nearest neighbor, some edges in the graph maybe redundant and not contribute in the graph learning the process. Therefore, finding a relationship between the principal components and rigid body can reduce the time and space complexity when training as well as testing the model. One way is to use Neural Relational Inference (NRI) Kipf et al. (2018) which is a variant of variational autoencoder models, characterized by latent spaces structured as graphs. This particular model finds application in the prediction of particle trajectories. The fundamental methodology employed by NRI involves leveraging the encoder of a variational autoencoder to probabilistically generate a graph by utilizing the provided data trajectory as input. This process entails the utilization of a graph neural network as the encoding mechanism. Subsequently, the resultant interaction graph functions as a latent variable denoted as z, which is subsequently integrated into the decoder. Here, z_{ij} signifies the presence or absence of an edge between nodes i and j in the graph. Using the trained encoder, we can sample out the latent space to reveal the most important edges in the graph which can then be used in training graph network simulator (GNS) for faster inference without any significant drop in accuracy.

Optimizing graph networks. The rising complexity of advanced Deep Neural Network (DNN) models has created considerable hurdles in the area of performance optimization, particularly in the form of large memory use and increased processing requirements. As a result, these complications have made the real-time deployment of such models quite challenging. In order to accelerate inference time in deep neural networks, model compression techniques such as pruning and quantization are some of the most common approaches. Pruning is a method for compressing models that involves removing specific neurons from neural networks with an emphasis on disregarding the neurons with the lowest weight magnitudes (Z. Li, Li, & Meng, 2023; Tailor, Fernández-Marqués, & Lane, 2020). The pruning process typically follows a sequence: the initial training of the base model, the subsequent ranking of model weights to identify the most significant ones, and the removal of weights falling below a specified pruning threshold. The pruned model is then subjected to re-evaluation and further training. However, the higher the pruning threshold the higher the potential drop in accuracy.

Quantization is also a common method for model compression where the values of the weights and the biases are reduced to a lower precision value. Reducing the precision of the model weights leads to a lower memory requirement and faster inference time (Z. Li et al., 2023; Tailor et al., 2020). A dynamic pruning method was introduced by Liu et al. (2022) for Graph Neural Networks (GNNs) which is integrated within training procedure. Unlike other methodologies, comprehensive graph pruning (CGP) integrates the pruning process by eliminating redundant connections during the model training due to which the need for retraining is eliminated, significantly reducing computational costs. The co-sparsifying technique introduced by CGP addresses the optimization of three primary components in GNNs: graph structures, node characteristics, and model parameters. This approach not only eliminates the node feature dimensions but also streamlines both the graph structure and model parameters. Along with this, CGP incorporates a regrowth process into its methodology, ensuring the efficacy of the pruning operation. Through reintroducing the pruned connections which were crucial for the performance of the GNN, this procedural step safeguards the model's functionality from any adverse effects resulting from the pruning process. Gao et al. (2022) introduced a new approach to accelerate the training and inference when a sizable graph dataset is being used in a GNN model. In GNNs, each node in the graph must spread its characteristics to its neighbours during the inference process. However, since the same characteristics may propagate more than once via several pathways, the conventional propagation order may result in duplicate calculations. Every node has a different propagation order using the adaptive propagation order approach, which is determined by its topological information. A node with a high degree, for example, one that is linked to a large number of other nodes, may see its characteristics propagate sooner than a node with a low degree. This guarantees that the most significant characteristics spread first, cutting down on needless calculations and thus accelerating the inference process.

1.4 Contributions

The thesis makes the following contributions:

1. Identifies a partial graph framework for Subspace Graph Network Simulator (GNS), assisted by the **Neural Relational Inference** (NRI) method for the graph network as input that keeps most of the information.

2. Accelerates the real time performance of the graph network simulator (GNS) using the partial graph framework for simulating the dynamics of granular flows propelled by rigid bodies and the interacting forces on a standard GPU (NVIDIA GeForce RTX 3080) with 10 GB VRAM.

1.5 Thesis Outline

The structure of the thesis is as follows: Chapter 2 is focused on the graph neural networks as well as the graph network architecture with the modifications made to adapt it to real time simulator for granular flows. It also addresses the dataset used for the GNS as well as NRI and mentions the dimensionality reduction of the dataset using PCA and kernel PCA. The chapter concludes by discussing the existing framework of the graph network simulator, specifically formulated for simulating real-time rigid body-driven interactions in granular flows. In Chapter 3, the focus shifts to Neural Relational Inference (NRI) model, its implementation and the formulation of partial graph framework for accelerating the GNS for the case of excavation and wheel-soil interaction. The chapter ends by discussing the results both quantitatively and qualitatively. Finally, the thesis document concludes in Chapter 4 by discussing the limitations of using partial graph with GNS for training and inference along with suggestions for future works. The source code is available on GitHub[1].

1.6 Publication

1. Mohammad Jasim Usmani and Krzysztof Skonieczny,Accelerating Graph Networks for Real Time Physics Simulation", 6th European - African Regional Conference of the ISTVS, Lublin, Poland, October 2023.

https://github.com/jasimusmani/acc_ssgp[1]

Chapter 2

Related Work: Real-Time GNN based Simulation

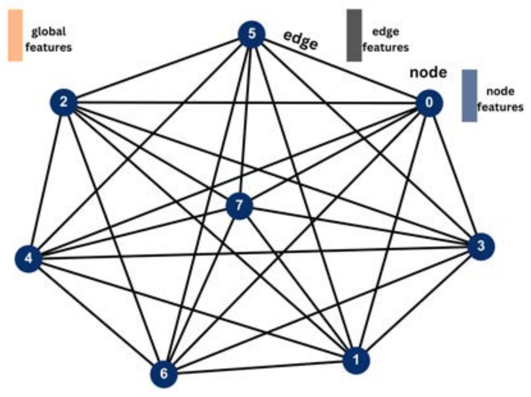
Modelling the physical dynamics of granular flows is very computationally intensive and is commonly approached by Discrete Element Method (DEM) as well as continuum based methods. As these models are no where near real-time, their use case becomes limited. To address the challenge of modelling the dynamics, a novel approach based on Graph Neural Networks was introduced by Sanchez-Gonzalez et al. (2020) to simulate physics based interactions. They employed a graph network architecture proposed by Battaglia et al. (2018) to learn the underlying dynamics of various physics simulations involving substances like sand, water, and goop. The proposed Graph Network Simulator resulted in accurate rollout performance as well as generalization. Even though the GNN based approach reduced the time complexity, this was still far from real-time rollout. To tackle the problem of real-time simulation for wheel-terrain interaction as well as excavation blade-driven granular simulation interaction, Haeri & Skonieczny (2021) proposed to reduce the granular data to reduced PCA modes and fed these to the Graph Network along with the particles representing an interacting rigid body. A similar approach was previously proposed by Holden et al. (2019) for using PCA modes and training neural networks on subspace dataset to generate real-time simulations of different objects for video games. Using subspace data instead of full space data significantly decreased the space complexity which reduced the graph size significantly. This led to real time rollout and up to 300 to 5000 faster times than the conventional simulation method.

In section 2.1, this chapter shall elucidate the contextual underpinnings of graph neural networks. Subsequently, section 2.2 shall undertake an explanation of the architectural aspects of graph networks as applied to physics-based simulations (Sanchez-Gonzalez et al., 2020). Further, section 2.3 will provide a detailed explanation of the dataset employed in the training of the Graph Network Simulator (GNS), while section 2.4 will delve into the theoretical details of Principal Component Analysis (PCA) in section 2.4.1. This segment shall also encompass an exploration of kernel PCA in section 2.4.2 and a critical analysis of its inapplicability within the present context. In section 2.5, our focus shall be directed towards the adaptation of the graph network simulator originally introduced by Haeri and Skonieczny in 2021 which will be dedicated to the analysis of the training and inference processes associated with the modified GNS.

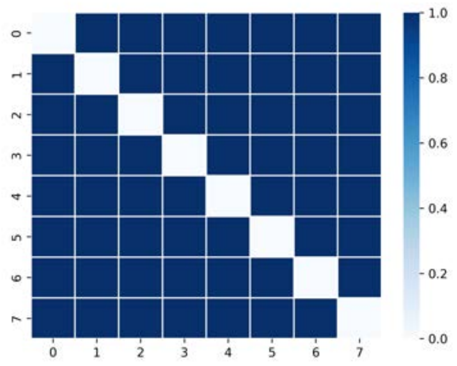
2.1 Graph Neural Networks

Graph is data structure that is usually formulated by nodes and edges where the nodes are connected to edges depending upon their relationship. Some naturally occurring graph based data structures could be elemental structure of the chemical compounds where the nodes are the elements and the edges represent the bond between them. Even the interaction of physical systems such as wheel-terrain interaction can be categorized in the form of graph data structure where the nodes can be the soil particles while the edges can be the distances between the particles (Sanchez-Gonzalez et al., 2020).

In the context of graph theory, a fundamental means of characterizing a graph is by means of its adjacency matrix. This matrix serves as a concise representation wherein the rows and columns are uniquely associated with the nodes constituting the graph. Within the adjacency matrix, entries may assume one of two forms: binary, denoting the presence or absence of a connection between nodes, where '1' signifies an established connection, and '0' refers to the absence of connection; or weighted, where entries assume non-binary values, often representing the significance or strength of the connections.



(a) A fully connected graph with 8 nodes



(b) Adjacency matrix

Figure 2.1: Sample Graph

It is crucial to distinguish the graph data structure from other modalities such as images when training with simple neural network for training. Unlike images, the adjacency matrix of the graph does not work well if we try to directly flatten the adjacency matrix to feed into neural network architectures. Attempting to do so would result in a network architecture reliant on the specific ordering of elements within the adjacency matrix. Consequently, such a network would inherently lack permutation invariance or equivariance, making it unsuitable for tasks that require robustness to changes in the order or arrangement of graph nodes. This structural difference in graph based data structures necessitates specialized framework for effectively modelling graph based data within neural network frameworks.

Graph Neural Network (GNN) utilizes a neural message-passing mechanism, wherein vectorized messages are interchanged among nodes within the graph and these messages are updated using neural networks. For a graph G where u, v and e are global, node and edge features respectively. h_u, h_v and h_e are global, node and edge embeddings respectively. The message passing can be formally described as follows:

$$\text{mssg}_{N(v_i)}^{(k)} = \text{AGGREGATE}^{(k)}\left(h_{v_i}^{(k)} \,\middle|\, \forall v_i \in N(v_i)\right) \tag{1}$$

$$h_{v_i}^{(k+1)} = \text{Update}^{(k)}\left(h_{v_i}^{k}, \text{AGGREGATE}^{(k)}\left(h_{v_i'}^{k} \,\middle|\, \forall v_i' \in N(v_i)\right)\right) \tag{2}$$

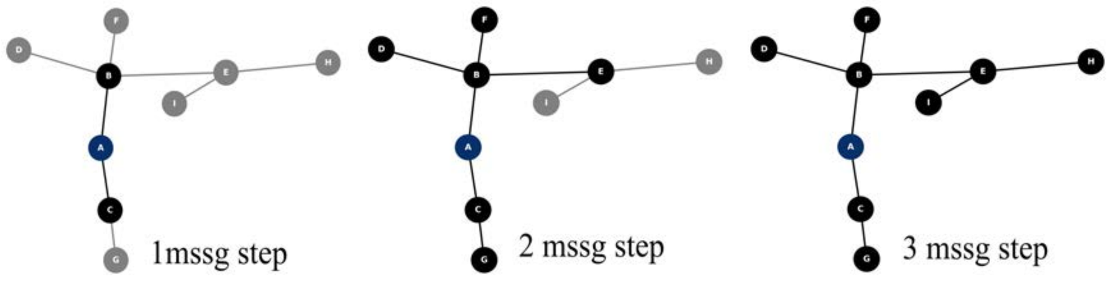


Figure 2.2: Message passing steps in a graph

To generate a message for a node v_i the message passing function aggregates the features from all the the neighbouring nodes using a neural network while the update the function also uses a neural network to update the node embedding. As the order does not matter during the process of aggregation, this formulation is permutation invariant and equivariant. The number of message passing step decides the depth of the graph when aggregating the messages from the neighbouring nodes. Higher message passing steps leads to accumulation of messages from farther nodes as shown in figure 2.2.

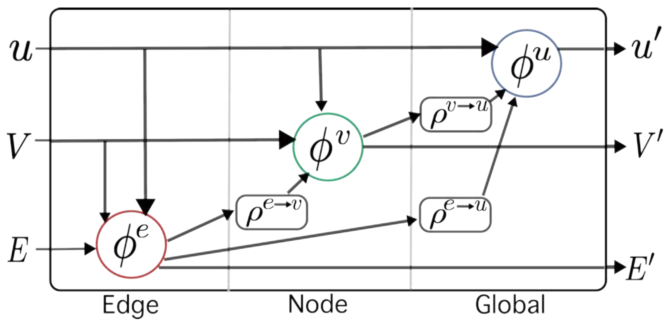
2.2 Graph Network Architecture

The Graph Network Simulator (GNS) utilizes graph networks (GN) (Battaglia et al., 2018) to learn physics and extract underlying dynamics from the training data. In this context, the granular particles can be represented as nodes in a graph, while the edges symbolize the interactions between these particles. The graph network is a message passing framework which simultaneously generates the edge embedding, node embedding as well as the global embedding.

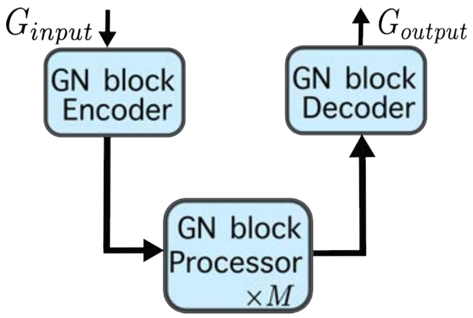
The GN performs the update step in a sequence. It first compute the updated edge attributes (e'_k) followed by the per node aggregation of edge features ($\overline{e}'_i$).

$$e'_k = \phi^e(e_k, v_{rk}, v_{sk}, u) \tag{3}$$

$$\overline{e}'_i = \rho^{e\to v}\left(E'_i\right) \tag{4}$$



(a) Graph Network Block



(b) Encode-Process-Decode

Figure 2.3: Graph Network Architecture

Using the aggregated edge attributes, it then updates the node attributes (v_i') followed by global edge aggregation ($\overline{e}'$).

$$v_i' = \phi^v(e_k', v_i, u) \tag{5}$$

$$\overline{e}' = \rho^{e\to u}\left(E'\right) \tag{6}$$

The node features($\overline{v}'$) are aggregated globally followed by the final global feature update (u').

$$\overline{v}' = \rho^{v\to u}\left(V'\right) \tag{7}$$

$$u' = \phi^u(\overline{e}', \overline{v}', u) \tag{8}$$

where $E_i' = \{(e_k', r_k, s_k)\}_{r_k=i, k=1:N^e}$, $V' = \{(v_i')\}_{i=1:N^v}$ and $E' = \bigcup_i E_i' = \{(e_k', r_k, s_k)\}_{k=1:N^e}$. Here, r and s are the receiver and sender respectively.

The GNS uses the GN algorithm to learn and update the features associated with the nodes and edges. The GNS has three modules which are the encoder, processor, and decoder which are built with the graph network architecture to embed, aggregate, and update the node, edge and global features. Here, multi-layer perceptron (MLP) is used with rectified linear unit (ReLU) as the activation function. Here, each MLP has two layers where the output layer has a size of 128 for the case of encoder and processor. The decoder has an output size of 6 as it predicts the three dimensional acceleration and force.

Encoder: The encoder's role is to represent the initial states of the particles as a latent graph, with the particle positions serving as node features and the displacements between sender and receiver nodes encoded as edge features. Graph construction is accomplished using the nearest neighbour algorithm, creating connections between sender and receiver nodes based on proximity.

Processor: The processor uses the interaction network as proposed by (Gilmer et al., 2017) which is a subset of the GN algorithm to perform the message passing and update the initial graph by the encoder. Multi layer perceptron is used for update and aggregation function in the network and is permutation invariant.

Decoder: The decoder takes the updated graph from the processor to further decode the dynamics by predicting the position of the particles. To promote generalization and avoid error accumulation during inference, random-walk noise is added to the input training data to ensure that the data distribution in the input is like the one in the rollout. The noise is sampled from a normal distribution with zero mean and 6×10^{-4} as the standard deviation which is hardcoded in the GNS codebase (Sanchez-Gonzalez et al., 2020).

2.3 Dataset

In order to train the graph network simulator, dataset containing the particle trajectories are generated using the MPM method as implemented by Haeri and Skonieczny (2022). The dataset consists of 250 examples of excavation driven granular flow of two different sizes of excavation blades, one which is represented as 15 particles arranged in a 3×5 array and 20 particles arranged in a 4×5 array. The wheel dataset contains 243 examples where the wheel is represented as two circular concentric circles represented by 8 particles each as shown in figure 2.4. The dataset is partitioned into training, testing, and validation sets with proportions of 80%, 10%, and 10%, respectively. The parameters of the examples of the training and testing dataset is a combination of the parameters as show in table 2.1.

Table 2.1: Details of the excavation and wheel dataset. Here Motion 1 and 2 refers to discrete ramp-horizontal-ramp motion and continuous curved motion respectively (Haeri & Skonieczny, 2022).

Excavation Parameter	**Values**
Angle [deg]	0, 4, 10, 31, 45
Depth [cm]	2, 4, 5, 8, 10
Speed [cm/s]	1, 4, 8, 10, 15
Motion	1, 2

Wheel Parameter	**Values**
Gravity [m/s^2]	1.62, 3.72, 9.81
Friction [deg]	30, 37, 43
Load [N]	100, 164, 225
Diameter [cm]	5, 15, 30
Slip [%]	20, 40, 70

The dataset is fed in the form of tensors to the GNS and have 321 time steps per example. The node features are positions of the soil particles along with the particles representing the excavation blade and the forces on the rigid body. The position tensor is arranged in the shape (321, 4895, 3) when the rigid body is represented by 15 particles and by (321, 4900, 3) when the rigid body is represented by 20 particles. Here, 321 is the time steps, 4880 are the number of soil particles in the simulation and 3 refers to the 3 dimensions of the position in x, y and z direction. The force acting on the excavation blade is represented as (321, 3) where 321 is the time steps and 3 is the dimensions in x, y and z direction.

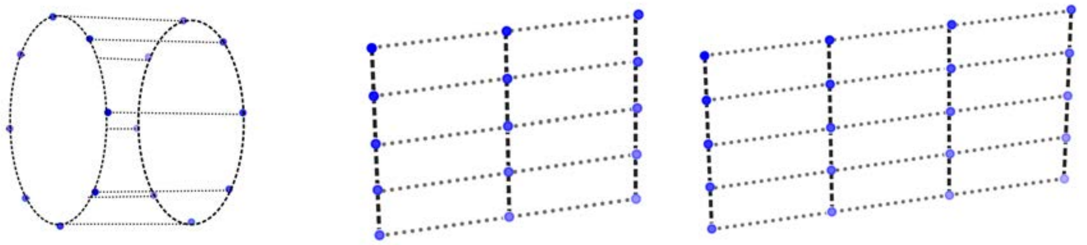

(a) (b) (c)

Figure 2.4: Particles representing rigid body (dotted lines are for visualisation)

Similarly, in the case of wheel, the node features are position and forces which are in the shape (321, 6021, 3) where 321 is the time steps or the number of frames, 6005 are the number of soil particles in the simulation along with 16 particles representing the wheel and 3 refers to the 3 dimensions of the position in x, y and z direction. The force acting on the wheel is represented as a tensor of shape (321, 3) where 321 is the time steps and 3 is the dimensions in x, y and z direction. Along with node features, the wheel example also contains global features which indicates the gravity value and the wheel diameter for each example.

2.4 Dimensionality Reduction

The main objective of the graph network simulator (GNS) is to capture the underlying dynamics between the rigid body and soil particles. The soil particles constitute to more than 90% of the dataset and the graph size is directly proportional to the number of particles. Here, having all the particles as nodes makes the graph very large and leads to large graphics processing unit (GPU) memory requirements for training the GNS. Moreover, the rigid body does not interact with all the soil particles leaving majority of them still throughout. Therefore, in order to reduce the space complexity of the model and to reduce the inference time, Haeri and Skonieczny (2021) used a fully connected graph instead of using nearest neighbour for graph construction and applied principal component analysis on the soil dataset and reduced them to pca modes.

2.4.1 Principal Component Analysis

Principal component analysis (PCA) is an unsupervised linear dimensionality reduction technique which maps a high dimensional dataset to lower dimension by keeping the maximum variance. This is achieved through the selection of a set of orthogonal axes known as principal components which can be identified by the computing the eigenvalues and eigenvectors. The primary principal component captures the most substantial variation in the data, while the subsequent principal components capture the maximum orthogonal variance. PCA has various applications such as dimensionality reduction, feature extraction and compression. In the case of soil particles, PCA acts a feature selection method where the PCA modes that captures most of the variance in the dataset are selected. The idea behind PCA is that valuable information is embedded within the variability of individual features. Thus, when a feature exhibits higher levels of variation, it inherently carries a more significant information. Haeri and Skonieczny (2021) used eigen value decomposition to compute the principal components. For a data $X_{m\times n}$, steps for PCA are shown below:

Algorithm 1 Principal Component Analysis for granular flow data reduction [Hotelling (1933), Haeri and Skonieczny (2021)]

```
Input: $X$
Output: $W$
1: $\bar{x} = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}$
2: $Y = X_{m\times n} - \bar{x}_{m\times n}$
3: $C_{n\times n} = \frac{1}{n-1}Y^TY$
4: $Cw_i = \lambda_i w_i$ (computing the eigenvalues and eigen vectors)
5: $W = [w_1, w_2, ...., w_n]$ (Sorted based on descending order of eigenvalues)
```

The eigenvectors (W) are arranged in the descending order of eigenvalues. The main advantage of using PCA is the ease of mapping the dataset back which is essential in the case of simulating granular flow interactions for accurate visualization. However, it is not able to capture the non-linearities present in the dataset. To capture the non-linearities in the dataset, a non-linear dimensionality reduction technique such as kernel PCA can be used.

2.4.2 Kernel Principal Component Analysis

Kernel principal component analysis (Schlkopf, Smola, & Mller, 1998) is an extension of the principal component analysis that maps the data to a higher dimensional space by using kernel functions. Then PCA is applied to the data in the higher dimensional space by an appropriate positive semidefinite kernel function. The kernel function computes the inner product between the data points. The process of applying kernel PCA is described as follows:

Choose a kernel function. Some of the common kernel functions are linear function ($K(x_i, x_j) = x_j^T x_i$), polynomial function ($K(x_i, X_j) = (x_j^T x_i+1)^a$) and radial basis field or gaussian ($K(x_i, x_j) = e^{\frac{-\|x_i-x_j\|^2}{2\sigma^2}}$).

For a $M \times N$ data matrix x, kernel matrix matrix is computed as

$$K_{ij} = \kappa(x_i, x_j) \tag{9}$$

Compute the kernel matrix. The kernel matrix (K) is centered to ensure that the has zero mean. This can done by subtracting the mean of the computed kernel matrix. Here, 1_N is a $N \times N$ matrix with all elements equal to $\frac{1}{N}$ and N is the number of data points.

$$K_c = K - 1_N K K 1_N + 1_N K 1_N \tag{10}$$

Eigen value decomposition (EVD). In order to calculate the eigenvalues and eigenvectors of the centered kernel matrix, eigenvalue decomposition is used. Here, V is the eigenvectors and D is a diagonal matrix containing the eigenvalues in descending order.

$$VDV^T = EVD(K_c) \tag{11}$$

Data Projection. To perform the non-linear dimensionality reduction on the dataset, the top M' eigenvectors are used to transform to a higher dimensionality space where $\phi(x')$ is the final projected dataset of shape $M \times M'$ and $\mathbf{V}'$ are the selected eigenvectors for transformation.

$$\phi\left(\mathbf{x}_i\right) = \left[K\left(\mathbf{x}_i, \mathbf{x}_1\right), K\left(\mathbf{x}_i, \mathbf{x}_2\right), \ldots, K\left(\mathbf{x}_i, \mathbf{x}_M'\right)\right] \cdot \mathbf{V}' \tag{12}$$

Reconstruction. In order to map back the data to the original space, pre-image for each data point needs to computed (Bakir, Weston, & Scholkopf, 2003). This can be achieved by using the kernel ridge regression (KRR) which constructs a mapping function. This function facilitates the transformation of data from a lower-dimensional space into a higher-dimensional space. Within the scope of kernel PCA, KRR finds application to compute the pre-image for each data point within the original feature space. These pre-images are intended to closely approximate the corresponding original data points. So, for every data point, the it learns a mapping function that seeks to establish its pre-image within the high-dimensional feature space which can be used to reconstruct back to the original space.

Dimensionality reduction was performed using kernel PCA on of the examples in the excavation dataset. Through a systematic exploration of various kernel functions, it was found that the radial basis field function with a specific parameter setting ($\gamma = 0.04$) exhibited the closest resemblance to the ground truth. However, it is important to note that even with this selected kernel function, the model's performance still falls considerably short of accurately representing the actual dataset.

Kernel PCA demonstrates significant efficacy in capturing the nonlinear features within a dataset, making it a valuable tool for various tasks. However, one challenge associated with Kernel PCA is in the task of reconstructing the original dataset. The transformation into a higher-dimensional space achieved by kernelization directs the data to be linearly separable in that dimension and reveals the nonlinear patterns, but the inverse mapping from the transformed space to the original feature space is not straightforward, which poses difficulties for reconstructing the original dataset which is essential for visualizing the inference from the GNS. Thus, Kernel PCA is not considered further, and linear PCA is used in the remainder of this research for its ability to reconstruct the data back to higher dimensional space without any significant loss in information.

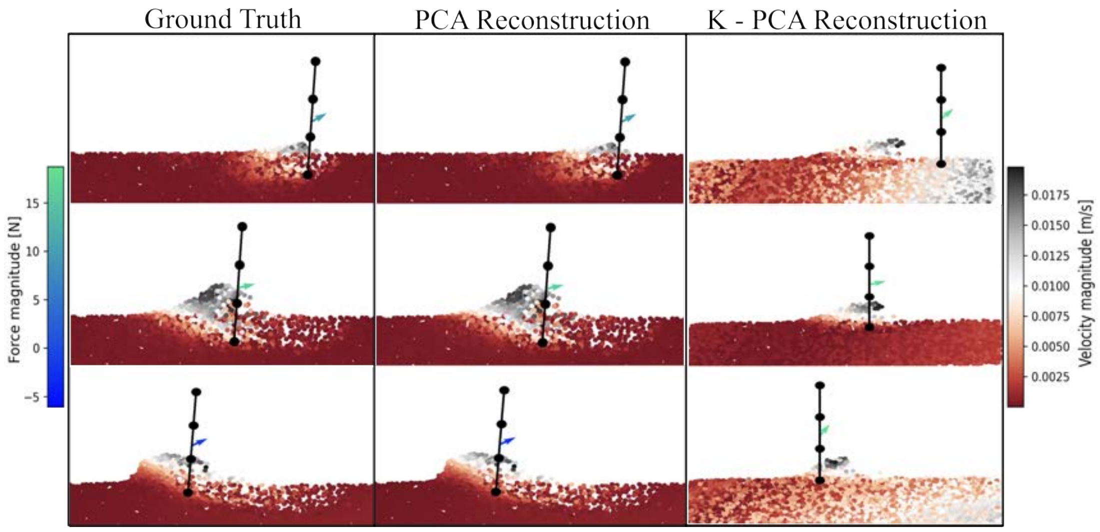


Figure 2.5: Ground truth, PCA reconstruction and kernel PCA reconstruction of the excavation example

2.5 Graph Network Simulator: Training and Inference

The GNS framework was extended for the case of rigid body driven granular flows (Haeri & Skonieczny, 2021). The dataset for training the GNS was generated using high-fidelity Non-local Granular Fluidity Material Point Method (NGF-MPM) simulations (Haeri & Skonieczny, 2022). The GNS model trained on the excavation dataset primarily focuses on predicting two key aspects:

the position of granular particles and the force applied to the rigid body. In the case of wheel, the GNS model faces a more intricate challenge. It not only predicts the positions of granular particles but also predicts the positions of particles representing the rigid body.

The model is trained using ADAM optimizer (Kingma & Ba, 2017) and uses a single message passing step for training and generating rollout. The GNS employs multi layer perceptron (MLP) for update functions. Each MLP has a dimensionality of 128 with two hidden layers and uses ReLU activations. The decoder also uses MLP but has an output dimensionality of 6 as it predicts the interaction between granular flow and the rigid body (Haeri & Skonieczny, 2021). Random walk noise with zero mean and standard deviation equal to 6.7×10^{-4} (hardcoded in the GNS codebase (Sanchez-Gonzalez et al., 2020)) is added to the input training data to ensure that the data distribution in the input is like the one in the rollout. This is to avoid error accumulation during rollout.

Once the model is trained, the model is fed initial positions of the soil particles along with the forces acting on the rigid body where the position of the soil particles gets reduced to PCA modes. Then, the reduced PCA modes along with the rigid body position and forces is fed as input to the trained GNS which predicts the next time step. The prediction is fed back again in autoregressive manner to generate rollouts. This enables the trained GNS to continuously generate the prediction by just providing the initial position and forces. Due the dimensionality reduction, the GNS learned the dynamics between the interaction of the rigid body and the PCA modes. This reduced the the nodes in the graph from 6016 particles to 24 particles in the case of wheel and 5000 particles to 28 particles in the case of excavation. This significant reduction in graph led to accurate real-time performance on single standalone GPU during inference. The following changes were adapted by Haeri and Skonieczny (2021) to the GNS for accurate and real time modelling of granular flow:

1. The dataset fed into the network includes the position of the rigid body (R_i), the interacting force on the rigid body (F_i), and, nominally, the positions of the granular particles (S_i).

2. To make the simulation real-time, the dimensionality of the granular particle position data was reduced using Principal Component Analysis (PCA) to 8 modes (S_i).

3. The GNS was adapted to predict not only particle positions but also the forces acting on the

rigid body.

4. The loss function was adapted to include the mean squared error of the predicted and ground truth interaction force on the rigid body.

$$L\left(Z^t, Z^{t+1}; \theta\right) = \left(\sum_{i=1}^{N^v} \left\|\hat{a}_i^{t+1} - a_i^{t+1}\right\|^2\right) / N^v + \left\|\hat{F}^{t+1} - F^{t+1}\right\|^2 \tag{13}$$

Here $\hat{F}^{t+1}$ is the ground truth force and $\hat{a}_i^{t+1}$ is the ground truth acceleration.

5. Instead of using the nearest neighbour algorithm for graph construction in the encoder, a fully connected undirected graph was used for message passing and learning.

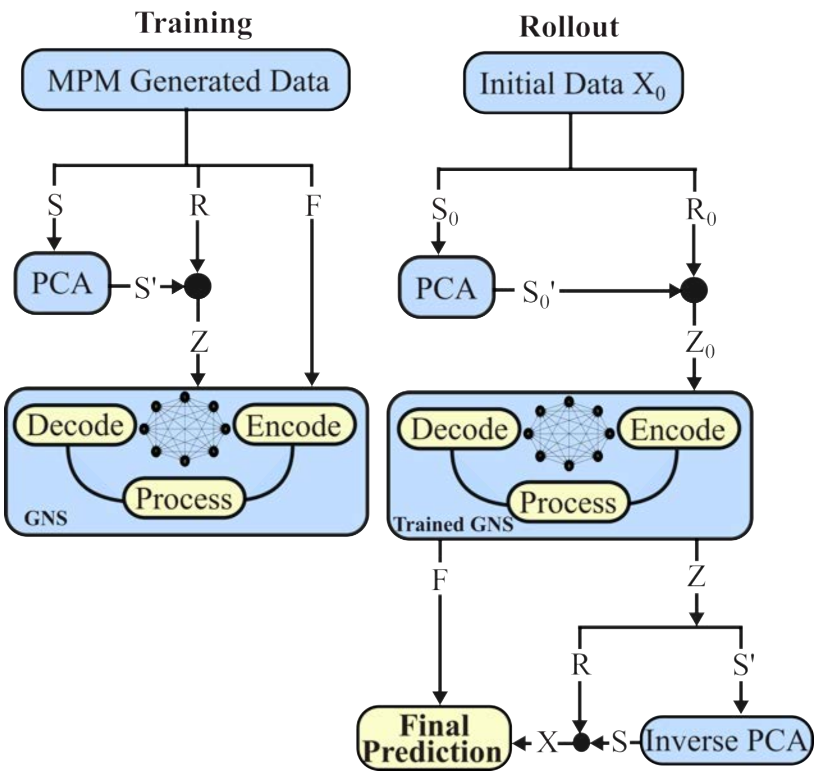


Figure 2.6: GNS formulation for training and rollout for accelerated rigid body driven granular flow where S refers to the full space granular particle position, S refers to the PCA reduced granular particle positions, R refers to the particles representing rigid body, F refers to the force acting on the rigid body.

Through their formulation, the authors successfully achieved accurate real-time modelling of excavation and wheel-soil interactions. Nevertheless, the graph construction approach employed in the encoder module was a simplistic fully connected scheme. This approach incurs a time complexity of $O(N^e)$ when updating the edge features, where N represents the number of nodes in the graph. Consequently, employing a smaller graph would decrease the time complexity and enhance inference speed. This raises the question of identifying the optimal partial graph configuration that minimizes the number of edges, consequently reducing the rollout time which is explored in chapter 3.

Chapter 3

Accelerating Real-Time GNN based Simulation

In the context of graph neural networks, the adjacency matrix introduced in figure 2.1b, is a way to directly represent the graph. The sum of the elements of the adjacency matrix (equal to the number of connections) is directly proportional to the inference time. Higher the number of connections in the graph, the longer it takes during the message passing and the update step. This research aims at identifying a sparser adjacency matrix of the graph that retains key connections. By reducing the number of edges, the graph would facilitate faster inference time. Neural Relational Inference (NRI), an unsupervised variational autoencoder model proposed by Kipf et al. (2018) takes in the particle trajectories as input and gives a partial graph indicating the most important edges. Using the encoder-decoder framework, NRI learns a latent representation of the input graph and identifies the most important edges that take part in the interaction while neglecting the redundant ones. The main contribution of the thesis is accelerating the modelling of rigid body-driven granular flows by utilizing a partial graph instead of a fully connected one. The Neural Relational Inference model is employed to identify the most optimal partial graph for the PCA modes that will reduce the inference time without significantly sacrificing accuracy.

Within this chapter, section 3.1 starts with a formal introduction to the variational autoencoder and provides mathematical insights into reparameterization trick used in its formulation. Further, section 3.2 furnishes an explanation of the neural relational inference framework, highlighting its architecture and the underlying loss function while section 3.2.1 describes the training procedure of NRI for the application of rigid body driven granular flows. Subsequently, section 3.3 delves into the discussion on accelerating graph networks, partial graph generation through the utilization of the neural relational inference framework and formalizes the partial graph framework for GNS in the case of rigid body driven granular flow simulations. Section 3.4 discusses a rigid body correction technique that prevents any physical distortion in the rigid body during simulation followed by section 3.5 which talks about the impact of training noise in the GNS. Finally the chapter ends with section 3.6 which provides detailed description of training and rollout using GNS on partially connected graph and compares the simulation results.

3.1 Variational Autoencoder

An autoencoder is a type of neural network designed to learn to recreate the input fed to it (Rumelhart, Hinton, & Williams, 1986). The autoencoder generally consists of two parts, an encoder and a decoder. The encoder is designed to compress the input to a lower dimensional space which is achieved by a bottleneck in the architecture. The decoder tries to reconstruct the input back again from the lower dimensional space or latent space learned by the encoder. This process of compression and reconstruction leads to the network learning and capturing the most important features of the datasets. A variational autoencoder is designed to learn a probability distribution of the latent space instead of a specific latent space. This enables the autoencoder to gain generative abilities, as the latent space is probabilistic, new unique samples can be generated by sampling from the learned latent space. Let latent space be represented by z and input data as x. The process of observing the latent space can formulated as computing the posterior $p(z \mid x)$.

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)} \tag{14}$$

However, computing the evidence $p(x)$ usually turns out to be intractable.

$$p(x) = \int p(x \mid z)p(z)dz \tag{15}$$

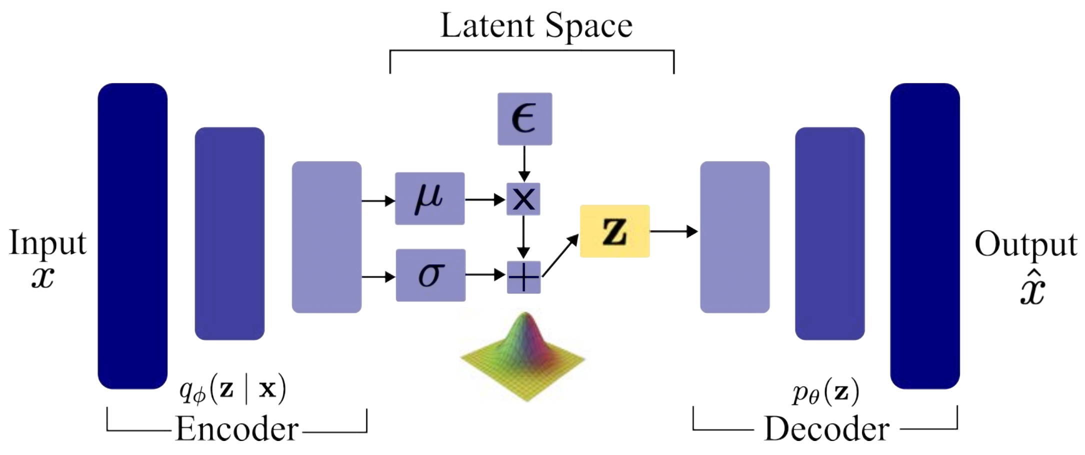


Figure 3.1: Variational Autoencoder

Therefore, in order to compute the posterior, the formulation of variational autoencoder utilizes variational inference to find an approximate distribution $q(z \mid x)$ which is similar to the true posterior $p(z \mid x)$. The final objective function is represented in equation 16, where the first term represents the reconstruction loss and the second term is the regularization term which is the Kullback-Leibler divergence between the approximated posterior and the prior to ensure that the learned distribution resembles the chosen prior which is usually a standard normal distribution.

$$\mathcal{L}_{\phi,\theta}(\mathbf{x}) = E_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathrm{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z})\right] \tag{16}$$

However, the backpropagation cannot take place due to due to presence of a stochastic node, which restricts gradient computation. Formal justification can be described as follows (Gundersen, 2018). For an expectation $E_{p(z)}\left[f_\theta(z)\right]$, where $f_\theta(z)$ is differentiable and parameterized over θ while $p(z)$ is the probability density function over which the expectation is calculated. The gradient computation with respect to theta is tractable.

$$\begin{aligned}
\nabla_\theta E_{p(z)}\left[f_\theta(z)\right] &= \nabla_\theta\left[\int_z p(z) f_\theta(z) dz\right] \\
&= \int_z p(z)\left[\nabla_\theta f_\theta(z)\right] dz \\
&= E_{p(z)}\left[\nabla_\theta f_\theta(z)\right]
\end{aligned}$$

The result shows that the gradient of the expectation (E) with respect to theta θ is the expectation of the gradient with respect to theta, however this does not hold true when the probability density function over which the expectation is calculated is also parameterized by θ.

$$\begin{aligned}
\nabla_\theta E_{p_\theta(z)}\left[f_\theta(z)\right] &= \nabla_\theta\left[\int_z p_\theta(z) f_\theta(z) dz\right] \\
&= \int_z \nabla_\theta\left[p_\theta(z) f_\theta(z)\right] dz \\
&= \int_z f_\theta(z) \nabla_\theta p_\theta(z) dz + \int_z p_\theta(z) \nabla_\theta f_\theta(z) dz \\
&= \int_z f_\theta(z) \nabla_\theta p_\theta(z) dz + E_{p_\theta(z)}\left[\nabla_\theta f_\theta(z)\right]
\end{aligned}$$

Upon simplification, the first term may not be an expectation. The challenge arises when attempting to compute the gradient of the density funciton that is parameterized and subject to optimization. In essence, the dynamic nature of the distribution and its changing nature to variations in the parameters leads to complexity for the computation of the gradient. This problem is addressed by a reparameterization of the lower bound introduced by Kingma and Welling (2013). This implies that we take a random sample from a normal distribution $N(0,1)$, and then transform it using the learned mean μ and variance σ of the encoder for the input $\mathbf{x}$ as shown in equation 17. This gets rid of the stochasticity and enables backpropagation in the network.

$$\begin{aligned}
\mathbf{z} &= \mu + \sigma \cdot \epsilon \\
\mathbf{z} &= g_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}, \mathbf{x})
\end{aligned} \tag{17}$$

The reparameterized $\mathbf{z}$ can be used instead which makes it deterministic in nature and allows for backpropagation for training the network. Using reparameterization, the gradient of the expectation over a paramterized probability function is equal to the expectation of the gradient as shown in equation 18.

$$\begin{aligned}
E_{p_\theta(\mathbf{z})}\left[f\left(\mathbf{z}\right)\right] &= E_{p(\boldsymbol{\epsilon})}\left[f\left(g_\theta\left(\boldsymbol{\epsilon}, \mathbf{x}\right)\right)\right] \\
\nabla_\theta E_{p_\theta(\mathbf{z})}\left[f\left(\mathbf{z}\right)\right] &= \nabla_\theta E_{p(\boldsymbol{\epsilon})}\left[f\left(g_\theta\left(\boldsymbol{\epsilon}, \mathbf{x}\right)\right)\right] \\
&= E_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\theta}} f\left(g_\theta\left(\boldsymbol{\epsilon}, \mathbf{x}\right)\right)\right]
\end{aligned} \tag{18}$$

3.2 Neural Relational Inference

The Neural Relational Inference (NRI) model is an unsupervised learning model based on the variational autoencoder architecture where the latent space encodes the pairwise interactions between the edges of the senders and receivers (Kipf et al., 2018). It is able to learn and encode the dynamics from the trajectory dataset. The NRI uses graph neural networks to represent and learn the interaction between the nodes by message-passing. Here, the NRI trains an encoder that models the physical interactions between the input trajectories. The role of the decoder is to predict future trajectories as well as reconstruct the ground truth trajectories fed into the model.

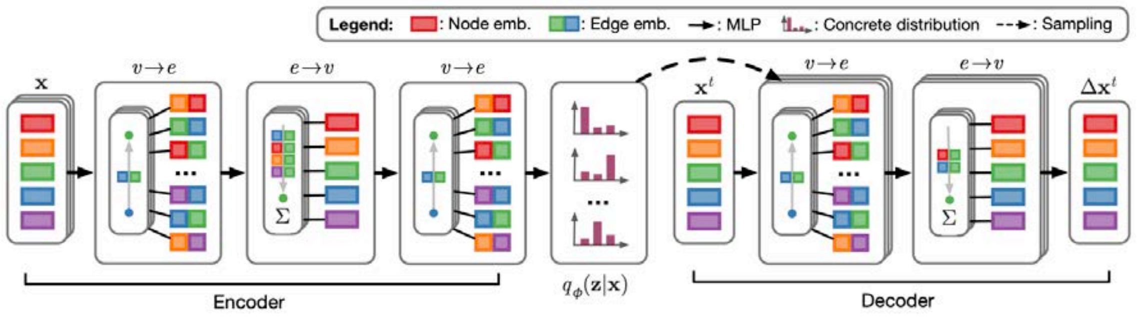


Figure 3.2: The Neural Relational Inference (NRI) architecture (Kipf et al., 2018)

The NRI tries to optimize the evidence lower bound:

$$\mathcal{L} = E_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathrm{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z})\right] \tag{19}$$

where $q_\phi(\mathbf{z} \mid \mathbf{x})$ is the discrete distribution learned encoder, $\log p_\theta(\mathbf{x} \mid \mathbf{z})$ is the predicted trajectory by the decoder, $p_\theta(\mathbf{z})$ is the prior and KL refers to the KL divergence. The encoder tries to predict the pairwise relationship between the nodes in the graph with a fully connected graph as input. The following message-passing operations take place.

$$\mathbf{h}_j^1 = f_{\mathrm{emb}}\left(\mathbf{x}_j\right) \tag{20}$$

$$v \to e: \quad \mathbf{h}_{(i,j)}^1 = f_e^1\left(\left[\mathbf{h}_i^1, \mathbf{h}_j^1\right]\right) \tag{21}$$

$$e \to v: \quad \mathbf{h}_j^2 = f_v^1 \left(\sum_{i \neq j} \mathbf{h}_{(i,j)}^1 \right) \tag{22}$$

$$v \to e: \quad \mathbf{h}_{(i,j)}^2 = f_e^2 \left(\left[\mathbf{h}_i^3, \mathbf{h}_j^3 \right] \right) \tag{23}$$

$$q_\phi(\mathbf{z} \mid \mathbf{x}) = \operatorname{softmax} \left(\mathbf{h}_{(i,j)}^2 \right) \tag{24}$$

Equations 20, 21, 22 and 23 mathematically describe the messaging passing operations where $\mathbf{h}_i^1$ and $\mathbf{h}_j^1$ refer to the node embedding of the node v_i and v_j in layer 1 respectively. $\mathbf{h}_{(i,j)}^2$ and $\mathbf{h}_{(i,j)}^2$ refers to the node embedding of the node v_i and v_j,in layer in 2 respectively. $\mathbf{h}_{(i,j)}^1$ and $\mathbf{h}_{(i,j)}^2$ refers to the edge embedding of node v_i and v_j in the first layer and second layer respectively. $q_\phi(\mathbf{z} \mid \mathbf{x})$ in equation 24 represents the predicted probability of the latent space given the trajectory of the particles and f refers to the neural network. However, sampling from the encoder is not possible as the distribution is discrete. Instead, the NRI model uses the Gumbel-Softmax trick to sample from this distribution where g refers to a sample from the Gumbel distribution and τ softmax temperature which decides how smooth the samples will be. The decoder aims to predict the future trajectory given the latent input graph z where each edge type is represented as a neural network.

$$z_{ij} = \operatorname{softmax} \left(\left(\mathbf{h}_{(i,j)}^2 + g \right) / \tau \right) \tag{25}$$

The decoder can be mathematically described as follows:

$$\mathrm{v} \to \mathrm{e}: \quad \tilde{\mathrm{h}}_{(\mathrm{i,j})}^{\mathrm{t}} = \sum_{\mathrm{k}} \mathrm{z}_{\mathrm{ij,k}} \tilde{\mathrm{f}}_{\mathrm{e}}^{\mathrm{k}} \left(\left[\mathrm{x}_{\mathrm{i}}^{\mathrm{t}}, \mathrm{x}_{\mathrm{j}}^{\mathrm{t}} \right] \right) \tag{26}$$

$$e \to v: \quad \mu_j^{t+1} = x_j^t + \tilde{f}_v \left(\sum_{i \neq j} \tilde{h}_{(i,j)}^t \right) \tag{27}$$

$$p \left(x_j^{t+1} \mid x^t, z \right) = \mathcal{N} \left(\mu_j^{t+1}, \sigma^2 I \right) \tag{28}$$

In Equations 26 and 27, Kipf et al. (2018) provides mathematical descriptions of the message passing operations within the proposed model. In particular, $\tilde{h}_{(i,j)}^t$ represents the edge embedding, while μ_j^{t+1}, denotes the mean of the predicted trajectory. The fundamental concept underlying the

NRI model is to acquire a comprehensive understanding of the physical dynamics exhibited by trajectories based on a dataset of previous trajectories. This understanding enables the model to accurately predict future timesteps given an initial condition. Since the NRI model is built upon the framework of a variational autoencoder, the bottleneck encodes pairwise interactions, thereby providing valuable insights into relative influence within the dataset.

3.2.1 Training and Inference

The Neural Relational Inference (NRI) model processes a sequence of states, representing the trajectories of Principal Component Analysis (PCA) modes (reduced-order representation of granular particle positions) over time. Figure 3.3 shows an interaction between the rigid body and PCA modes, however, the NRI does not produce consistent results when fed with trajectory of both the PCA modes and rigid body. Therefore, in this case, only the trajectory of the PCA modes are fed into NRI to reveal the interaction graph between them. The input comprises of the positional and velocity information of the PCA modes, along with an initial guess of the adjacency matrix (fully connected or no connection). The encoder utilizes this input to calculate the probability distribution of the latent space ($q_\phi(\mathbf{z} \mid \mathbf{x})$).

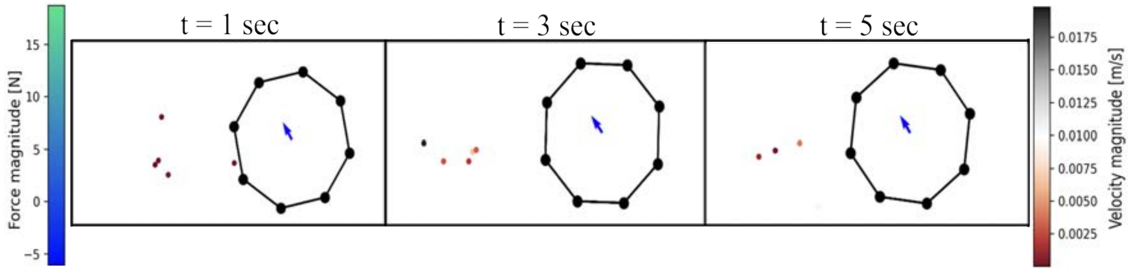


Figure 3.3: Interaction between PCA modes and rigid body at time 1 second, 3 second and 5 second.

For each edge, discrete values between 0 and 1 are generated through a softmax function applied to the embedding of the nodes. A threshold of 0.6 is employed to categorize the presence or absence of an edge: values less than 0.6 signify no edge, while values exceeding 0.6 denote the existence of an edge. Subsequently, the decoder forecasts the future trajectory of the PCA modes, and the Neural Relational Inference (NRI) model computes the final loss, facilitating the backpropagation of gradients. The reconstruction loss, detailed in equation 19, is approximated in accordance with the methodology introduced by Kipf et al. (2018), as outlined in equation 29. The Kullback-Leibler (KL) divergence term, under a uniform prior, is approximated as the sum of entropies, as expressed in equation 30.

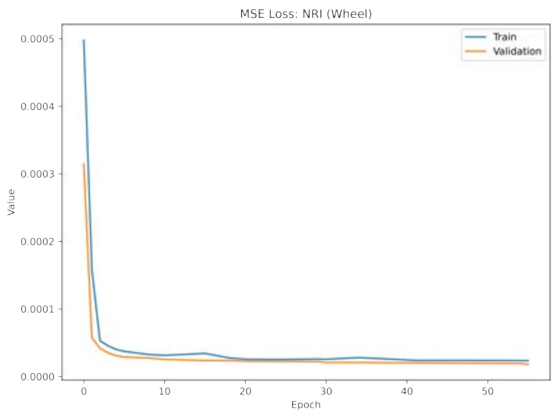
$$-\sum_{j}\sum_{t=2}^{T}\frac{\left\|\mathbf{x}_j^t-\boldsymbol{\mu}_j^t\right\|^2}{2\sigma^2}+\text{ const} \tag{29}$$

$$\sum_{i\neq j} H\left(q_\phi\left(\mathbf{z}_{ij}\mid\mathbf{x}\right)\right)+\text{ const.} \tag{30}$$

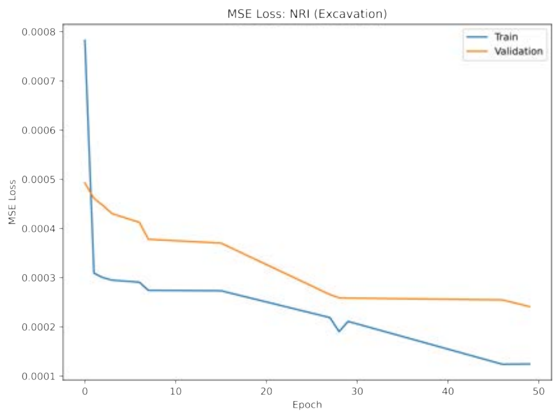
As the distribution is not continuous, the reparameterization trick for the variational autoencoder cannot be used. To overcome this problem, Kipf et al. (2018) used the Gumbel-Softmax trick to sample from a categorical distribution in order to backpropagate the gradients in the network. The final modified gumbel-softmax used for training the NRI model is shown in equation 31.

$$y_i=\frac{e^{\frac{(\log(\pi_i)+g_i)}{\tau}}}{\sum_{n=1}^{m} e^{\frac{(\log(\pi_n)+g_n)}{\tau}}}\text{ for } i=1,\ldots,m \tag{31}$$

Here, g is sampled from gumbel distribution $Gumbel(0,1)$, π refers to the class probabilities and τ is the softmax temperature where lower values of τ lead to expected value of the categorical variables while the higher values lead to uniform distribution over the categories.



(a) MSE loss between the input trajectory and output from the decoder for wheel



(b) MSE loss between the input trajectory and output from the decoder for excavation

Figure 3.4: MSE loss in NRI for training and validation steps

The mean squared error loss between the input trajectory and output from the decoder for both wheel and excavation is shown in figure 3.4. The NRI was trained using adaptive momentum estimation (Kingma & Ba, 2017), with 5×10^{-4} as the learning rate and 0.5 as the decay factor. The number of epochs were set to 200, however the model converges before and stores the best values.

3.3 Accelerating Subspace Graph Network Simulator for rigid body driven granular flows using NRI generated partial graph

In the framework of the Graph Network Simulator (GNS) for rigid body driven granular flows, our groups prior work used a fully connected graph of PCA modes along with the rigid body nodes. To make the GNS computationally efficient while maintaining its accuracy, the NRI model is used in this work to identify a partial connection between the extracted PCA modes. NRI was applied to both the excavation dataset as well as the wheel dataset. In the excavation example, only PCA modes (reduced-order representation of granular particle positions) and the force on the center of mass of the excavation blade were predicted, as the rigid body (flat plate blade) followed a preset trajectory. In the wheel example, both the PCA modes (representing granular particle positions) and the wheel position were predicted along with the force on the center of mass of the wheel.

The NRI model was fed the positions and the velocities (differences between subsequent pairs of position data) of the excavation dataset and the wheel dataset. The encoder learns an interaction network representing a partial graph for the PCA modes of the datasets as shown in figure 3.5. The senders refer to the nodes that send messages using the message passing process to the receiver nodes. A dark square indicates an edge with important message passing, while lighter squares indicate little to no information being passed. A partial graph can thus be constructed with just the edges corresponding to dark squares. The partial graph framework is split between four parts where the graph has edges between senders and receivers. The senders and receivers define the interaction between PCA modes to PCA modes, rigid body to rigid body, rigid body to PCA modes, and PCA modes to rigid body. The partial graph formulations for each of these interactions are explained below, and are summarized by figure 3.6 and 3.7 for wheel and excavation datasets, respectively.

The NRI model was fed the positions and the velocities (differences between subsequent pairs

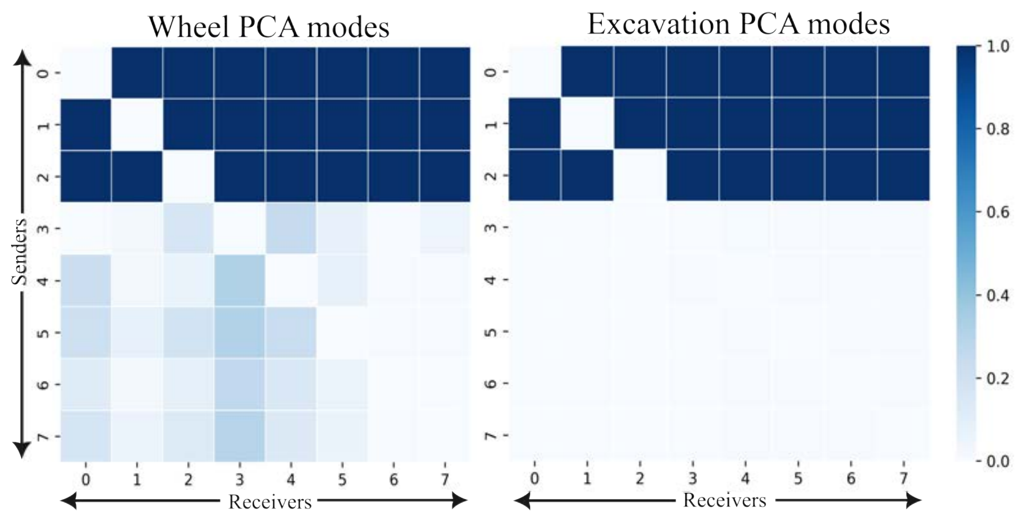


Figure 3.5: Output from NRI: partial graph representing the interaction between PCA modes for wheel and excavation datasets.

of position data) of the excavation dataset and the wheel dataset. The encoder learns an interaction network representing a partial graph for the PCA modes of the datasets as shown in figure 3.5. The senders refer to the nodes that send messages using the message passing process to the receiver nodes. A dark square indicates an edge with important message passing, while lighter squares indicate little to no information being passed. A partial graph can thus be constructed with just the edges corresponding to dark squares. The partial graph framework is split between four parts where the graph has edges between senders and receivers. The senders and receivers define the interaction between PCA modes to PCA modes, rigid body to rigid body, rigid body to PCA modes, and PCA modes to rigid body. The partial graph formulations for each of these interactions are explained below, and are summarized by figure 3.6 and 3.7 for wheel and excavation datasets, respectively.

PCA modes to PCA modes: The interaction between PCA modes is predicted by the NRI model, as described above. It is worth noting that the NRI prediction solely captures the interaction between the PCA modes, disregarding the interaction of stationary granular particles within the PCA mode. To address this limitation, an additional mode is introduced to represent the stationary granular particles (compare figure 3.5 with figure 3.6 and figure 3.7).

Rigid Body to Rigid Body: When considering the interaction between particles representing rigid bodies, message passing occurs only if the motion of the rigid body is also predicted by the

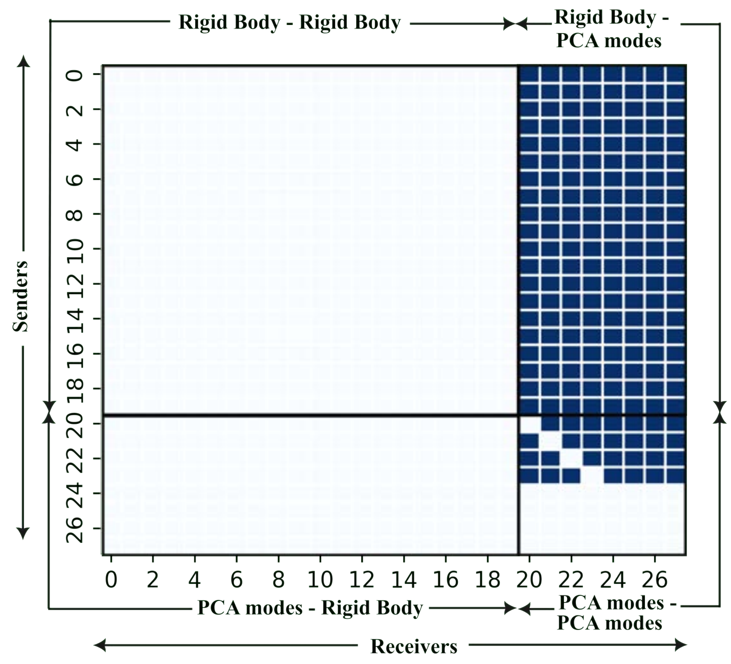


Figure 3.6: Partial graph for excavation driven granular flow.

network. In the excavation case, where the rigid body motion is not predicted, no message passing occurs between the rigid body particles, resulting in the absence of edges in the adjacency graph to represent these interactions.

However, in the case of the wheel, where the network predicts the motion of the rigid body, message passing becomes necessary between the particles representing the rigid body. To determine the appropriate rigid body edges, various trials were conducted. For the wheel scenario, it was determined that establishing edge connections from each rigid body particle to its consecutive rigid body particles yielded satisfactory results.

Rigid Body to PCA modes: The message-passing from the rigid-body to the PCA modes occurs in both the excavation example as well as the wheel. This part is crucial for rigid body driven granular flow, so all the particles representing the rigid body have an edge in the adjacency matrix for all the connections with all the PCA modes. This enables the GNS to learn the relative motion of the PCA modes with respect to the rigid body.

PCA modes to Rigid Body: The message-passing in this case occurs only when the positions of the particles representing the rigid body are also predicted. For this case, the GNS needs to

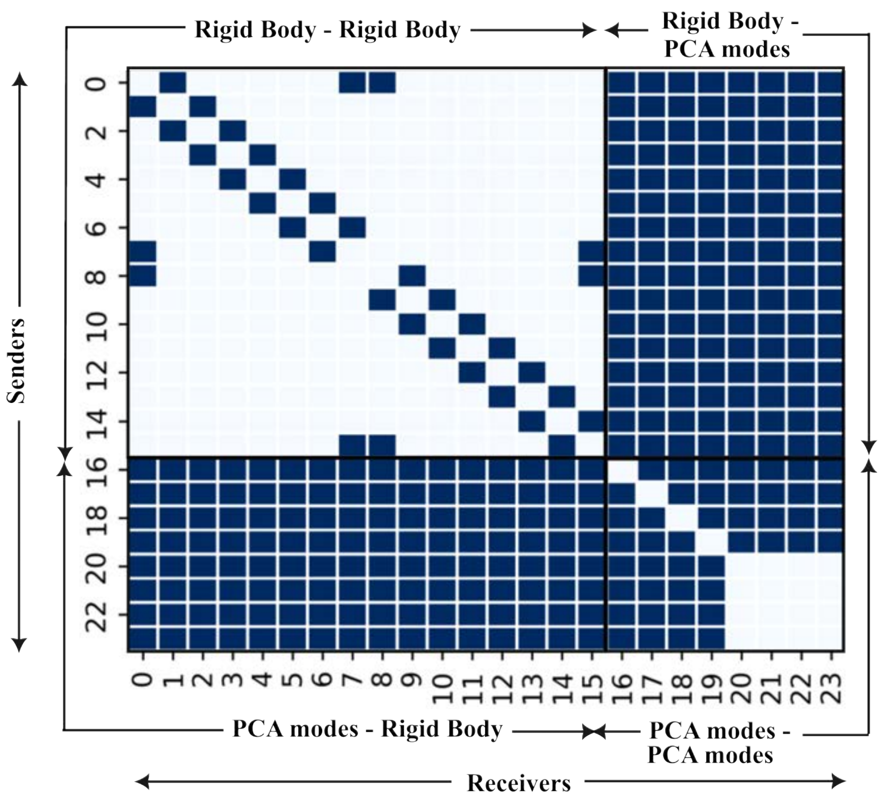


Figure 3.7: Partial graph for wheel driven granular flow.

Algorithm 2 Constructing Sparse Adjacency Matrix in Graph Network Simulator for Interaction between Rigid Body and PCA modes

```
1: procedure ConstructAdjacencyMatrix
2:     for all PCA modes to PCA modes do
3:         Predict interaction using NRI model
4:     end for
5:     for all Rigid Body to Rigid Body do
6:         if Motion of rigid body predicted by network then
7:             add edge based on geometry of the rigid body
8:         end if
9:     end for
10:    for all Rigid Body to PCA modes do
11:        add edges to all nodes
12:    end for
13:    for all PCA modes to Rigid Body do
14:        if Positions of particles representing rigid body predicted then
15:            add edges to all nodes
16:        end if
17:    end for
18: end procedure
```

learn the motion of the particles representing the rigid body relative to the PCA modes. Messaging passing is necessary between the granular particles and the wheel, leading to an undirected graph that enables the model to learn the dynamics of the wheel motion. Therefore, there is an edge in the adjacency matrix for all the connections between PCA modes to the rigid body in the case of the wheel but no edge in the case of excavation.

In the excavation example, the position of the particles representing the blade is pre-defined and is not predicted by the network. Consequently, there is no messaging passing required from the granular particles to the blade, resulting in a directed graph.

The optimized partial graph formulation resulted in a significant reduction in the number of edges within the adjacency graph. Specifically, for excavation driven granular flow, the number of edges decreased from 784 to 188, while for wheel driven granular flow, it decreased from 576 to 336. These reduced partial graphs successfully replicated the desired outcomes while offering faster rollout times, primarily attributed to the decreased number of edges present in the adjacency matrix. However, in the case of predicting the node positions of the wheel, a discrepancy was observed where the particles within the rigid body failed to maintain rigid relative spacing. This issue is addressed in the subsequent sub-section through the utilization of least square fitting techniques.

3.4 Rigid Body Correction by Least Square Fitting of 3D points

In the case of the wheel, predicting the position of the particles representing the rigid body introduces a slight error in the wheel position. This error propagates to incorrect slip predictions and eventually inaccurate predictions for the granular particles. To mitigate the accumulation of errors during the rollout, it is essential to correct the position of the particles representing the rigid body, i.e. maintaining the wheels rigid shape. To address this issue, a formulation is presented to minimize the discrepancy between the particles representing the wheel before and after rotation. This involves finding an appropriate rotation matrix and translation vector that minimize the error. The rigid body correction is not introduced during the training of the GNS and is only used during the rollout to keep the rigid body intact. The problem can be formally described as a minimization problem, with the objective of minimizing the difference between the post rotation prediction of the particles representing the wheel and rigid transformation of pre-rotation.

$$\min_{R\in\Omega,d}\sum_{i=1}^{n}\|Rx_i+d-y_i\|^2$$

Here n is the number of rigid body particles, *R* is the rotation matrix and *d* is the translation vector. x_i and y_i are the particle positions before wheel rotation and after rotation respectively. This is adapted from Sorkine-Hornung and Rabinovich (2017) and Söderkvist and Wedin (1994) who formulated the process as shown in algorithm 1 below.

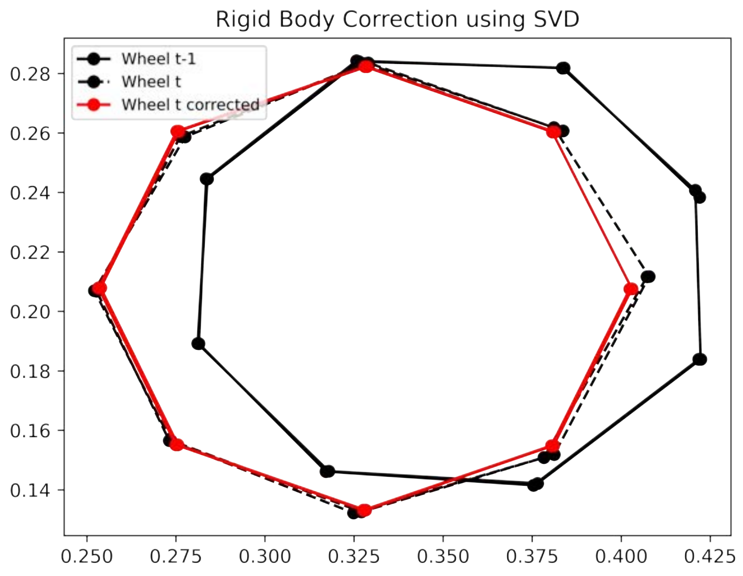


Figure 3.8: Rigid body correction at $t = 220$ for wheel driven granular flow.

Algorithm 3 Rigid Body Correction by Least Square fitting [Sorkine-Hornung and Rabinovich (2017)]

```
Input: $x_1, x_2, .., x_n, y_1, y_2, .., y_n$
Output: $R$, $t$
1: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$
2: $A = [x_1 - \bar{x}, \ldots, x_n - \bar{x}]$, $B = [y_1 - \bar{y}, \ldots, y_n - \bar{y}]$
3: $C = BA^T$
4: $USV^T = C$
5: $R = U \times diag(1, 1, det(UV^T) \times V^T$
6: $d = \bar{y} - R\bar{x}$
```

This algorithm does not explicitly correct the particle positions of the rigid body but rather extracts a rotation matrix and a translation vector to rotate the rigid body particles from before the rotation. This algorithm is added along with the rollout to minimize any error in the rigid body prediction. For the purpose of demonstration, figure 3.8 shows the correction of the deformed wheel for 220^{th} frame in the rollout.

3.5 Training Noise

The training noise added to the data was generated using random-walk, initially sampled from a normal distribution with zero mean and standard deviation 6.7×10^{-4} (hardcoded in the GNS codebase (Sanchez-Gonzalez et al., 2020)). This addition of random noise in the training data is due to several reasons. Random noise leads to data augmentation in every step which promotes generalization and prevents over-fitting. The main purpose of adding noise is to mimic the error accumulation when we perform rollout to generate predictions for the dataset. The introduction of random noise serves the purpose of training the model with an awareness of error accumulation tendencies, thereby preventing the possibility for deviations in predictions. Nevertheless, the selection of the appropriate standard deviation for this random noise is essential, as higher magnitude of noise has the potential to introduce distortion into the original dataset. In the context of our study, the standard deviation of the Principal Component Analysis (PCA) modes was found to be on the order of 10^{-5}, which is notably lower than that of the standard deviation of the random noise. This observation is visually depicted in figure 3.9, wherein the introduction of random noise with a standard deviation exceeding that of the dataset's standard deviation leads to a significant distortion of the dataset. This addition of noise is counterproductive, as it directs the model towards training on an adulterated dataset, rather than enhancing the robustness of the model. This led to the training data that was randomized to the point of data corruption, rather than helping in generalization. To tackle this, the standard deviation of the normal distribution was modified from 6.7×10^{-4} which was used by Sanchez-Gonzalez et al. (2020) in the GNS codebase to 6.7×10^{-6}. This random noise addition does not alter the dataset but adds slight perturbations mimicking the scenario of rollout error accumulation. This reduced any significant alterations in the training dataset as seen in figure 3.9.

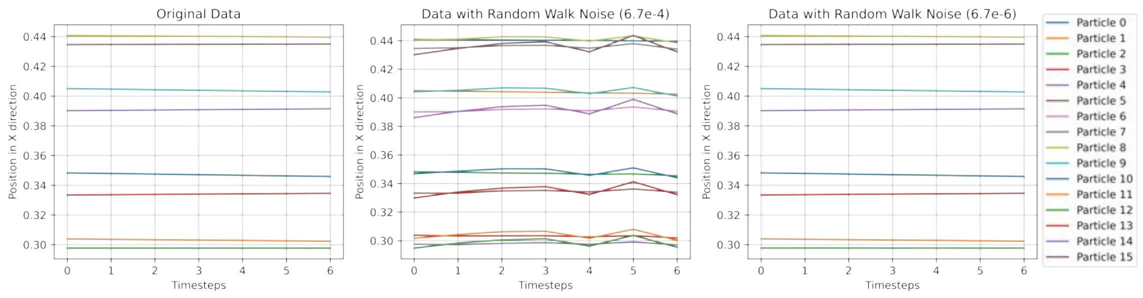


Figure 3.9: Impact of random noise on the particles representing rigid body (wheel)

3.6 Rollout

To generate the rollout, the graph network simulator is given the initial positions along with the initial forces, and the GNS then iteratively predicts the particle positions and force acting on the rigid body. The graph input into the Graph Neural Simulator (GNS) is not a completely interconnected graph; instead, it is a partial graph. As the GNS uses a partially connected graph instead of a fully connected graph, redundant nodes are eliminated. However, this leads to nodes where there is no message passing which leads lower prediction accuracy. In the context of granular flow driven by excavation, the model is supplied with the initial position and forces during the rollout process, which is utilized to generate position and force values for subsequent time steps in an autoregressive fashion. In the case of wheel driven granular flow, the predicted position of the rigid body undergoes a correction process to ensure the preservation of the rigid body's shape (as described in section 3.4).

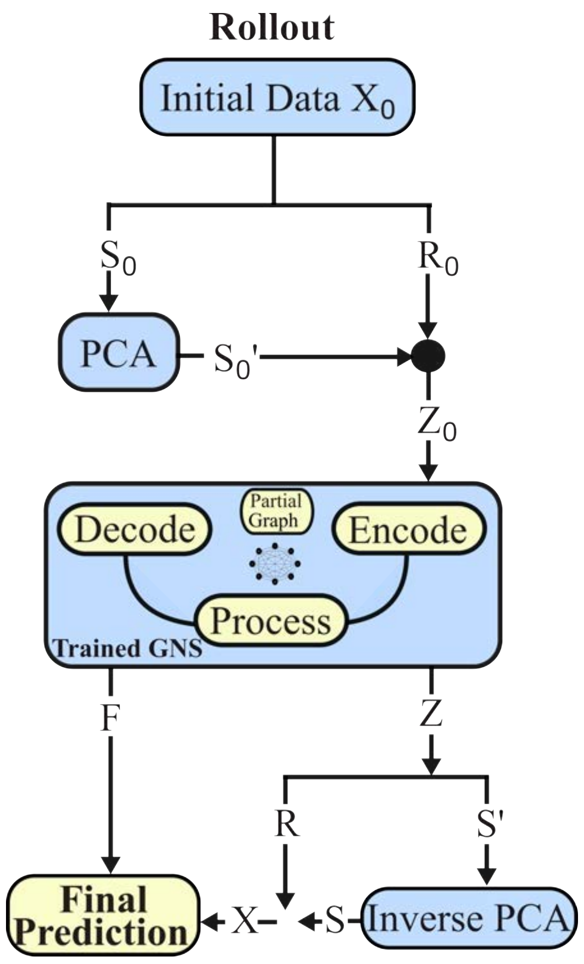


Figure 3.10: GNS formulation for rollout with NRI assisted partial graph for accelerated rigid body driven granular flow where S refers to the full space granular particle position, S refers to the PCA reduced granular particle positions, R refers to the particles representing rigid body, F refers to the force acting on the rigid body.

Runtime Efficiency: Table 3.1 shows the runtime efficiency of the MPM method, full space dataset, subspace dataset with a fully connected graph, and subspace dataset with a partially connected graph (this work). For the 60 Hz simulation, the partial graph has a faster training time as well as rollout time compared to the fully connected graph where it is 3x faster for excavation and 2x faster in the case of wheel. The GNS predicts future trajectory of the PCA modes and the rigid body. To obtain predictions in the full space, it is necessary to transform the predicted PCA modes back to the full space using the inverse PCA. However, this transformation introduces an additional computational overhead. In the context of excavation and wheel-driven granular flow, this overhead amounts to an additional 2.6 ms and 4.2 ms per second of simulation at 60 Hz, respectively.

Table 3.1: Run time comparison of rigid body driven simulation using Material Point Method (MPM), full space GNS (FS), subspace GNS with fully connected graph (FCG) (Haeri & Skonieczny, 2021) and subspace GNS with partially connected graph (PCG). The excavation blade is represented by 15^a and 20^b particles while the wheel is represented as 16 particles. GPU^1 : NVIDIA GeForce RTX 3080 10GB, GPU^2: NVIDIA Tesla P6 16GB, CPU^1: Intel Core i7-10700 2.9GHz 8-Core, and CPU^2: Intel Core i7-6700 3.4GHz Quad-Core.

Dataset	Method	Training [sec/step]		Rollout [sec/sec] (60 Hz)		Simulation [sec/sec] (60 Hz)
		GPU^1	GPU^2	CPU^1	GPU^1	CPU^2
Excavation	MPM	-	-	-	-	270
	FS	OOM	1	-	OOM	-
	FCG	0.016	0.064	$(0.344 - 0.348)^a$ $(0.486 - 0.488)^b$	$(0.447 - 0.491)^a$ $(0.558 - 0.583)^b$	-
	PCG (ours)	**0.01**	-	**$(0.127 - 0.128)^a$** **$(0.15 - 0.152)^b$**	**$(0.132 - 0.170)^a$** **$(0.244 - 0.342)^b$**	
Wheel	MPM	-	-	-	-	240
	FS	OOM	0.5	-	OOM	-
	FCG	0.017	0.068	0.374 - 0.378	0.434 - 0.499	-
	PCG (ours)	**0.011**	-	**0.182 - 0.186**	**0.180 - 0.184**	-

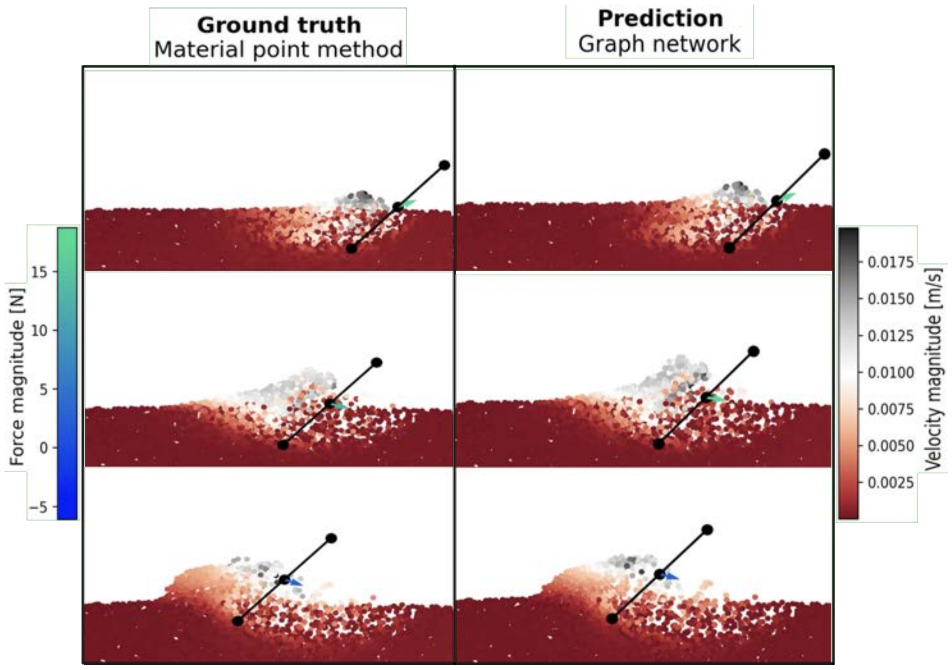


Figure 3.11: Ground Truth and Prediction of the excavation driven granular flow at time 1 second, 3 second and 5 second.

Prediction Accuracy: In order to compare the accuracy of the models, the visualization of the particles and forces of the simulation are compared with the ground truth and the prediction in the x-y plane for the excavation dataset at timesteps 1 second, 3 second, and 5 second mark as shown in figure 3.11. The histogram in figure 3.12 quantifies the error between the predicted particle positions from the GNS with the partially connected graph and the ground truth from MPM for the excavation dataset. In the prediction, most of the particles have a mean squared error between the order of 10^{-8} and 10^{-6} (units of mm^2) which points towards the stationary particles in the simulation. Due to the reduction of the edges, prediction accuracy of the force is slightly reduced, but, the error in predicting the particle positions has reduced, however there is no significant impact the on visual representation in the rollout as seen in figure 3.11 and figure 3.14.

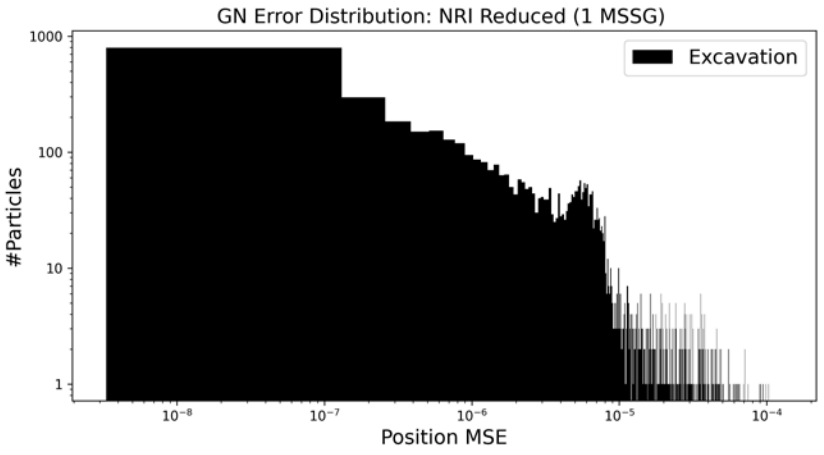


Figure 3.12: Histogram of error between particle positions of the ground truth and prediction of the excavation driven granular flow.

In the context of the wheel dataset, a comparison is made between the visualization of particles and forces obtained from the simulation, the ground truth, and the predictions in the x-y plane. When the simulation does not incorporate the rigid body correction, the particles representing the rigid body exhibit incorrect predictions. Consequently, this leads to an amplified error in the position of the granular particles. It is important to note that the positions of the granular particles are dependent on the positions of the particles representing the rigid body. Therefore, any inaccuracies in the prediction of the rigid body positions result in increased errors in the positions of the granular particles during rollout as depicted in figure 3.13.

When the rigid body motion is corrected, both the rigid body particles as well as the granular particles produce accurate prediction with respect to the ground truth. The visualization between the ground truth and the prediction for the case when rigid motion is not corrected as well as when the rigid body motion is corrected is shown in the figure 3.13 and 3.14 respectively. The histogram quantifies the error between the predicted particle positions from the GNS with the partially connected graph and the ground truth from MPM for the wheel dataset. In the prediction, most of the particles have a mean squared error between the order of 10^{-8} and 10^{-6} which reflects the accuracy of the simulation. However, there is higher error in the case when there is no rigid body correction applied in the rollout as depicted in figure 3.13.

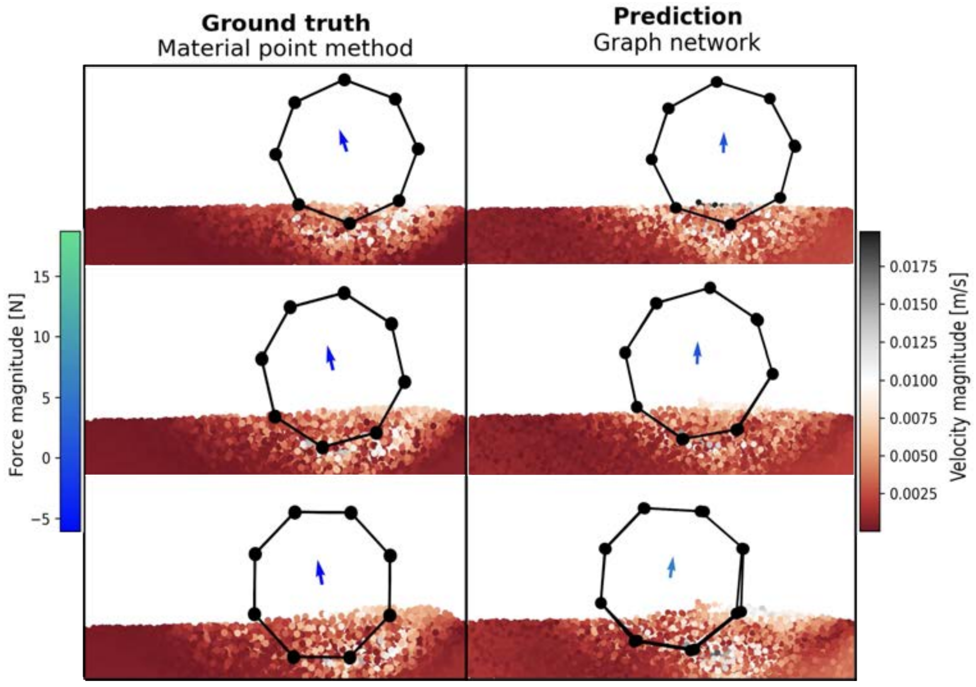


Figure 3.13: Ground Truth and Prediction of the wheel driven granular flow at time 1 second, 3 second and 5 second without rigid body correction.

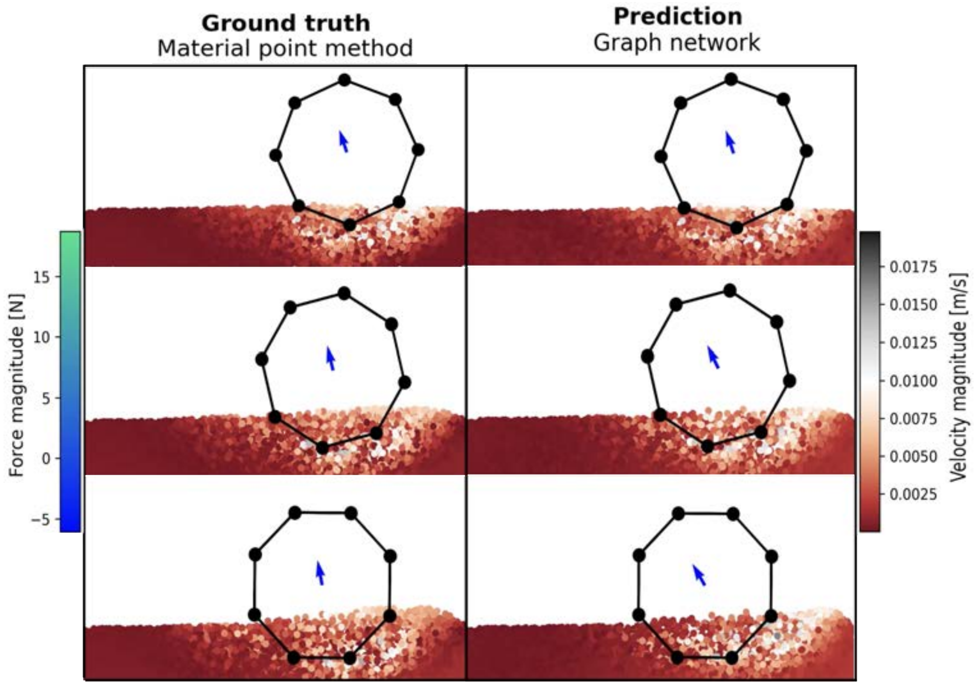
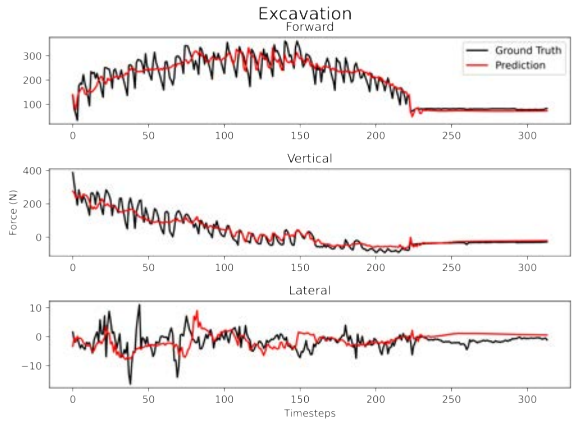
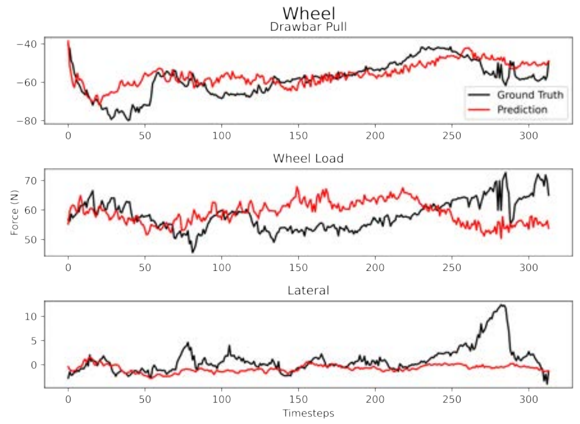


Figure 3.14: Ground Truth and Prediction of the wheel driven granular flow at time 1 second, 3 second and 5 second with rigid body correction.

The addition of the rigid body correction which keeps the shape of the rigid body intact during rollout not only avoids any deformations of rigid body, but also leads to more accurate prediction of the particle. Table 3.2 shows the mean squared error of the position of the soil particles and rigid body for the two cases: with rigid body correction and without rigid body correction.



(a) Excavation



(b) Wheel

Figure 3.15: Ground truth and prediction of the interaction forces

The force prediction for the wheel and excavation for the examples in figure 3.13 and figure 3.14 are shown in figure 3.15a and 3.15b respectively. In these visual representations, the black color signifies the ground truth forces, while the red color represents the predictions generated by the graph network simulator utilizing a partial graph. It can be seen that after applying rigid body correction, the prediction of soil particles as well as the rigid body drastically improves (force predictions without rigid body correction are presented in figure 3.16). This is because the GNS learns the interaction between the rigid body and PCA modes, thus, any distortion in the rigid body can lead to false predictions. Even the force predictions are severely affected due to rigid body distortion as seen in figure 3.16 as well as table 3.2.

Table 3.2: Wheel without rigid body correction (NC) and wheel with rigid body correction (C): position mean squared error and force mean percentage error in accelerated GNS rollout.

Test Example	Position MSE $\times 10^{-4}$		Position MSE $\times 10^{-4}$		Force MPE [%]	
	Rigid Body		Soil Particles		Rigid Body	
Example	Wheel (NC)	Wheel (C)	Wheel (NC)	Wheel (C)	Wheel (NC)	Wheel (C)
1	6.11	0.081	0.0005	0.00014	-269.37	-13.06
2	9.14	0.047	0.0028	0.00066	-6.99	-89.99
3	4.43	0.056	0.017	0.0299	173.180	111.70
4	6.45	0.07	0.0099	0.0013	-140.75	24.89
5	8.43	0.047	0.0009	0.000032	21.466	-78.83
6	5.57	0.0032	0.036	0.01	340.023	233.35
7	6.31	0.0035	0.042	0.017	659.116	-22.27
8	4.43	0.071	0.017	0.093	173.18	111.71
9	5.39	0.04	0.091	0.013	-104.35	-142.75
Mean	6.25	0.046	0.024	0.018	-	-
MPM	-	-			-0.5	5.2

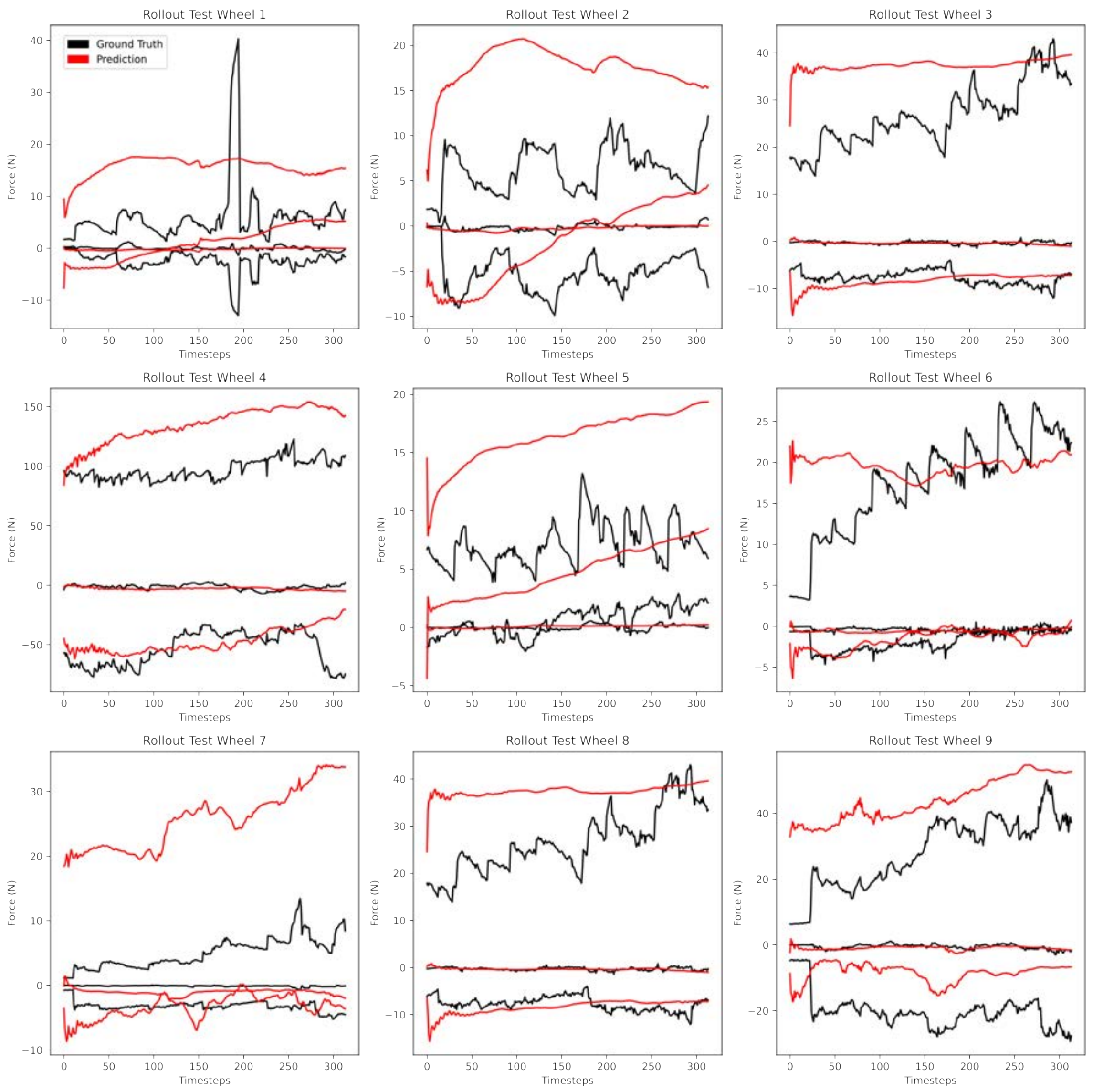
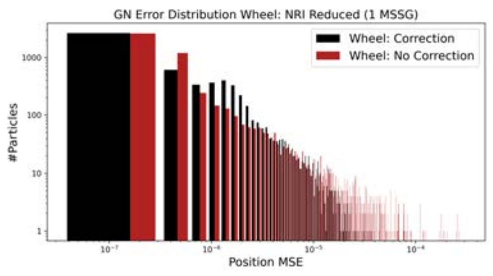
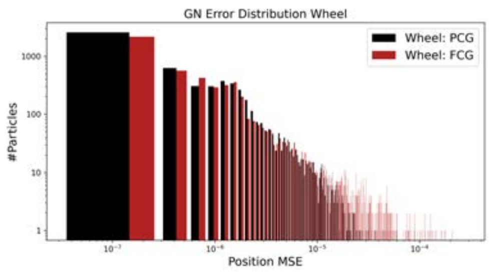


Figure 3.16: Ground truth and prediction of the force in 9 different test examples for wheel driven granular flow simulation without rigid body correction.



(a) Histogram of error between particle positions of the prediction with and without rigid body correction



(b) Histogram of error between particle positions of the prediction when using fully connected graph (FCG) and partially connected graph (PCG)

Figure 3.17: Histogram of error between particle positions of the ground truth and prediction of the wheel driven granular flow.

Some more results. The histogram in figure 3.17a visualizes the mean squared error in the particle positions with and without rigid body correction. The soil particle positions are achieved by applying inverse PCA to the rollout generated using GNS. It can be seen in the histogram that rigid body correction using singular value decomposition (SVD) significantly reduces the error in the prediction of the particle positions. Figure 3.17b compares the error prediction between ground truth and prediction for different graph inputs: fully connected graph (Haeri & Skonieczny, 2021) and partially connected graph. The histogram shows that the partially connected graph with the rigid body correction has slightly lower error than the fully connected graph.

Figure 3.18 and figure 3.19 depict the excavation driven granular flows for different depth, angle and speeds of the excavation blade at the 5 second mark. The red to black colors represent the velocity of the particles while blue to green colors represent the magnitude of the force acting on the rigid body and the arrow represents the direction of the force. Some more examples of wheel driven granular flows are shown in figure 3.20 and figure 3.21 where wheel of different diameter in different gravity levels such as earth, lunar and martian are depicted at the 5 second mark.

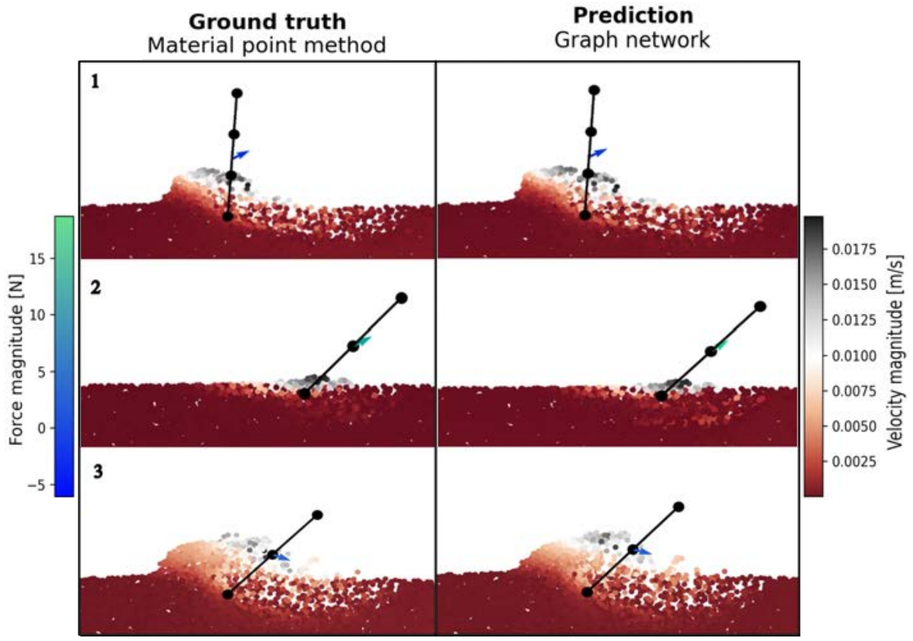


Figure 3.18: Ground truth and prediction of 3 different test examples for excavation driven granular flow simulation at time 5 seconds.

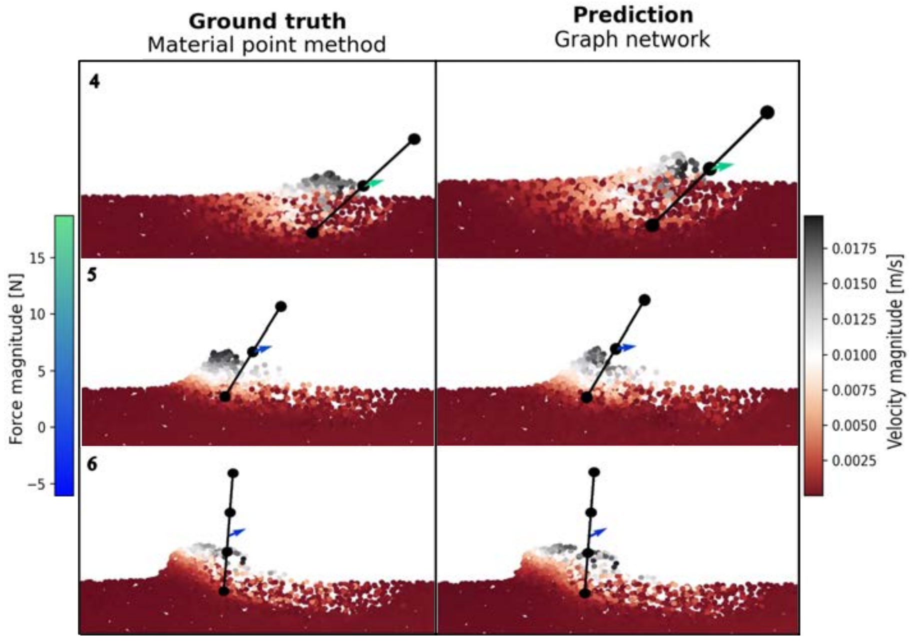


Figure 3.19: Ground truth and prediction of 3 different test examples for excavation driven granular flow simulation at time 5 seconds.

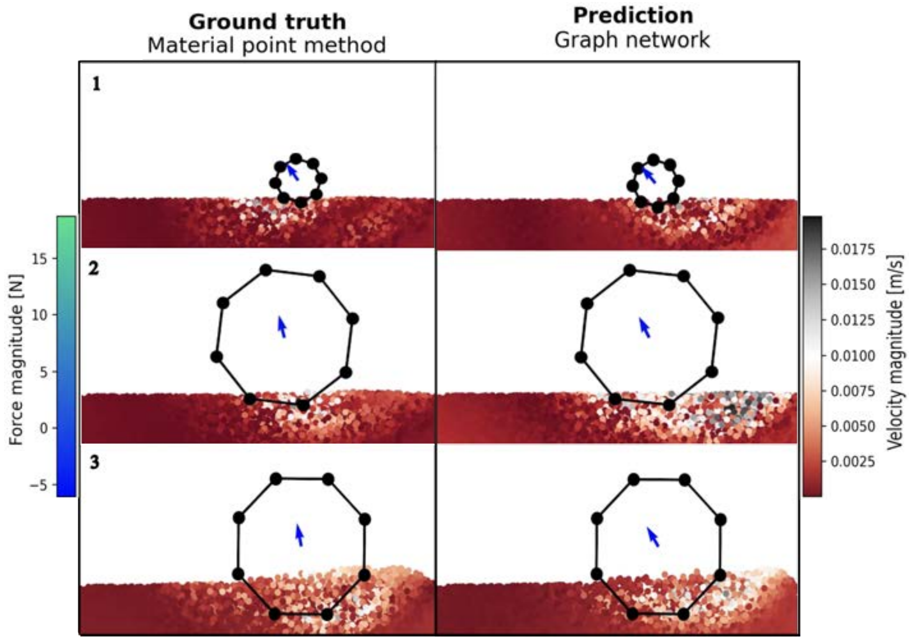


Figure 3.20: Ground truth and prediction of 3 different test examples for wheel driven granular flow simulation with rigid body correction at time 5 seconds.

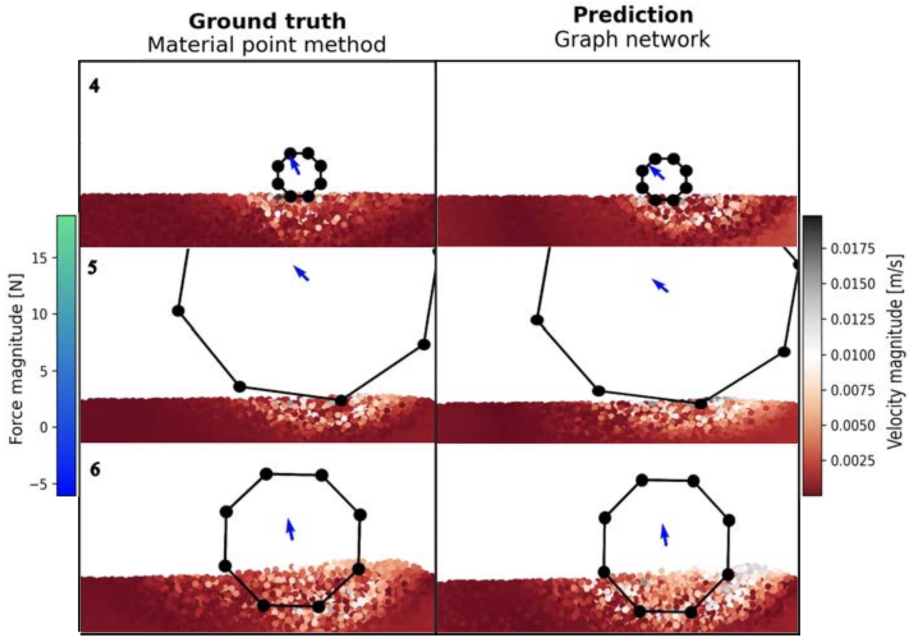


Figure 3.21: Ground truth and prediction of 3 different test examples for wheel driven granular flow simulation with rigid body correction at time 5 seconds.

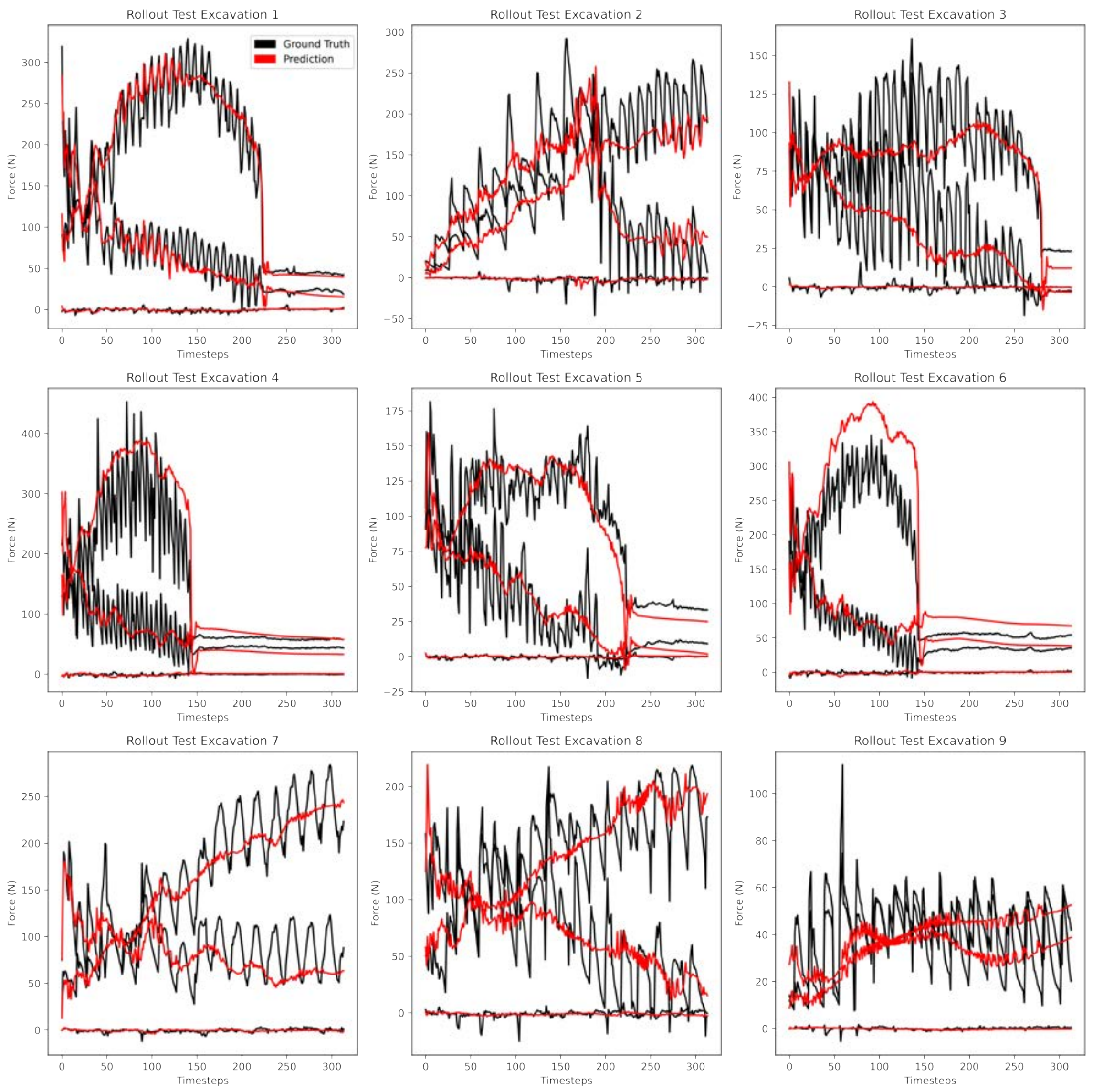


Figure 3.22: Ground truth and prediction of the force in 9 different test examples for excavation driven granular flow simulation.

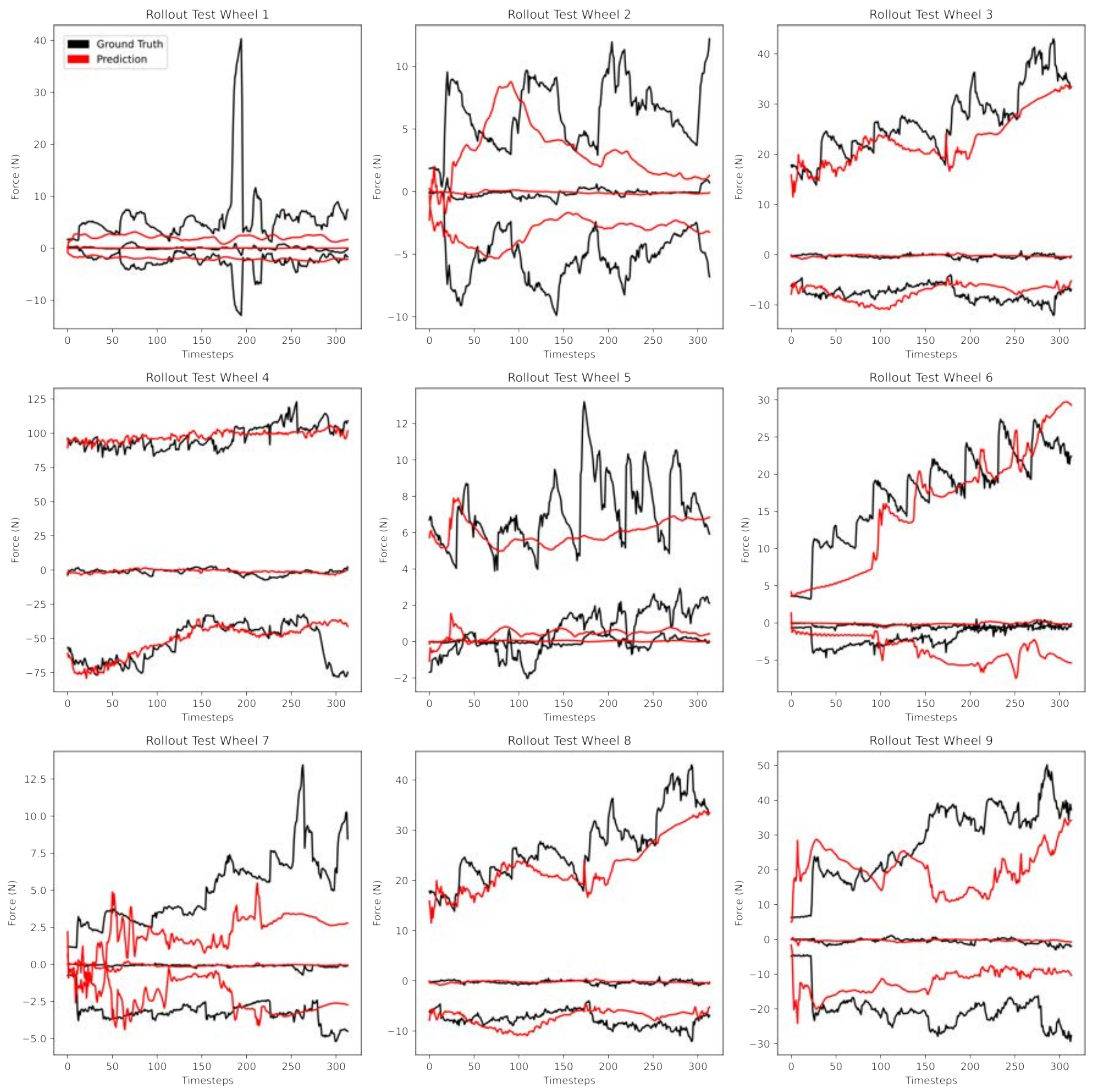


Figure 3.23: Ground truth and prediction of force in 9 different test examples for wheel driven granular flow simulation with rigid body correction.

In Table 3.3, we present an assessment of accuracy concerning both excavation and wheel-driven granular flows with rigid body correction. This assessment is based on a comparison of mean squared error (MSE) for particle positions and mean percentage error (MPE) for the forces acting on the rigid body. Notably, when transitioning from a fully connected graph to a partial graph, we observe an increase in force prediction errors for both excavation and wheel examples and a reduction in the particle position error. This phenomenon can primarily be attributed to the reduced number of edges available for message passing which makes it challenging for GNS to predict the forces.

Table 3.3: Wheel and Excavation (Exc.) position mean squared error and force mean percentage error in accelerated GNS rollout with fully connected graph (FCG) (Haeri & Skonieczny, 2021) and partially connected graph (PCG).

Test Example	Position MSE $\times 10^{-4}$				Force MPE [%]			
	FCG		**PCG**		FCG		**PCG**	
Example	Exc.	Wheel	**Exc.**	**Wheel**	Exc.	Wheel	**Exc.**	**Wheel**
1	0.01	0.01	0.049	0.00014	4.5	10.9	-62.71	-13.06
2	0.01	0.02	0.003	0.00066	-5.3	-16.2	-30.88	-89.99
3	0.00	0.00	0.03	0.0299	0.3	-18.5	-1.62	111.70
4	0.02	0.03	0.027	0.0013	-10.5	-22.8	397.24	24.89
5	2.41	0.05	0.032	0.000032	11.6	-21.7	82.13	-78.83
6	0.02	0.09	0.057	0.01	-2.4	-5.1	-28.30	233.35
7	2.86	0.03	0.027	0.017	-17.2	0.2	-69.30	-22.27
8	0.01	0.02	0.001	0.093	-1.9	26.6	-179.33	111.71
9	0.03	0.03	0.007	0.013	-5.1	-31.0	-54.02	-142.75
Mean	0.60	0.03	0.025	0.018	-	-	-	-
MPM	-	-	-	-	-	-	-0.5	5.2

For the excavation example, the number of edges was reduced from an initial 784 to 188, representing a significant reduction of 76.02%. Similarly, for the wheel example, the graph size decreased from 576 to 336, reflecting a 41.66% reduction in the number of edges. It is worth noting, however, that despite these substantial reductions in graph complexity, the visual error during the final rendered rollout remains minimal and does not significantly impact the overall visual representation of the results.

Chapter 4

Conclusion and Future Work

This research focused on accelerating the performance the performance of the graph network simulator for rigid body driven granular flows on two main cases: excavation which emulates construction vehicles excavating sand and wheel-terrain interaction in microgravity for simulating the rover wheel interaction on extra-terrestrial bodies. This work extends the work developed by (Haeri & Skonieczny, 2021) where using graph neural networks and principal component analysis (PCA), the authors achieved real time performance with high accuracy.

However, the final adjacency graph was a fully connected graph and this work addresses on finding the most optimal partial graph by getting rid of the redundant edges which do not effectively contribute towards the learning of the subspace GNS. We used the Neural Relational Inference (NRI) model, based on a variational autoencoder, to identify the most important edges that contribute to the interaction between the subspace dynamics of the dataset. The variational autoencoder compresses the input data to a compressed latent space, and edge sampling reveals the most important connection and eliminates the redundant ones. The development of the final partial graph framework involved the NRI model predicting the interaction between PCA modes. Thus, this partial graph framework for training and inference resulted in a 3x acceleration from the original subspace GNS for the case of excavation and a 2x acceleration for wheel driven granular flow simulation. Quantitative measurement of the results shows that the force prediction error is higher in the case of the partial graph compared to the fully connected graph. The partial graph framework for the excavation example reduced the number of edges from 784 to 188, indicating a 76.02% decrease.

Similarly, in the wheel example, the graph size was reduced from 576 to 336, indicating a 41.66% drop in edges. Notably, the mean squared error of the particle positions in each frame during the final generated rollout remains rather small and has minimal to no effect on the overall visual representation of the simulation, even with these significant reductions in graph connections. Thus, there is a tradeoff when using a sparser adjacency matrix for training the graph network simulator as compared to a fully connected graph which has better force predictions. In the case where the force predictions are not as important, the partial graph framework can be used for computational efficiency due to its smaller adjacency graph. However, in situations where force predictions are essential, such as providing haptic feedback to a user through the simulator, using a fully connected graph is recommended for better accuracy in force predictions.

4.1 Limitations and future work

With the assistance from neural relational inference, the partial graph framework sped up the simulation, leading to faster inference time in the case of rigid body driven granular flow for excavation and wheel in a dynamic environment. However, further improvements can reduce the inference time even more.

- The significant reduction in the graph size uses fewer edges during message passing steps, making it difficult for the network as the GNS with a partially connected graph demonstrated lower accuracy in force predictions compared to the fully connected method. However, the simulation of the rollout shows no significant visual error which is computed by the mean squared error of the particle positions in the simulation. Y. Shao et al. (2022) developed a transformer-based simulator to improve accuracy as an alternative to GNN based methods due to their computational complexity when increasing the graph size. This model predicts individual particles, unlike GNN based models, which learn the pairwise interaction. Thus, the use of this model with subspace data could result in faster inference time.
- Recent studies (Tolstikhin et al., 2021) demonstrate that the MLP-mixer architecture exhibits comparable performance to transformer-based architectures (Vaswani et al., 2017). Unlike

transformer architectures that demand substantial computational power with increasing input data and sequence length, the MLP-Mixer architecture eliminates attention-based mechanisms to address this challenge. This innovative approach projects the data into a higher-dimensional space using multi-layer perceptrons (MLP) where the MLP-Mixer architecture generates tokens, which undergo processing within the MLP-Mixer block. Through the use of MLP, this block learns temporal and spatial features within the dataset. The final tokens are subsequently directed to the classification head for the purpose of classification. (Bouazizi, Holzbock, Kressel, Dietmayer, & Belagiannis, 2022) developed the motion mixer architecture to analyze and predict human pose in three dimensions. This architecture leverages the MLP-Mixer design to address a time-series forecasting challenge instead of classification.

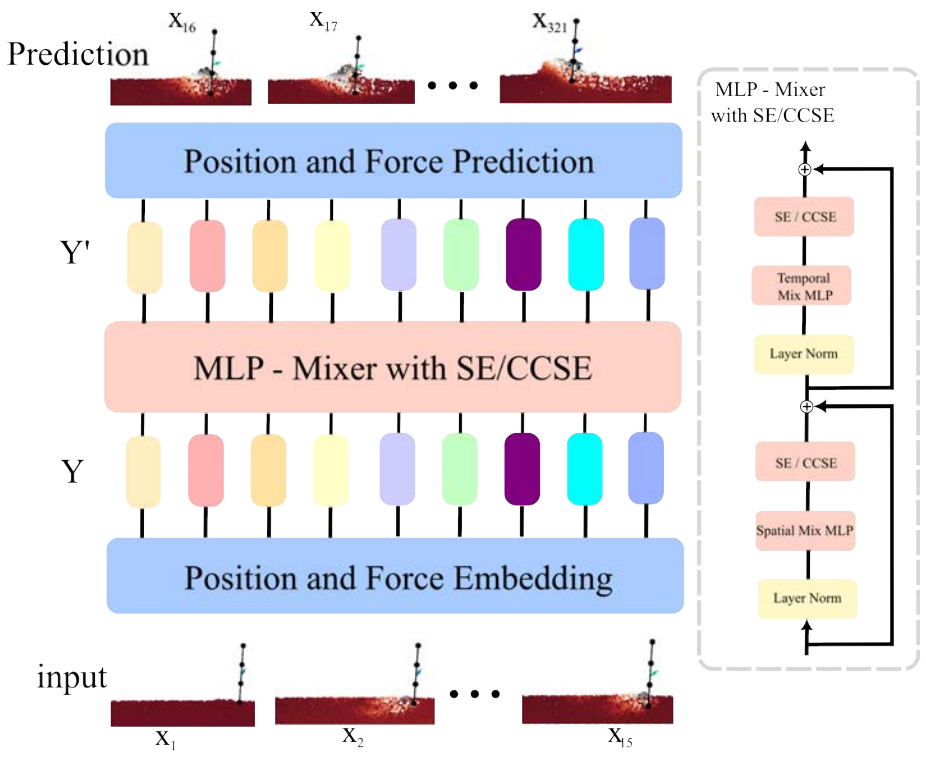


Figure 4.1: Motion Mixer architecture for excavation datasets. Here SE is squeeze and excitation and CCSE is concurrent channel and spatial excitation.

Incorporating a squeeze and excitation block (SE block) (Hu, Shen, Albanie, Sun, & Wu, 2019), motion mixer ranks spatial and temporal features from the spatial-mix MLP and temporal-mix MLP. The SE block architecture imposes minimal computational overhead, making it adaptable to enhance the performance of any baseline architecture. The SE block, employing global average pooling, serves as an attention mechanism, eliminates spatial dependency, and generates a descriptor based on the channel. This descriptor aids in re-ranking the feature map to focus on more critical channels. Given its success in forecasting human pose, we used this architecture to the context of excavation-driven granular flows. During training, the input to the model was a segment of the position along with forces on the particles and objective is to predict the subsequent segment. In the inference phase, the model is configured to predict the position of soil particles and the forces acting on the rigid body. As it only uses MLPs for learning, the inference time is faster as compared to graph neural networks. In the case of excavation, it takes on an average around 96 ms to generate one second of simulation which is faster than our current subspace GNS using partial graphs which is roughly between 120ms and 150 ms. However, the default architecture proposed is not able to efficiently learn the dynamics as seen in figure 4.1. The embedding dimensions were increased to 1218 and the squeeze and excitation block was replaced by concurrent channel and spatial excitation (CCSE) block (Roy, Navab, & Wachinger, 2018).

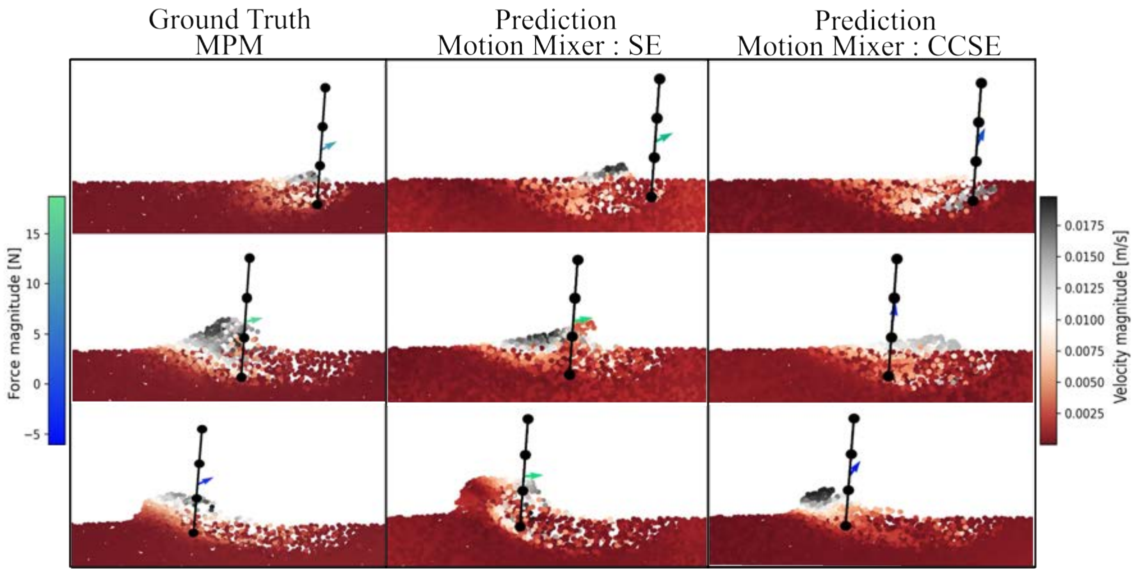


Figure 4.2: Motion Mixer prediction for excavation datasets at time 1 sec, 3 sec and 5 sec. Here SE is squeeze and excitation block and CCSE is concurrent channel and spatial excitation.

Instead of just ranking the spatial features, the CCSE block simultaneously identifies the most important features for both the channel as well as the spatial features and then finally combines the output. With these modifications, the model is able to generate simulation but with a lag and flicker when rendering the simulation as seen in figure 4.2. More research in developing the motion mixer architecture for rigid body driven granular flow can reduce the inference time as MLP-Mixer just uses MLPs which requires less computation power. Specifically, modifying or tuning the SE block or the CCSE block can improve its performance on subspace data as seen in figure 4.2. Modification of the initial embedding layer can also lead to better overall prediction.

- In the Graph Network Simulator (GNS), the partial graph remains static and is hardcoded during both the training and rollout phases, necessitating a two-step process. However, recent studies by Kovalenko, Pozdnyakov, and Makarov (2022) and Ahmad, Jalil, Nazir, and Taj (2023) have proposed various methods for learning the adjacency graph during model training, yielding promising results. Therefore, if the training process can identify and learn the graph, it could streamline the training and inference processes in the GNS.

- To further reduce the edges in the adjacency graph, a dynamic graph can be used based on the interaction between soil particles and the particles representing the rigid body. The number of rigid body particles interacting with the soil varies depending on factors like depth of the wheel or the excavation blade, eliminating the necessity for a message passing step. This would require performing inverse PCA to the predictions generated from the GNS and identifying which rigid body particles are nearest to the soil particles during dynamic construction of the adjacency graph. Employing this approach can lead to a reduction in graph size and improvement in inference time.

- The codebase of the GNS is written using custom libraries and tensorflow in python. Pythons runtime performance is considerably slower because it uses a global interpreter lock (GIL) that uses a single core in the CPU, even if multi-core is available. Python, being a dynamically typed language, does not pre-define variables and checks them during run-time, which also slows down the final execution time (Meier & Gross, 2019). Mojo, an innovative programming language (Xurshid, 2023), uses a Python-like syntax and builds on the Multi-Level

Intermediate Representation (MLIR). Mojo generates machine-level code, following a compilation approach akin to the C programming language. Mojo, tailored specifically for AI applications, is compatible with major Python libraries, including but not limited to NumPy, scikit-learn, and SciPy, as well as popular deep learning frameworks like TensorFlow and PyTorch. This feature incorporates existing Python code into Mojo with minimal modifications. Exploring and developing the codebase in Mojo could potentially accelerate the inference time of the subspace GNS.

Appendix A

Notations

ϵ	Random variable drawn from a standard normal distribution
κ	Kernel function
λ	Eigenvalue
μ	Mean
$\mathcal{N}$	Graph node neighbourhood
ϕ	Feature update using neural network
ρ	Message passing or feature aggregation
τ	Temperature in gumbel softmax
θ	GNS model parameters
σ	Variance
a	Node/subspace particle acceleration
C	Covariance
d	Translation matrix
D	Diagonal matrix

e	Edges in the graph
E	Expectation
f	Force
F	Force
g	Gumbel softmax
G	Graph
h_e	Graph edge embeddings
h_u	Graph global embeddings
h_v	Graph node embeddings
K	Kernel matrix
n	Number of dimensions in data
N_e	Number of graph edges
N_k	Number of message-passing steps
N_v	Number of graph nodes
q	Approximate distribution
r_i	Index of graph receiver node of edge i
r	Number of subspace modes
R	Rotation matrix
s_i	Index of graph sender node of edge i
u	Graph global feature
v	Nodes in the graph

$\mathcal{V}$ Set of graph node features or embeddings

W Eigenvectors

Y Mean centered data

z Latent variable

Z Reduced flow and rigid states

References

Ahmad, O., Jalil, O. A., Nazir, U., & Taj, M. (2023). *Learning adjacency matrix for dynamic graph neural network.*

Allen, K. R., Guevara, T. L., Rubanova, Y., Stachenfeld, K., Sanchez-Gonzalez, A., Battaglia, P., & Pfaff, T. (2023, 14–18 Dec). Graph network simulators can learn discontinuous, rigid contact dynamics. In K. Liu, D. Kulic, & J. Ichnowski (Eds.), *Proceedings of the 6th conference on robot learning* (Vol. 205, pp. 1157–1167). PMLR. Retrieved from https://proceedings.mlr.press/v205/allen23a.html

Bakir, G. H., Weston, J., & Scholkopf, B. (2003). Learning to find pre-images. In *Neural information processing systems.* Retrieved from https://api.semanticscholar.org/CorpusID:80486

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., … Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *CoRR, abs/1806.01261.* Retrieved from http://arxiv.org/abs/1806.01261

Bouazizi, A., Holzbock, A., Kressel, U., Dietmayer, K., & Belagiannis, V. (2022). *Motionmixer: Mlp-based 3d human body pose forecasting.*

Gao, X., Zhang, W., Shao, Y., Nguyen, Q. V. H., Cui, B., & Yin, H. (2022). *Efficient graph neural network inference at large scale.*

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017, 06–11 Aug). Neural message passing for quantum chemistry. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (Vol. 70, pp. 1263–1272). PMLR. Retrieved from https://proceedings.mlr.press/v70/gilmer17a.html

Gundersen, G. (2018, Apr). *The reparameterization trick.* Retrieved from `https://gregorygundersen.com/blog/2018/04/29/reparameterization/#doersch2016tutorial`

Haeri, A., & Skonieczny, K. (2021). *Subspace graph physics: Real-time rigid body-driven granular flow simulation.*

Haeri, A., & Skonieczny, K. (2022, May). Three-dimensionsal granular flow continuum modeling via material point method with hyperelastic nonlocal granular fluidity. *Computer Methods in Applied Mechanics and Engineering*, *394*, 114904. Retrieved from `https://doi.org/10.1016/j.cma.2022.114904` doi: 10.1016/j.cma.2022.114904

Holden, D., Duong, B. C., Datta, S., & Nowrouzezahrai, D. (2019, July). Subspace neural physics. In *Proceedings of the 18th annual ACM SIGGRAPH/eurographics symposium on computer animation.* ACM. Retrieved from `https://doi.org/10.1145/3309486.3340245` doi: 10.1145/3309486.3340245

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*, 498-520. Retrieved from `https://api.semanticscholar.org/CorpusID:144828484`

Hu, J., Shen, L., Albanie, S., Sun, G., & Wu, E. (2019). *Squeeze-and-excitation networks.*

Karpman, E., Kövecses, J., Holz, D., & Skonieczny, K. (2020, October). Discrete element modelling for wheel-soil interaction and the analysis of the effect of gravity. *Journal of Terramechanics*, *91*, 139153. Retrieved from `http://dx.doi.org/10.1016/j.jterra.2020.06.002` doi: 10.1016/j.jterra.2020.06.002

Kasim, M. F., Watson-Parris, D., Deaconu, L., Oliver, S., Hatfield, P., Froula, D. H., … Vinko, S. M. (2021, December). Building high accuracy emulators for scientific simulations with deep neural architecture search. *Machine Learning: Science and Technology*, *3*(1), 015013. Retrieved from `http://dx.doi.org/10.1088/2632-2153/ac3ffa` doi: 10.1088/2632-2153/ac3ffa

Kingma, D. P., & Ba, J. (2017). *Adam: A method for stochastic optimization.*

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, *abs/1312.6114*. Retrieved from `https://api.semanticscholar.org/CorpusID:216078090`

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., & Zemel, R. (2018). *Neural relational inference for interacting systems.*

Kovács, L. L., Ghotbi, B., González, F., Niksirat, P., Skonieczny, K., & Kövecses, J. (2019). Effect of gravity in wheel/terrain interaction models. *Journal of Field Robotics*, *37*, 754 - 767. Retrieved from `https://api.semanticscholar.org/CorpusID:198939932`

Kovalenko, A., Pozdnyakov, V., & Makarov, I. (2022). *Graph neural networks with trainable adjacency matrices for fault diagnosis on multivariate sensor data.*

Li, J., Gao, Y., Dai, J., Li, S., Hao, A., & Qin, H. (2023). Mpmnet: A data-driven mpm framework for dynamic fluid-solid interaction. *IEEE Transactions on Visualization and Computer Graphics*, 1-14. doi: 10.1109/TVCG.2023.3272156

Li, Z., & Farimani, A. B. (2022, April). Graph neural network-accelerated lagrangian fluid simulation. *Computers &amp Graphics*, *103*, 201–211. Retrieved from `https://doi.org/10.1016/j.cag.2022.02.004` doi: 10.1016/j.cag.2022.02.004

Li, Z., Li, H., & Meng, L. (2023, Mar). Model compression for deep neural networks: A survey. *Computers*, *12*(3), 60. Retrieved from `http://dx.doi.org/10.3390/computers12030060` doi: 10.3390/computers12030060

Lino, M., Fotiadis, S., Bharath, A. A., & Cantwell, C. (2022). *Remus-gnn: A rotation-equivariant model for simulating continuum dynamics.*

Liu, C., Ma, X., Zhan, Y., Ding, L., Tao, D., Du, B., ... Mandic, D. (2022). *Comprehensive graph gradual pruning for sparse training in graph neural networks.*

McCullough, M., Jayakumar, P., Preston-Thomas, J., Hodges, H., & Shoop, S. A. (2017). Simple terramechanics models and their demonstration in the next generation nato reference mobility model.. Retrieved from `https://api.semanticscholar.org/CorpusID:126226922`

Meier, R., & Gross, T. R. (2019). Reflections on the compatibility, performance, and scalability of parallel python. In *Proceedings of the 15th acm sigplan international symposium on dynamic languages* (p. 91103). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3359619.3359747` doi: 10.1145/3359619.3359747

Nakashima, H. (2011). Soil–wheel interactions. In *Encyclopedia of agrophysics* (pp. 810–813). Springer Netherlands. Retrieved from `https://doi.org/10.1007/978-90-481-3585-1_256` doi: 10.1007/978-90-481-3585-1_256

Niksirat, P., Daca, A., & Skonieczny, K. (2020, May). The effects of reduced-gravity on planetary rover mobility. *The International Journal of Robotics Research*, *39*(7), 797–811. Retrieved from `https://doi.org/10.1177/0278364920913945` doi: 10.1177/0278364920913945

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., & Battaglia, P. W. (2021). *Learning mesh-based simulation with graph networks.*

Roy, A. G., Navab, N., & Wachinger, C. (2018). *Concurrent spatial and channel squeeze excitation in fully convolutional networks.*

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation.. Retrieved from `https://api.semanticscholar.org/CorpusID:62245742`

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P. (2020, 13–18 Jul). Learning to simulate complex physics with graph networks. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (Vol. 119, pp. 8459–8468). PMLR. Retrieved from `https://proceedings.mlr.press/v119/sanchez-gonzalez20a.html`

Schlkopf, B., Smola, A., & Mller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*(5), 1299-1319. doi: 10.1162/089976698300017467

Shao, H., Kugelstadt, T., Hädrich, T., Pałubicki, W., Bender, J., Pirk, S., & Michels, D. (2021). *Accurately solving rod dynamics with graph learning.* Retrieved from `https://openreview.net/forum?id=v2tmeZVV9-c`

Shao, Y., Loy, C. C., & Dai, B. (2022). Transformer with implicit edges for particle-based physics simulation. In *Lecture notes in computer science* (pp. 549–564). Springer Nature Switzerland. Retrieved from `https://doi.org/10.1007/978-3-031-19800-7_32` doi: 10.1007/978-3-031-19800-7_32

Shi, G., Zhang, D., Jin, M., Pan, S., & Yu, P. S. (2023). *Towards complex dynamic physics system* *simulation with graph neural odes.*

Skonieczny, K., Niksirat, P., & Nassiraei, A. A. F. (2017). Preparations for reduced gravity flights to examine exomars rover wheel-soil interactions.. Retrieved from `https://api.semanticscholar.org/CorpusID:211125498`

Skonieczny, K., Niksirat, P., & Nassiraei, A. A. F. (2019, June). Rapid automated soil preparation for testing planetary rover-soil interactions aboard reduced-gravity aircraft. *Journal of Terramechanics*, *83*, 35–44. Retrieved from `https://doi.org/10.1016/j.jterra.2019.03.001` doi: 10.1016/j.jterra.2019.03.001

Söderkvist, I., & Wedin, P.-Å. (1994, September). On condition numbers and algorithms for determining a rigid body movement. *BIT*, *34*(3), 424–436. Retrieved from `https://doi.org/10.1007/bf01935651` doi: 10.1007/bf01935651

Sorkine-Hornung, O., & Rabinovich, M. (2017). Least-squares rigid motion using svd. *Computing*.

Tailor, S. A., Fernández-Marqués, J., & Lane, N. D. (2020). Degree-quant: Quantization-aware training for graph neural networks. *CoRR*, *abs/2008.05000*. Retrieved from `https://arxiv.org/abs/2008.05000`

Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., ... Dosovitskiy, A. (2021). Mlp-mixer: An all-mlp architecture for vision. *CoRR*, *abs/2105.01601*. Retrieved from `https://arxiv.org/abs/2105.01601`

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. Retrieved from `http://arxiv.org/abs/1706.03762`

Wong, J. (2012, February). Predicting the performances of rigid rover wheels on extraterrestrial surfaces based on test results obtained on earth. *Journal of Terramechanics*, *49*(1), 4961. Retrieved from `http://dx.doi.org/10.1016/j.jterra.2011.11.002` doi: 10.1016/j.jterra.2011.11.002

Xurshid, M. (2023). Differences between mojo and python programming languages. In *Conference on digital innovation:" modern problems and solutions"*.