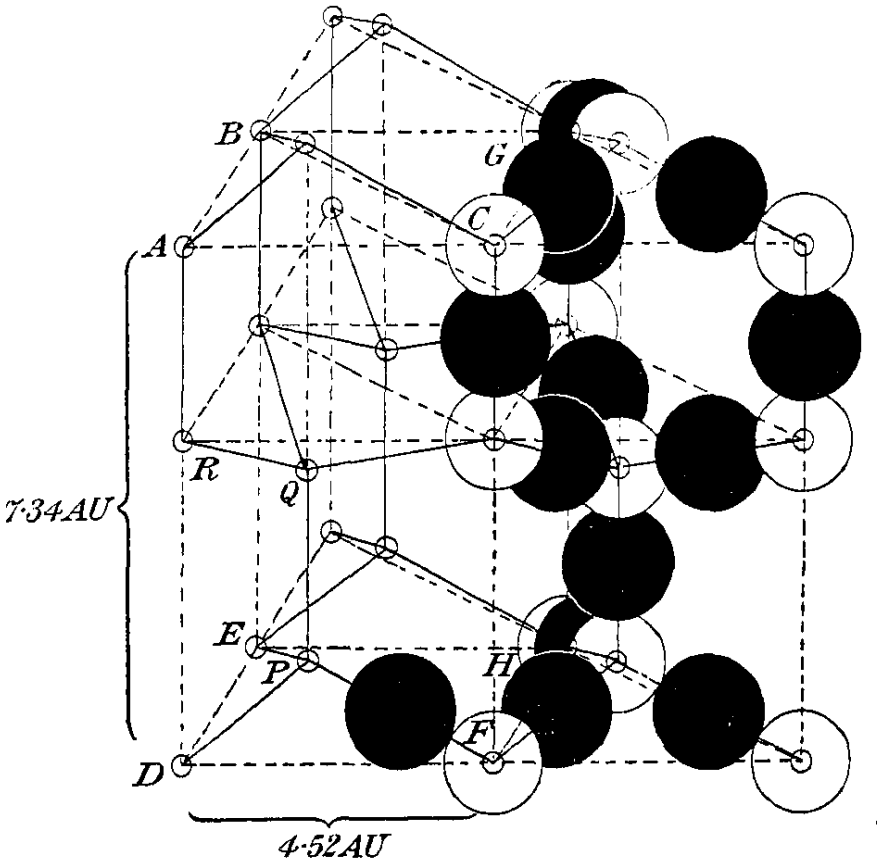
GRAPH NEURAL NETWORKS TO LEARN MESHFREE SNOW SIMULATIONS

MSC. THESIS



GRAPH NEURAL NETWORKS TO LEARN MESHFREE SNOW SIMULATIONS

MSC. THESIS

by

Joseph VAN LINN

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly Monday 3 July 2023 at 14:00

Student Number:	5459192	
Daily supervisor:	Bruno Ribeiro	
Project duration:	July, 2022 - July, 2023	
Thesis committee:	Dr. M.H.F. Sluiter,	TU Delft, 3ME faculty
	Dr. S. Kumar,	TU Delft, 3ME faculty
	Dr. B. Giovanardi,	TU Delft, Aerospace faculty
	Dr. M.A. Bessa,	Brown University, School of Engineering

TU Delft Delft University of Technology

Keywords: graph neural networks, snow science, material point method

Front Cover: The crystal structure of ice is shown. It exemplifies symmetry and graph structure. Image Bragg, W. H. [1]

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

Any fool can make something complicated.
It takes a genius to make it simple.

-Woody Guthrie

SUMMARY

Snow is a natural hazard to human life and infrastructure. This motivates current research efforts to understand the granular material. The material point method models snow as a continuum. Application length scales range from the microstructural level to full scale avalanches. This conventional numerical method relies on solely spatially local information to make local updates. The recent graph neural network machine learning model is shown to include both local and global information in making local updates. This model's promising attribute motivates its use to replace the conventional snow simulation method. However, it is uncertain if current graph neural network applications to learn physical simulations truly learn the underlying physics. This work is inspired by the finite element community's patch-test proposed in the 1960s. This insight is used to reimagine the means a graph neural network model is evaluated. Through this novel evaluation choice, may the model be investigated on the core properties of numerical methods. Further, a state-of-the-art graph neural network model is improved to utilize unnormalized features and targets in making stable predictions. Future research recommends these machine learning models in this application make architecture design choices such that the core properties of conventional numerical methods are met.

ACKNOWLEDGEMENTS

Single handily this document, the embodiment of a pilgrimage pursued at this institution, is made possible by my mother. Thanks to this individual, I took the significant leap into the unknown when hopping the pond and embarking upon an institution which I knew nothing of, no one attending(ed), and no idea of where it would take me. Today, I am humbled by this journey.

As a novice to research, this intangible pursuit quickly opens into a realm of uncertainty. The following individuals were instrumental in reifying this ambiguity: Miguel Bessa, for intently acting as the main advisor on this project, despite geographic constraints. Bruno Ribeiro, for his continual willingness to discuss content, brainstorm ideas and motivate persistence. Marcel Sluiter, for providing a decisive external viewpoint. Jiaxiang Yi, for vital comments on written works. The Bessa Research Group Members, for listening, supporting and bolstering the perseverance of research. Thank you.

The quest of knowledge has taken me from Michigan at large, Spain, Oregon, the greater US West, and now the Netherlands. The breadth of individuals along this journey epitomizes that education is amplified through interacting with those around. My gratitude for you is only made finite in saying,

Tot de volgende keer.

Delft

July 2023

CONTENTS

NOTATION

LIST OF ABBREVIATIONS

AD	Automatic differentiation
CNN	Convectional neural network
CFD	Computational fluid dynamics
CPU	Central processing unit
DEM	Discrete element method
DoE	Design of experiments
FEM	Finite element method
FLIP	Fluid implicit particle method
GDL	Geometric deep learning
GNN	Graph neural network
GNS	Graph network based simulator
GPU	Graphics processing unit
HPC	High performance computer
MAE	Mean absolute error
ML	Machine learning
MLP	Multi-layer perceptron
MLS-MPM	Moving least squares material point method
MPM	Material point method
MPS	Moving particle semi-implicit method
MRR	Mean reciprocal rank
MSE	Mean squared error
PBD	Particle based dynamics
PDE	Partial differential equation
PIC	Particle-in-cell
PINN	Physics informed neural network
RNN	Recurrent neural network
SPH	Smoothed particle hydrodynamics

LIST OF SYMBOLS[1]

E	Young's modulus
$\mathbf{e}_k$	Graph edge entity
ε	Shape tolerance
F, F_n	Function and its numerical approximate
$\mathbf{g}$, g	Gravity
h	Input sequence length
I	Mass moment of inertia
K	Plate curvature
L, $\mathscr{L}$	Loss
m, m_c	mass, center of mass
n	Eulerian cell width
$\mathcal{N}$	Normal distribution
$\mathcal{O}$	Computational complexity
θ, ϕ	Angle
σ, σ^2	Standard deviation, variance
r	Radius, connectivity radius
^{t_k}	The superscript over dynamic value indicates a batch of values
t, dt	Time, time step
$\mathbf{u}$	Graph global entity
$\mathbf{v}$, $\mathbf{v}_0$	Velocity
$\mathbf{v}_i$	Graph node entity
ν	Poisson's ratio
$\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}$	Position, velocity, acceleration
$\hat{\mathbf{x}}$, $\hat{\hat{\mathbf{x}}}$	Normalized velocity, acceleration
$\tilde{\mathbf{x}}, \tilde{\dot{\mathbf{x}}}, \tilde{\ddot{\mathbf{x}}}$	Position, velocity, acceleration with applied noise

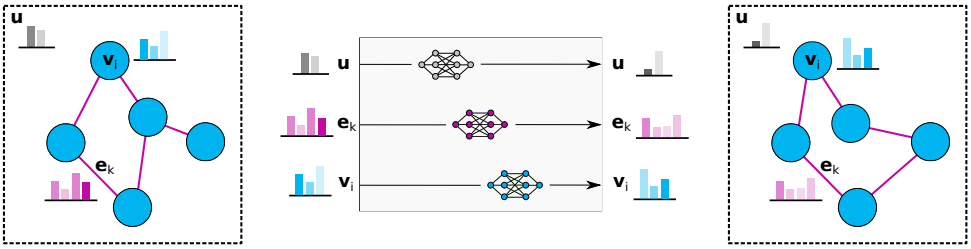
[1] This list is inclusive to the main document, excluding all Appendices. The symbols used in Appendices are inclusive to the respective Appendix.

1

INTRODUCTION

THIS work ties together the two fields of computational snow science and machine learning (ML). At the macroscopic level, snow is modeled using the material point method (MPM). The computational efficiency of this model makes it popular for modeling large domains. However, shortcomings in this model persist. The Graph Neural Network (GNN) model offers the ability to improve these existing shortcomings. This work reimagines the convention that ML models in this application are evaluated on single scalar error metrics. Instead, curated benchmark tests are used to isolate physical quantities of interest. This provides a platform to investigate if these GNN models can truly learn the underlying physics. Further, parallels with the finite difference community are drawn to test for core properties numerical methods should posses.

The earliest micrographs of snow crystals dates back to the 19^{th} century with the work of Wilson A Bentley [2]. Ukitirô Nakaya's work in the 1930's brought about a formal classification of snow crystals [3] and as the field grew, formal review books emerged for the field [4, 5]. In the 70s Cundall and Strack introduced Discrete Element Method (DEM) to computationally model granular materials, and was eventually adopted by the snow science community [6]. In 2013 Stomakhin et al. [7] introduced MPM to the visual animation community for modeling the granular material, and MPM was later implemented by the snow science community [8].



(a) Input graph (b) GNN Processing (c) Output graph

Figure 1.1: Overview of the GNN model

In parallel with these recent advances in snow science comes the emergence of the specific ML model, GNNs, and its field of geometric deep learning (GDL). An overview of the GNN model is shown in Figure 1.1 This model is particularly advantageous due to the following:

1. As inputs GNNs structure data into nodes, edges and globals. This is a similar format seen in the ground truth simulation of particles, relations and system values.
2. They allow spatially local domain updates to be made using both local information and global information.
3. They exploit symmetries in the domain by using the same neural network to update all nodes in the graph.

This thesis balances the limitations of MPM with the idealistic promises of GNNs. Chapter 2 explains the current status in the snow science for modeling the material mechanically and dynamically, followed by a motivation for GNNs. Chapter 3 is inspired by the finite element community's patch test to evaluate the GNN model on first principles a numerical method ought to meet. Chapter 4 shows the reduction of noise in the model can be made by training to position data. Chapter 5 discusses key findings of these results, and why they matter for the current state of the art.

2

LITERATURE REVIEW

This works aims to improve predictions of snow behavior via computer simulations enhanced by ML. Snow is a naturally occurring geographic material which stretches to every corner of this planet. Therefore there are different kind of regimes and conditions under which snow needs to be predicted, such as snowflakes dancing in the sky as they fall to the ground, the destructive impact of avalanches ripping through forests, or the behavior of fallen snow as a neighbor shovels it from their sidewalk. However, there are important limitations when predicting snow behavior by state-of-the-art methods, as reviewed in Section 2.1. ML offers a new route that can potentially address these challenges, as discussed in Section 2.2. This literature review ends with the identification of a knowledge gap that motivates this thesis.

2.1. STATE-OF-THE ART SNOW MODELING

Snow as a material is a porous ice medium full of ice crystal aggregates, with the pores filled with air and water vapor [4, 9]. A defining characteristic of snow compared to other engineering materials is its largely irreversible compression [5]. Wet snow takes the variation of being grains coated with liquid water [4], where the degree of water present alters its mechanical behavior [9]. Snow is often referred to as a granular material [6, 9], a collection of macroscopic particles which are dissipative in nature when interacting [9]. Further, snow is considered thermodynamically unstable at normal environmental conditions [4], leading to its willingness to melt. This work focuses on simulations of the macroscopic dynamics and deformation of snow. This is analogous to the physical situation of throwing a snowball, or rolling over a snowman (Figure 2.3).

In this context, the first and prominent method group to introduce are meshfree methods. These methods' most notable advantage are their ability to capture large and localized deformations accurately due to nodal connectivity occurring as part of the computation, in contrast to being defined *a priori* [10, 11]. Figure 2.1 depicts the manner in which a field variable is updated: (i) an area of interest (support domain) is drawn

around a point of interest (ii) all nodes within this support domain are used to define a shape function (iii) the shape function is used to update the node's field variable (iv) repeat for all nodes. For an in-depth description, please see [10, 12].

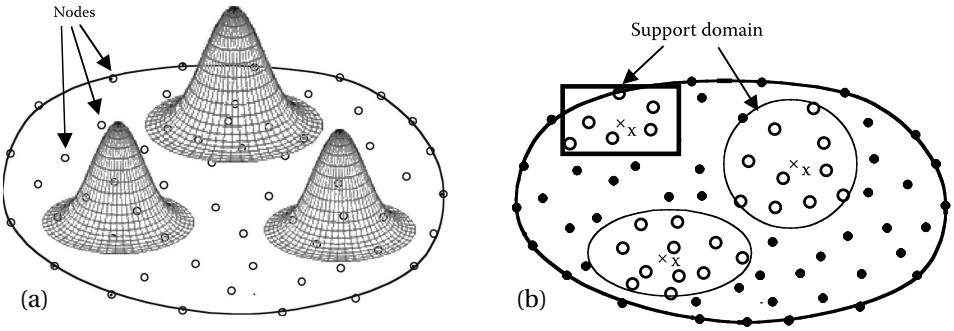


Figure 2.1: (a) Meshfree method weighted support domain (b) Meshfree support domain [12]

MPM, developed in 1994 by Sulsky et al. [13], is a prominent meshfree snow simulation tool. MPM is derived from the particle-in-cell (PIC) method, proposed in 1957 to simulate compressible fluid dynamics problems [14]. MPM is widely applied in the geosciences arena by capturing large deformations at sizeable length scales [15, 16]. MPM uses a background Eulerian mesh which remains unchanged through the entire runtime, while the particles are tracked from a Lagrangian perspective. The Eulerian side is used to compute gradients, and the Lagrangian side tracks the movement of mass and history dependent phenomena, such as plasticity. Being a meshfree method, it handles large deformations well [17]. Figure 2.2 shows 1. the material points at the start of a time step, 2. the transfer to the Eulerian frame via a kernel operation, 3. calculation of the gradients, and 4. return transfer to the Lagrangian frame. See the following texts for an in-depth description of MPM [13, 18, 19].

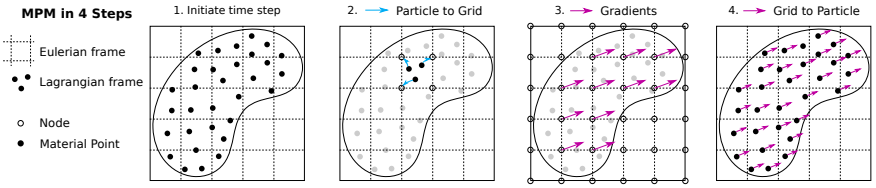


Figure 2.2: The 4 steps of MPM transferring to and from the Eulerian frame

Stomakhin et al. [7] use MPM to simulate snow in a dynamic environment. In collaboration with Disney this work was used in the film *Frozen*. Figure 2.3 displays MPM's ability to capture the appearance of snow as it deforms when interacting with solid objects and itself. MPM is used in other dynamic snow situations and scales to larger dimensions, most notably avalanches [8, 20–22]. Gaume et al. [8] showed the application of MPM to

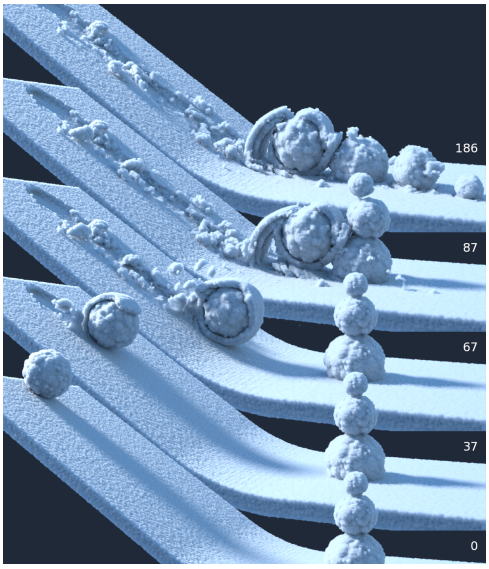


Figure 2.3: Material Point Method Application [7]

an avalanche with a system size of 30 million particles, requiring 5 days to simulate the 10 second video at 48 frames per second.

The Discrete Element Method (DEM) is a fully Lagrangian particle simulation method for granular materials [23], originally proposed in 1979 by Cundall & Strack [24]. The geometry of a system is discretized into individual particle elements. Each particle contains relevant translational and rotational dynamic attributes such as velocity, acceleration and angular velocity. The particles may assume non-spherical shapes, which have stress and strain computed during collisions [25, 26]. The process cycles between a force-displacement law to compute contact forces at particle collisions, and Newton's 2^{nd} law to update particle motion[23]. Figure 2.4 shows the contact diagram for two particles. See the following literature for an extensive explanation of DEM [24, 27].

DEM is used in simulating snow with investigations to the mechanical constitutive models which govern the inter-particle interactions [28, 29]. DEM simulates a range of length scales, from snow microstructure [30] to avalanches [29]. Further, correlation work with experimentally determined failure mechanisms shows DEM's ability to simulate the mechanical behavior of snow [28]. DEM is more computationally expensive than MPM [31], resulting in MPM being chosen at times instead of DEM [29].

Smoothed Particle Hydrodynamics (SPH) is a relevant particle based method originally proposed in the astrophysics field [32, 33]. SPH is a meshfree Lagrangian particle based method used to solve such fluids dynamic problems as the Navier-Stokes partial differential equations [34]. Particle interactions are computed via a kernel function, creating the smoothing effect attributed to the method [34]. SPH is shown in the snow physics community to capture snow avalanches [35, 36], and recently compressible snow be-

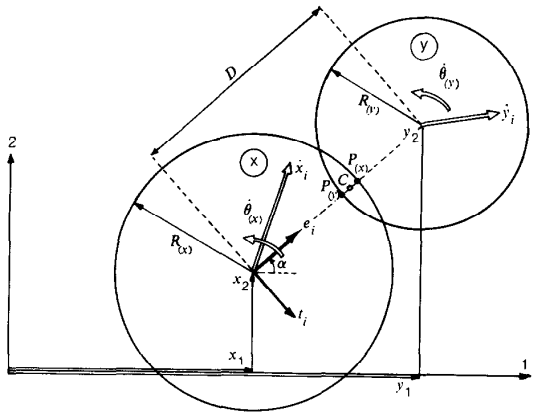


Figure 2.4: DEM force & displacement diagram [24]

havior [37]. In capturing avalanches, particle size and count are varied, capturing up to 256k particles [38]. Known limitations exist with respecting physical boundaries from kernel truncation errors [39].

The Moving Particle Semi-Implicit (MPS), originally developed to simulate incompressible flow, can simulate snow. Proposed in 1996 by Koshizuka and Oka [40], MPS computes interactions of neighboring particles with the constraint of a constant fluid density. User defined governing equations are discretized via particles and solved [41, 42]. The incompressibility assumption restricts this method to applications involving large scale deformations and free surface flows [41, 43, 44]. MPS is applied in simulating avalanches [43, 44], fluidized snow [45] and other geomaterials [46]. The method has fewer sources utilizing it compared to other particle methods such as MPM, DEM and SPH. Snow is characteristic of its irreversible compressibility [5], which is contradictory to MPS's incompressible formulation.

Mesh based methods, such as finite element analyses, are used to model snow [47, 48]. The finite element method (FEM) discretizes the entire geometry into smaller components, elements, which represent algebraic expressions to solve for desired values throughout the problem domain [49]. FEM is used for both mechanical behaviour and thermal analysis.

Finite elements are used to express snow's mechanical performance [47, 48, 50], with a popular application being automotive tire interaction with snow [51, 52]. However, mesh distortion from large continuum deformations is a known method limitation [53]. The SNOWPACK model [54–56] predicts snow mass settlement through the course of a winter. Heat and diffusion transport equations effectively capture the phase transformations of ice, water and corresponding air, while snow accumulation is tracked via a finite element method. The SNOWPACK model tracks large scale snow cover over days and winters to address avalanche risks [54].

Table 2.1: Comparison of parallel computing speedups

Implementation	Model	# CPUs	# GPUs	Speedup
Wang et al. [58]	MPM	-	4	4x
Shigeto et al. [59]	DEM	16	-	5.4x
Yang et al. [60]	SPH	1024	-	900x
Ihmsen et al. [61]	SPH	24	-	10x

Although having different computational complexities, all above-mentioned methods require significant computational resources in practice. Therefore, they are often computed using multiple CPUs or even GPUs in parallel. Parallel computing is necessary for today's modern computation to reduce calculation times as hardware reaches a limit [57]. Table 2.1 gives example speedup magnitudes in current parallel simulation literature. This shows that reducing computational costs through code architecture is a valid avenue.

2.1.1. METHOD COMPARISON AND PHYSICAL MODEL SELECTION

A number of simulation methods have been presented including MPM, FEM and MPS to name a few. These methods show their aptitude to capture the simulation of snow from both phenomenological and physical perspectives. In narrowing the scope of this work, simulation methods are selected based on their ability to capture the macroscopic large deformations and dynamics without considerations of thermal components. This constricts the scope to MPM, DEM, SPH and MPS, which are shown to handle large deformation situations well [7, 29, 35, 44].

SPH, DEM and MPM contain compressibility options to some degree [37, 62, 63]. In contrast, MPS is rooted as an incompressible derivation [40] which is not in alignment with the characterizing tendency of snow to be irreversibly compressible [5]. For this reason, MPS will not be considered. Further, as SNOWPACK and the numerical scheme including heat transport equations do not account for large deformations [54–56, 64, 65], they are not selected for further consideration. Lastly, as finite element methods deal with mesh distortion issues at large deformations, they are not considered for this application.

What remain are 3 particle based methods which are shown to simulate large deformation dynamic snow movement well, namely MPM, DEM and SPH. Table 2.2 details a comparison between all models, listing DEM and MPM as well performing models. MPM has a wider supported snow simulation open source code base in contrast to that of DEM. Due to the open-sourced code for MPM pertaining to snow [66], the method will be used in this work.

2.2. STATE-OF-THE-ART RELEVANT MACHINE LEARNING

A variety of ML methods exist when considering both domains of supervised and unsupervised learning. Relevant ML models are detailed herein. Aptitude is determined

Table 2.2: Comparison of prominent snow simulation methods - adapted from Stomakhin et al. [7]

Method	Volume Preservation	Plasticity	Fracture	Boundary Compliance	Computational Complexity
FEM	∗∗∗	∗∗	∗	∗∗∗	$\mathcal{O}(NW^2)$ [67]
MPS	∗	∗	∗∗∗	∗	$\mathcal{O}(N^2)$ [68]
SPH	∗∗	∗	∗∗∗	∗	$\mathcal{O}(N^{1.5})$ [69]
DEM	∗∗	∗∗∗	∗∗∗	∗∗	$\mathcal{O}(N^2)$ [27]
MPM	∗∗	∗∗∗	∗∗∗	∗∗	$\mathcal{O}(N)$ [70]

from the ability to predict relevant particle dynamics for the particle based simulation methods of MPM, DEM and SPH.

ML is the programming of computers such that they learn from the data presented to them [71]. The three main segments of the ML realm are supervised, unsupervised and reinforcement learning [72]. Supervised, unsupervised and reinforcement learning are optimal for different applications, and herein supervised learning will be used, as the simulation datasets are assumed labeled. Supervised learning maps inputs (features) to outputs (labels, targets) [72]. This domain is directly applicable to particle based simulators, as a current state (feature) is being used to predict the next time step state (target).

A description of rudimentary Neural Networks is shown in Appendix A. Next, networks with an aptitude to learn granular material simulators are addressed.

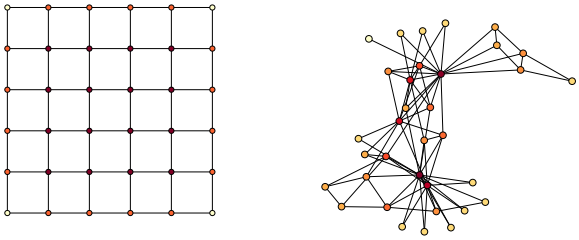
CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) are a neural network type commonly applied to image pattern recognition [73]. The data input is structured as a grid. CNNs are intended for systems where features are spatially dependent, thus the presence of a feature is of interest and not its location [74]. With this feature presence detected, a feature map is yielded as output [72](p.465).

CNNs have shown success in the application of learning fluid simulations [75, 76]. They are shown to learn and predict turbulent flow [75], generalize outside their training domain [77] and reduce the ground truth model computational costs by one to two order of magnitude [76].

GRAPH NEURAL NETWORKS

Instead of the structured data format suited for CNNs, graphs are a discrete data format built of nodes and edges. Figure 2.5 shows two example graphs. In tying this to particle simulations, it is intriguing to note the Eulerian background mesh that MPM uses, and the Lagrangian side which tracks particle movements. Figure 2.6 shows the essential nodes $\mathbf{v}_i$, and edges $\mathbf{e}_k$ connecting said nodes which, together with the global proper-



(a) Grid (Euclidean). (b) Arbitrary graph (Non-Euclidean).

Figure 2.5: Example Graph Visual Representation [78]

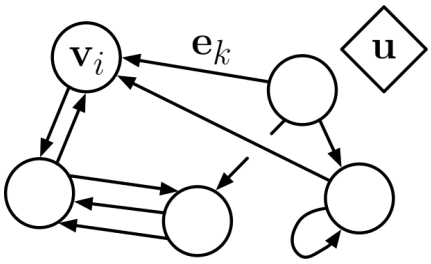


Figure 2.6: Graph Network Example [84]

ties **u**, make up the graph structure. Graphs are distinct from structured images in that they can be heterogeneous, that each node has a varying amount of edges connected to it. In contrast images and such structured data types have a uniform amount of edges attached to every node.

The first GNN models are rooted in the idea that graph nodes represent objects and edges represent their relationships [79, 80]. This idea is later pursued with interaction networks, which reason the way objects in a physical system interact [81]. Li et al. [82] introduced DPI-Nets to model deformable particle based physical simulators. Proroga-tion networks extends this by looking at the extension of just pair-wise interactions and considering the propagation of signals through the nodes of the system [83]. A distinct parallel is seen with particles and their relations, that they are discretized separately, analogous to a graph's nodes and edges. This information format correlation makes GNNs apt to learn and predict complex physical systems [81].

Of notable interest to this project are Graph Network-based Simulators (GNS) proposed by Sanchez-Gonzalez et al. [85] and shown in Figure 2.7. The model learns and predicts particle based fluid simulations, such as MPM, SPH and Particle Based Dynamics (PBD)

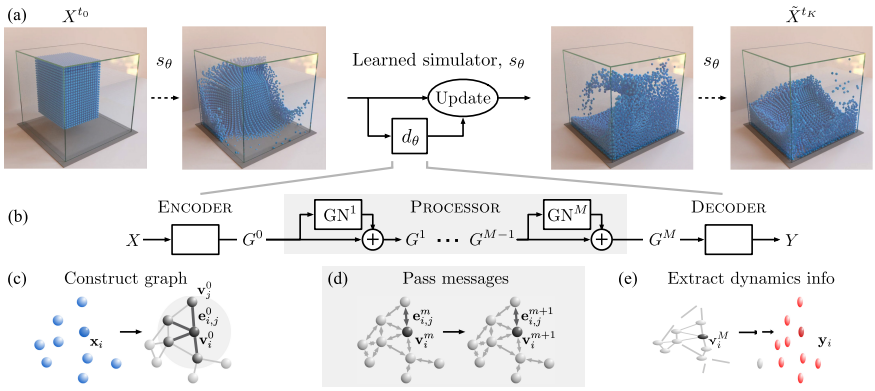


Figure 2.7: GNS Method Framework [85]

for such materials as water, sand and goop. The method uses an Encoder, Processor and Decoder framework which explicitly predicts the next state of particles based on current and previous time steps. The work shows higher performance over ConvNets [77] and DPI-Nets [82]. However, recent work does challenge the GNS model's true ability to learn underlying physics [86, 87]. A similar framework to GNS is adopted specifically for Lagrangian fluid simulation [88].

In contrast to predictions of the explicit update to the particle based on the state dynamics, is learning the underlying physical constraints. Yang et al. [89] first introduced this in learning physical systems. Rubanova et al. [90] drew upon this to develop a Constraint-based Graph Network Simulator (C-GNS) to learn the constraint function landscape, and using its minimum to define the implicit solution update.

PHYSICS INFORMED NEURAL NETWORKS

The models discussed so far have not utilized any existing governing equations as *a-priori* information in training and roll out. Physics-informed neural networks (PINNs), introduced by Raissi et al. [91] in 2017, use governing partial differential equations (PDEs), such as the Navier-Stokes equations as a regularizer in the training stage. When data is lacking for a problem, PINNs are shown to perform with higher accuracy and efficiency compared to a computational fluid dynamics (CFD) solver [92].

Solving a PDE with a neural network requires the parameterized PDE, accounting for the spatial-temporal function, initial conditions, boundary conditions and spatial boundaries. This is converted to a optimization problem to minimize the loss (L), where θ is iteratively updated [92]:

$$L = \omega_1 L_{PDE} + \omega_2 L_{data} + \omega_3 L_{IC} + \omega_4 L_{BC} \tag{2.1}$$

Figure 2.8 shows the framework for a PINN model. This feedforward model approximates the governing equation(s) with a neural network (blue box), feeding into au-

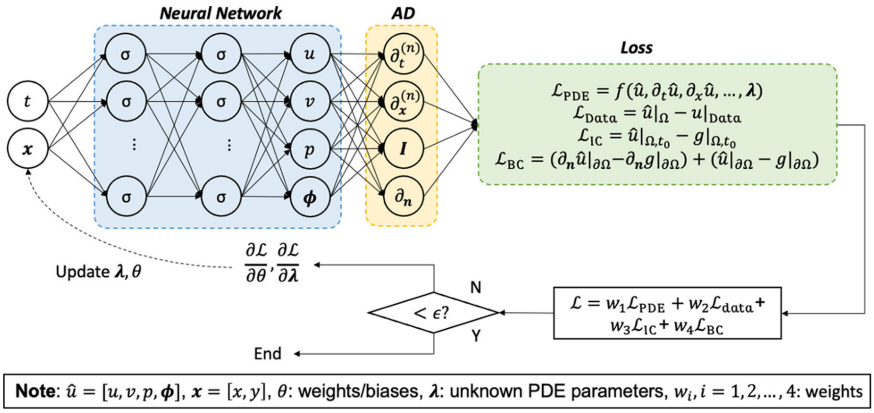


Figure 2.8: PINNs Method Framework [92]

Table 2.3: Example inductive biases found in deep learning models - adapted from [84]

Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	weak	-
Physics informed	Units	All-to-all	PDE	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

tomatic differentiation (AD)(yellow box) to find partial derivatives of the neural network outputs for its inputs, which are then used as residuals in the loss function evaluation. This process learns the unknown parameters of the neural network and the governing PDE. Recently, Shukla et al. [93] stressed the importance of physics informed graph networks (PIGNs) may have in reducing roll-out error and scaling to larger dimensions.

MACHINE LEARNING MODEL SELECTION

In taking a page out of linguistics theory, the Sapir-Whorf hypothesis states that the spoken language one uses alters their perception of the world around them [94]. In the context of ML, the idea of inductive bias has recently been presented, which lays the basis that the type of neural network one uses alters the potential information the ML model captures from the dataset [84, 95]. In the work of Battaglia et al. [84] the inductive bias of GNNs, CNNs and Recurrent Neural Networks (RNNs) are introduced, and shown in Table 2.3.

Battaglia et al. [84] explains that as fully connected layers have all nodes intertwined, allowing the full input signal to reach the output without reuse, there is little inductive bias imposed on the prediction. By contrast, the structured data format input for CNNs results in a spatial translation equivariance, while RNNs use a uniform rule for state updates with time, yielding a time invariant inductive bias. GNNs work to learn the edge

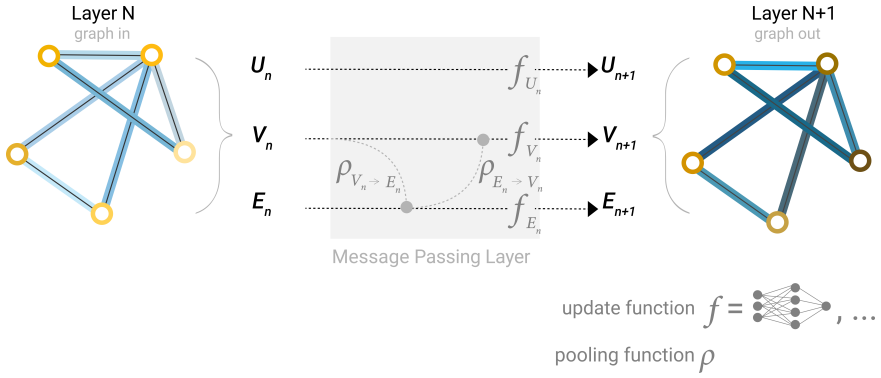


Figure 2.9: Illustration of a simple message passing layer [96]

(connection) relationship, leading to a bias invariant to the input node combination or amount of nodes, instead focusing on how objects interact. Cranmer et al. [95] utilizes the inductive bias of GNNs to symbolically derive the algebraic expression for Newtonian particle interactions.

Remark 1 *GNNs are the ML surrogate model of choice for particle based simulations due to their imperviousness to input permutations and inductive bias of spatial and temporal freedom.*

2.2.1. RECENT GRAPH NEURAL NETWORK DEVELOPMENTS

Figure 2.6 shows the essential nodes $\mathbf{v}_i$, and edges $\mathbf{e}_k$ connecting said nodes which, together with the global properties $\mathbf{u}$, make up the graph structure. However, this is only 3 out of the 4 data types captured by graphs. The fourth is connectivity, the nodes in the graph which are and are not connected.

Graphs, being representations of nodes interconnected, hold information *embedded* in these data types. Using the embedded information of GNN model inputs, a proceeding state is predicted. For example, a molecular dynamics simulation may be modeled as a graph where the molecules are nodes of position and velocity, the edges act as the Lennard-Jones potential and the global property is the total system energy. This physical information is used by the surrogate model to update the next position.

Using this graph structure, Figure 2.9 shows the 3 prediction types which GNNs make. These are edge $\mathbf{e_n}$, node $\mathbf{v_n}$ and global $\mathbf{u_n}$ updates. This figure also shows the specific example of mapping the current state $\mathbf{u_n}$ to following state $\mathbf{u_{n+1}}$ without interaction from the other graph features (nodes and edges). This direct transformation is the simplest GNN prediction. In contrast, allowing the features of the graph to interact during prediction defines the message passing concept.

Gilmer et al. [97] introduced message passing after a recognition of commonalities between existing GNN frameworks. Message passing is proposed as an aggregation of inherent node information, detailed as the following [78, 84, 97, 98]:

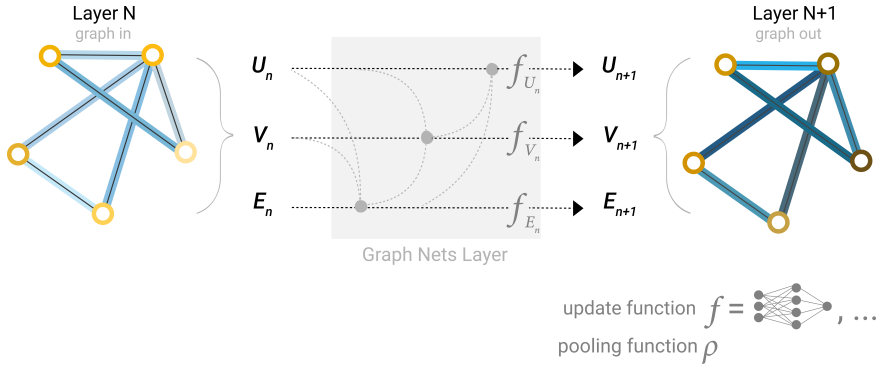


Figure 2.10: Illustration of Graph Nets message passing layer [96]

- Each node $\mathbf{v}_i$ receives a message from all neighbors connected.
  - Nodes further from adjacent nearest neighbors may be done so by adding additional layers which recognize nodes at further distances.
- All received messages are aggregated.
- The node state is updated based on this information.
- The graph's global properties **u** act as a readout state.

Battaglia et al. [84] extended this initial message passing work to learn edge representations, in addition to nodal and global properties. Their work created `Graph Nets`, an open source GNN library available through `Github`. `Graph Nets` allow the connection of all 3 graph features, as shown in Figure 2.10, and the information sharing between them exemplifies message passing.

A key component of message passing is the need to *aggregate* information between different features. Nodes and edges contain different embedded information. Falling back on the molecular dynamics example this is the difference between the molecule velocity and the inter-molecule potential. To transfer information between these two, or the global property, a defined function is necessary in their conversion. Aggregation is the gathering of the information from selected neighboring features in a permutation invariant way, such as summation, minimum and maximum operations.

An ongoing challenge is aggregation between nodes not connected, and far away on a graph [99, 100]. Message passing allows the information of a node to be propagated through the network at each GNN layer. That is, if the GNN aggregates information at each node from neighboring nodes, this will diffuse k node steps away after k GNN layers. To resolve this issue, Sanchez-Lengeling et al. [96] acknowledges the use of *virtual edges*, which is the addition of edges to connect every single node on the graph. However, this method is known to be computationally unsuitable for large graphs and is considered an open-research area [96].

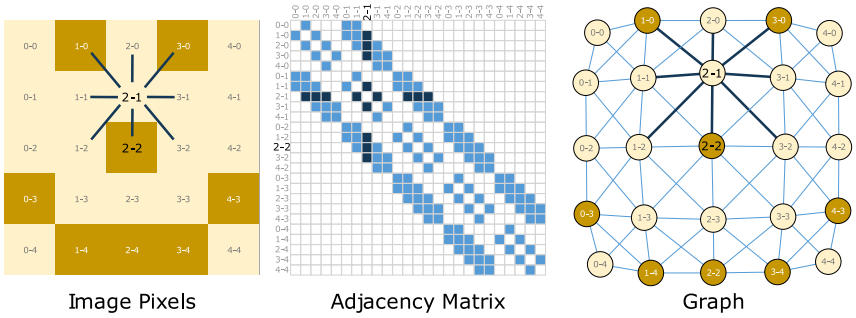
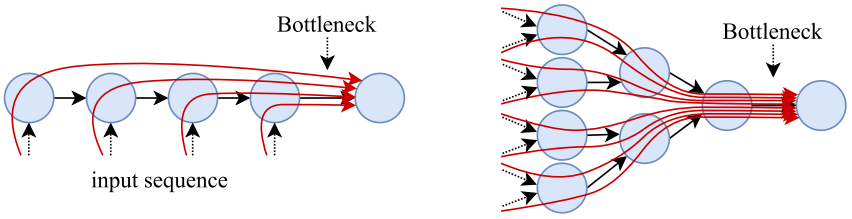


Figure 2.11: Illustration of graph and adjacency matrix correlation [96]



(a) The bottleneck of RNN seq2seq models

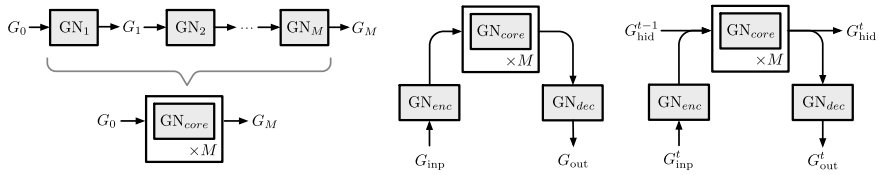
(b) The bottleneck of graph neural networks

Figure 2.12: RNN & GNN Bottleneck [98]

Neighbor sampling is the method by which surrounding nodal information is chosen to be aggregated in concentrating the information on a single local node. Figure 2.11 is an example sample method of choosing the nearest neighbor `Image Pixels`. The adjacency matrix in Figure 2.11 is a matrix of binary expressions detailing which nodes are and are not connected [96]. Neighbor sampling in graphs is identified as a current open research question [96], with a variety of techniques currently identified [101]. This is particularly advantageous when working with large graphs, and the number of neighbors to consider can be computationally expensive [102].

A bottleneck of GNNs is introduced by Alon & Yahav [98]. During the aggregation stages, both neighboring nodes and/or long distance nodes may send messages, resulting in their summation on a local node. The conglomeration results in a large amount of information trying to be expressed by the state of a single node. The inability of a single node to express such a breadth of information is referred by Alon & Yahav [98] as *over-squashing*. Figure 2.12 gives a visual of this phenomena.

Alon & Yahav [98] additionally acknowledge *under-reaching*, in which the chosen distance of neighboring nodes, the size of the minimum image, is not able to capture the physical essence of the system at hand. Thus, there exists a trade-off between the distance of neighboring nodes which are aggregated to find a sweet-spot between *under-reaching* and *over-squashing*. This *over-squashing* phenomena is not seen in all work,



(a) Composition of GN blocks (b) Encode-process-decode (c) Recurrent GN architecture

Figure 2.13: Diagram example of Encoder-Processor-Decoder Constructions [84]

Table 2.4: Overview of Encoder-Processor-Decoder Strategies

Model	`Encoder`	`Processor`	`Decoder`
GNS [85]	Particles are represented as nodes, with every node within a radius R connected and encoded as into latent space via MLPs	Stack of M GN Blocks, all in series, which predicts the future state $G^M = \texttt{Processor}(G^0)$	A MLP takes G^M as input and outputs acceleration, followed by Euler integration for velocity and position
C-GNS [90]	Similar to GNS	Similar to GNS, with added output of a scalar c of the constraint landscape, which the minimum of determines the implicit update	The optimized update to minimize c is decoded similar to GNS
Mesh-Based [103]	Domain relevant mesh information is encoded into both Eulerian and Lagrangian frames	Similar to GNS, except both Eulerian and Lagrangian edges are taken as input	The predicted state is expressed as output features and Euler integration yields desired quantities

and instead only *under-reaching* [85].

Remark 2 *The method by which the neighborhood nodes are chosen for aggregation appears to greatly affect the GNN performance. Node selection is analogous to importance sampling in Monte Carlo methods. This is an open range of study in literature, and explicitly acknowledged as such by Sanchez-Lengeling et al. [96].*

The Encoder-Processor-Decoder process, conceptualized in Figure D.5, is a common GNN architecture used when acting as surrogate models for physical simulations [85, 90, 103]. The nodes of the graph store embedded information, exemplified in Figure 2.11 where the image pixels may be encoded as RGB values per pixel. The Encoder phase maps the physical phenomena of the simulation to an embedded latent space with nodes and edges as representatives. The Processor uses the encoded information as inputs for the GNN model, which predicts a future state. The use of this latent space is intended to alleviate the *over-squashing* phenomena. The method and underlying architecture used here varies by model, with a few prominent ones documented in Table 2.4. A common use among the models described are multi-layer perceptrons (MLPs) acting as the encoder and decoder, while layers of GN blocks act as the processor.

There are two ways in which physical updates are made, explicit [81, 85] and implicit

[89, 90]. Kolter at al. [104] describes the update differences between explicit and implicit formulations well. In short, an explicit update determines the output based on a defined function of the input. In contrast, an implicit update determines the output based on a definition that the input and output must meet together. Thus, there is not an explicit determination of the update, but instead a quest for a condition to be met.

GNNs have shown the benefit of maintaining the data structure of particle based fluid simulators, however at the cost of unfeasible scaling [78]. Current research is working to tackle this domain and scale GNNs to larger node counts [99, 105–107]. Noteworthy, is the creation of the OGB Large-Scale Challenge as a global competition to develop Graph Network models which perform adequately on large scale datasets [107].

Such a competition requires an evaluation metric to quantify models in ranking. This challenge uses the mean reciprocal rank (MRR) and mean absolute error (MAE). This single scalar value quantifies the performance of these large scale GNN models on standardized datasets. A similar trend is seen in the use of GNNs to learn complex physics simulations. Table 2.5 shows a range of single scalar error metrics which these models use to quantify performance. Many other error metrics exist [108].

Table 2.5: Current models in literature and their performance metric

Model	Error metric
GNS [85]	MSE
C-GNS [90]	MSE
MeshGraphNets [103]	root MSE
EGNNs [109]	MSE
DPI-Nets [82]	MSE
MultiScale MeshGraphNets [106]	MSE

This section on recent GNN developments shows the current practices seen in literature. The graph structure can be adapted to complex physical simulations, molecule classification and large scale graphs. Message passing appears to improve model performance and exact implementation methods is an active area of research. The bottleneck of propagating high volumes of information in a graph is introduced. This empirical finding is alleviated using an encoder, processor, decoder structure. Last, the performance metric to evaluate these models are introduced.

KNOWLEDGE GAP

GNN architectures leverage the inductive bias of permutation invariance. Current works in GNN literature apply learning complex physical systems to show the expansive physical ranges captured via the ML model, including sand collisions, fluid simulations and solid mechanics models [77, 85, 90, 103]. These models evaluate their performance on the metric of single scalar values, as shown in Table 2.5. By definition the MSE value quantifies how well the tuned function fits the data it is provided. These do not inform the user if the underlying physics is met. The current numerical methods used in snow simulations, shown in Table 2.2, are all evaluated in their ability to meet physical phe-

nomena. The ability of GNN models to meet these same metrics is unknown. This yields the 1^{st} research question:

> Are GNNs trained to learn physics simulations able to capture the underlying physics?

Another limitation of this error metric is that it does not give physical information based on its magnitude. This makes it impossible to judge if changes at the input to the GNN yield any significant changes in the output. This yields the 2^{nd} research question:

> Are GNNs in this application robust to their inputs?

2.3. CONCLUSION

The snow science community has seen significant advances in the last decade to reduce the computational time to model the granular material. The use of MPM over DEM facilitates this computation cost reduction. However these methods continue to rely on the conventional use of local information to make local updates. In parallel, the insurgence of scientific ML shows that complex physics simulations may be learned via the availability of sufficient data to train models [77, 81]. GNNs utilize global information in making local updates. This makes the ML model an ideal replacement to extend on MPM's locality limitation. The GNN model's natural representation of particle domains, the architecture's inductive bias and its recent promise to capture complex physical setups place it in the center light to learn complex snow simulations. However, the lack of understood physics captured by this new method makes its implementation in the snow science community uncertain.

3

CONSISTENCY, STABILITY AND CONVERGENCE

This chapter assumes the reader is familiar with the GNS model[1] of Sanchez-Gonzalez et al. [85]. For background on the model, the reader is encouraged to read Appendix D for a streamlined overview. For a detailed description, see the original publication.

INTRODUCTION

THIS work is inspired from two landmark papers to reimagine how GNN models in this application are evaluated. One, the numerical properties of consistency, stability and convergence are vital components any numerical method should meet [110]. See Appendix C for the definitions of these 3 terms used in this work. Two, through implementation of curated benchmark examples, these properties are investigated [111].

The research questions stated at the end of Chapter 2 are addressed by redefining the way which these models are evaluated. This is done through two keys steps. One, model robustness is investigated by creating numerical studies which address stability and convergence. Two, the patch test in the finite elements community provides a way to define the error based on physical laws which should be met.

[1] For clarity, GNS is a specific GNN implementation created by Sanchez-Gonzalez et al. [85].

Table 3.1: MPM and GNN method equivalents

	MPM	**GNN**
Continuum discretization	Material points	Nodes
Global domain structure	Eulerian mesh	Graph connectivity
Local domain structure	Shape function	Message passing architecture

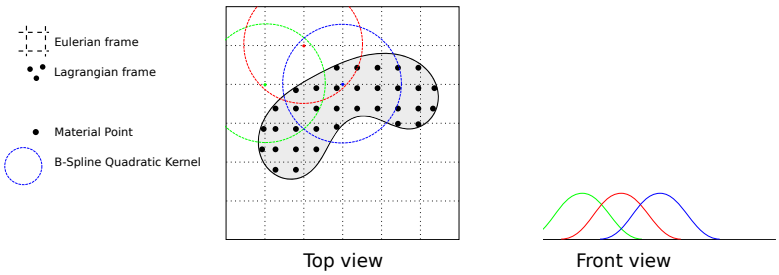


Figure 3.1: Visualization of the MPM continuum domain discretization, and the overlap of kernel nodes

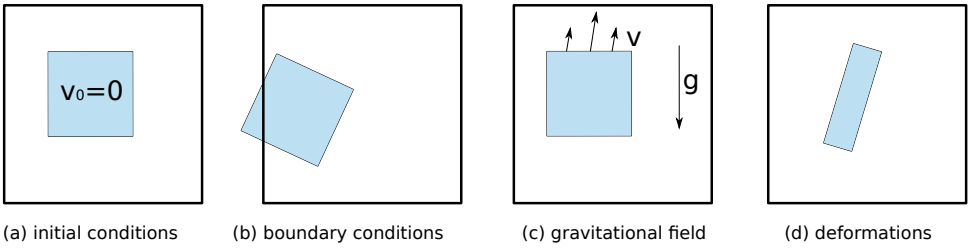


Figure 3.2: Example basic physics of interest

Section 2.2 illustrates GNNs have promising capabilities to learn conventional numerical solutions to PDEs. This places them in the center light in the exploration of ML scientific computing. The popular MPM model in the snow science community [7, 8] requires the continuum body to be discretized with Lagrangian points, a background Eulerian mesh and shape functions (kernels) defined *a priori* at the mesh nodes (Figure 3.1). In contrast, a state of the art open source GNN model [85] requires the position data for an entire trajectory to be available for model training and an optimal GNN connectivity and message passing architecture defined *a priori*. A comparison of these two methods' basic building blocks is shown in Table 3.1.

Chapter 2 shows a common practice in current ML physical simulation literature is to quantify model performance over a single scalar MSE. However, by definition this quantifies how well the trained ML model fits the provided data. This single value does not capture if the underlying physics of interest in a problem is met. Figure 3.2 illustrates a range of physical phenomena that are of scientific interest. However, whether these are met can not be affirmed by the MSE value alone. An example is now given.

Figure 3.3 showcases visually appealing results, similar to what is done in the GNS implementation [85] and other prominent models [81–83, 89, 90, 103]. These visual results are obtained via a dataset created by the MLS-MPM code of Hu et al. [66], and the demo model of their GitHub page. The GNS model was trained on the dataset. An example of inference is shown in Figure 3.3. See Appendix F for an explanation of the example setup and the GNS model implementation.

The error metrics used in the GNS implementation are shown in Table 3.2. These metrics are evaluated for N particles, over k time steps, j dimensions, their position $\mathbf{x}$, normalized acceleration[2] $\hat{\ddot{\mathbf{x}}}_i^{t_k}$, GNN acceleration prediction $d_\theta(\mathbf{x}_i^{t_k})$ and position prediction $\mathbf{x}_i'$ after twice pseudo Euler integration from acceleration[3]. The Loss (L_2 Norm Acceleration) is used to train the model. These error metrics are explained more in depth in Section 3.2 and Chapter 4.

Table 3.2: Error metrics

Metric	**Evaluation**
Loss (L_2 Norm Acceleration)	$\frac{1}{K}\frac{1}{N}\|\|d_\theta(\mathbf{x}_i^{t_k}) - \hat{\ddot{\mathbf{x}}}_i^{t_k}\|\|^2$
Acceleration MSE	$\frac{1}{j}\frac{1}{K}\frac{1}{N}\sum_{k=1}^{K}\sum_{i=1}^{N}(d_\theta(\mathbf{x}_i^{t_k}) - \hat{\ddot{\mathbf{x}}}_i^{t_k})^2$
One step position MSE	$\frac{1}{j}\frac{1}{K}\frac{1}{N}\sum_{k=1}^{K}\sum_{i=1}^{N}(\hat{\mathbf{x}}_i^k - \mathbf{x}_i^k)^2$

The final acceleration MSE is computed on per-particle normalized acceleration, and results shown in Table 3.3. This highlights that both visual results (Figure 3.3) and numerical results (Table 3.3) can be given without any understanding whether the physics of interest (Figure 3.2) is met. This thesis places a spotlight on this paradox.

Fortunately, a solution exists which is inspired from the finite element community. The patch-test was introduced in the '60s to the finite element community as a means to verify if a finite element design accurately captures the physical problem for which it is applied [112, 113]. The test examines that both physical quantities of interest at nodes and points internal to elements meet continuity requirements. A concrete example is shown in Figure 3.4, where a plate in bending with constant curvature K applied at the external nodes (red) yields the same curvature at the internal nodes (blue). A newly defined element passes the consistency condition of the patch-test if it approximates the uniform curvature within an acceptable tolerance at all nodes (red & blue).

[2] Acceleration is normalized to unit variance and zero mean using statistics computed over the entire training dataset.

[3] Pseudo refers to an integration being performed assuming Δt is 1, regardless of what the actual time step is.

Table 3.3: Snow blocks implementation loss values

Dataset	**Training steps**	**Acceleration MSE**
snow blocks v1	10^5	0.0089343
snow blocks v1	10^6	0.0057879
snow blocks v2	10^5	0.0087074
snow blocks v2	10^6	0.0058148

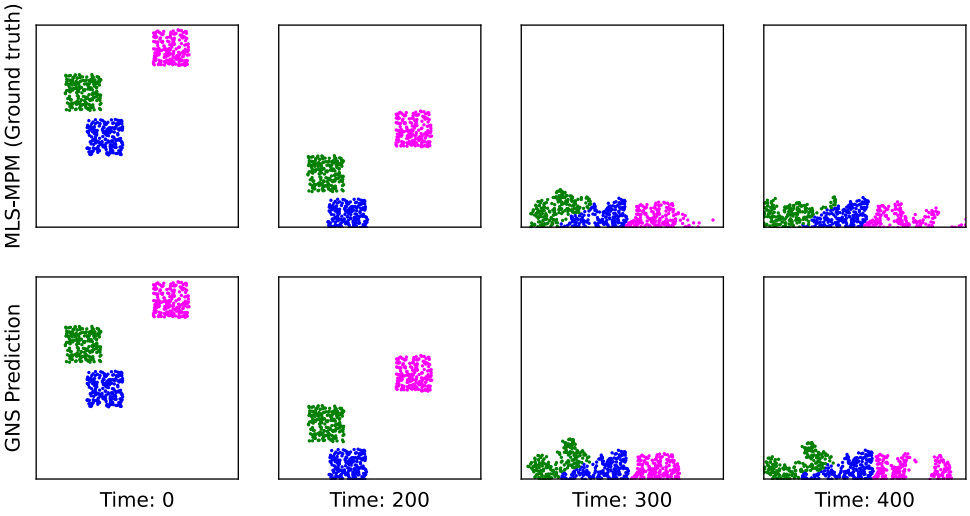


Figure 3.3: Snow blocks visual implementation

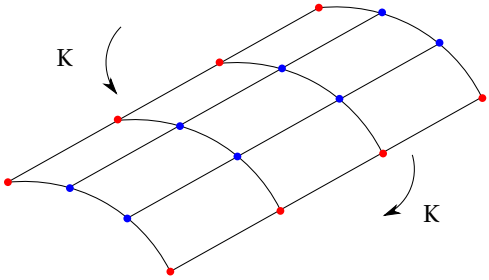


Figure 3.4: Example patch-test for a uniform plate in bending

Thus, the patch-test provides the key to traverse from evaluating this ML model against fitting data, and instead against an external error metric which captures physical underlying principles. The true power of this is realized when taking a next step, and using this external error metric to evaluate this ML model against the core properties a numerical method must meet: consistency, stability and convergence.

A similar evaluation concept is used in the meshfree field [114–116]. The basic premise uses a simple problem where assumptions allow an analytical solution. An example is a beam in bending, the numerical solution is compared with the exact assumed analytical solution [117]. This does limit example problems to those which meet the simplification requirements of the analytical solution. The comparison is a metric to quantify the quality of the meshfree scheme used in the proposed application.

This requires that simplified problems are defined to be setup in MPM. This is resolved via the original benchmark functions to evaluate the model's performance [13, 17]. These

Table 3.4: Summary of current benchmark tests

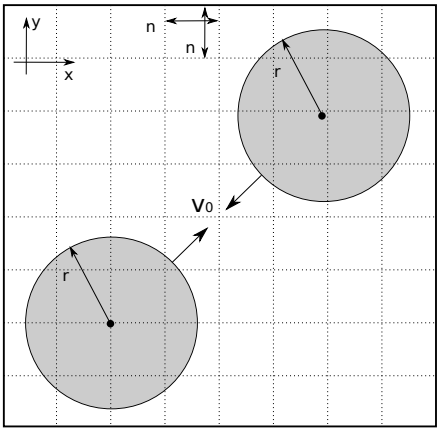
Benchmark	**Property investigated**	**Boundary condition**	**Plasticity**
Vibrating elastic cylinder [13, 118]	System kinetic energy & strain energy	None	No
Elastic bodies [13]	System kinetic energy, strain energy & momentum	None	No
Inelastic bodies [13]	System kinetic energy, strain energy & momentum	None	Yes
Hollow cylinder [17]	Visual	Neumann	No
Rolling disk on incline plan [70]	Center of mass	Neumann	No
Bar in tension [119]	Stress at Eulerian node	Neumann & Dirichlet	No
Plastic bar in tension [119]	Stress at Eulerian node	Neumann & Dirichlet	Yes
Self weight compression [120]	Stress at material point	Neumann & Dirichlet	Yes

are basic physical tests, such as rotating an object, or sending two elastic bodies into a collision. With the development of the method and its variants came more benchmark functions to showcase the capabilities and limitations of the method. These are shown in Table 3.4.

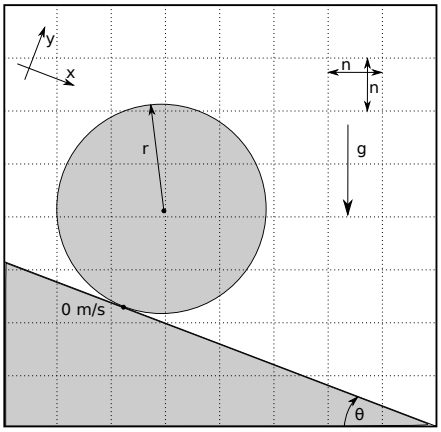
The selection of benchmark tests is to address the knowledge gap of underlying physics currently not captured by GNN models in literature, as discussed in the end of Chapter 2. This is done by choosing the physical principles which are expected to be a challenge in the ground truth data. That is, for the ultimate goal of a GNN model to replace the MPM method, it ought to be tested against the current limitations of MPM. Energy dissipation consistent with hybrid Eulerian-Lagrangian methods and difficulties imposing boundary conditions are the two physical phenomena chosen. Their motivation is now given.

Literature shows MPM and its variants have a history of energy dissipation. The original proposal of MPM [13] is rooted in PIC [14], an early hybrid Eulerian-Lagrangian approach. A recognition of the numerical dissipation effects in PIC is resolved with the proposed Fluid Implicit Particle (FLIP) method, which focuses on conserving energy and momentum [121]. Upon proposal of MPM [13], Sulsky et al. include a benchmark test to quantify the numerical dissipation of the new method (Figure 3.5a). In later years, this numerical dissipation of MPM and other hybrid Eulerian-Lagrangian methods is explored by Jiang et al. [122], leading to their own resolution. Thus, this long standing struggle of hybrid Eulerian-Lagrangian approaches to conserve energy makes this benchmark test an ideal standard.

The rolling disk on an inclined plane (Figure 3.5b) was created by Bardenhagen et al. [70] to test the paper's friction model presented for inter-material point contact. This requires a zero velocity boundary condition is applied at the contact of the disk with the



(a) Elastic bodies benchmark test



(b) Rolling disk benchmark visualization

Figure 3.5: Proposed benchmark tests

plane. Meshfree methods commonly struggle to impose boundary conditions. Examples are truncation error in SPH [39], the resolution of this with reproducing kernel methods [123], or the element-free Galerkin methods using Lagrange multipliers to enforce boundary conditions [117]. This historical phenomena of meshfree methods makes this test, where a boundary condition on the nodes is implemented, attractive. However, a drawback is this test does not directly quantify compliance with the *stick* boundary condition. Instead it only quantifies the disk's position against the analytical solution. This test is ideal by requiring only positional data to evaluate, which is the sole quantity the current open-source GNS model predicts [85].

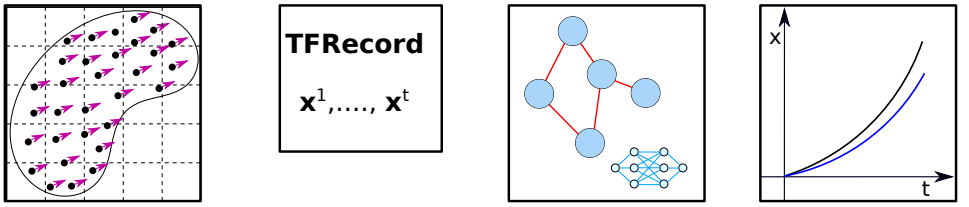
MODELS USED

The open source `c++` code formulation of MLS-MPM proposed by Hu et al. [66][4] implements the constitutive model for snow proposed in the paper of Stomakin et al. [7]. This damage based model is detailed in Appendix B, as well the MLS-MPM implementation is described in Algorithm B.1. The model is used in this work for implementing both the benchmark tests, and creating the datasets for the GNS model. It is the *ground truth* model.

For implementing a GNN, the state of the art model written in `python` by Sanchez-Gonzalez et al. [85] is used. The primary reason for its selection is the open code publicly available. Secondary, the model is shown in its publication to *learn* the dynamics of sand, a granular material as snow.

Both the GNS model and MLS-MPM sample code used here are minimally documented by the respective authors. To make this thesis possible, fundamental understanding of both code bases is accomplished via extensive documentation. The documented code

[4] Please see Appendix G for an analysis on the discretization assumptions made by this method.



(a) MPM data generation (b) data storage (c) GNN implementation (d) benchmark evaluation

Figure 3.6: Project computational pipeline

is available on `GitHub` through the BessaResearchGroup.[5]

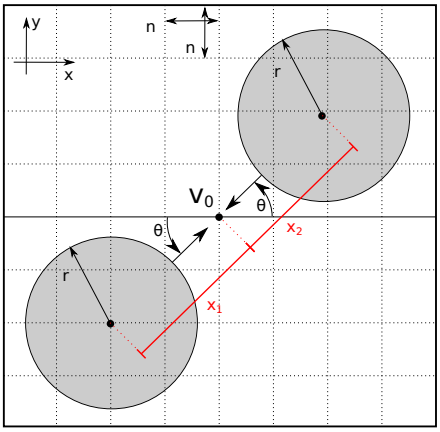
Parameters are subdivided into two regimes, GNS model hyperparameters and dataset definition parameters. Due to the custom data creation component of this work via benchmark tests, the way these datasets are defined is an independent variable of the author. The resulting computational pipeline is: using the MLS-MPM method to implement benchmark tests, creating data from these benchmark simulations, organizing into datasets, training the GNS model, predicting trajectories and finally processing these predictions into the physical quantities relevant to each benchmark. This project flow is shown in Figure 3.6.

Dataset storage uses `TensorFlow`'s `TFRecord` binary data format. This is for two key reasons: (*i*) The data may be directly read into the GNS model which is built on `TensorFlow` 1, (*ii*) The binary format provides a compact means to transfer data.

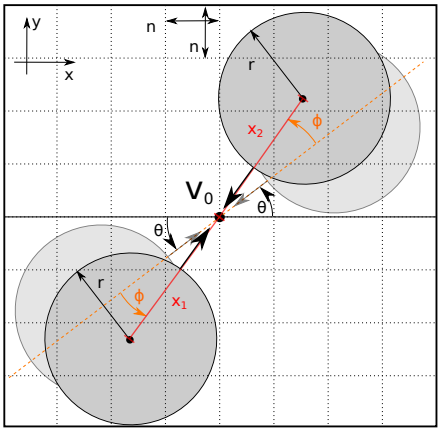
For repeatability in results, a seed is set for `TensorFlow`, which the GNS model is built on. This fixes the random initialization of the following three items: (*i*) the weights and biases of the neural network, (*ii*) dataset shuffling during training, (*iii*) the random walk noise sequence added to the training data.

Implementing the GNS model is a computationally expensive process, with setups requiring training times from hours to days. The resources available through the high performance computer (HPC) at Delft University of Technology's ICT Department made this work possible.

[5] The repository is currently held private. Access may be granted upon request.



(a) Elastic bodies v1 setup



(b) Elastic bodies v2 setup

Figure 3.7: Visualization of the elastic bodies benchmark test

3.1. ELASTIC BODIES BENCHMARK TEST

SETTING

Figure 3.5a shows the physical setup for the elastic bodies benchmark created by Sulsky et al. [13]. This test contains two elastic bodies, each with the initial condition of the identical velocity magnitude, only opposite in direction. The two bodies impact and then rebound, moving away to the opposite corners. During collision, all kinetic energy is transferred to strain energy, and then returned to kinetic energy as the bodies move away. This setup isolates energy quantities. A *perfect score* on the test is the sum of kinetic and strain energy remaining constant during the entire simulation. This evaluates that no dissipative mechanisms are found, as no dissipative mechanisms are intentionally implemented.

MLS-MPM IMPLEMENTATION

Figure 3.7a shows the setup described by Sulsky et al. [13] to implement the benchmark test. Table 3.5 lists the definitions and values given for the variables in Figure 3.7. Note units are intentionally not given, as they are not provided in the benchmark publication. For this reason, terms of unit length, unit time, unit energy etc. are used when describing this benchmark. The benchmark is implemented as closely as described by the publication, with the only deviations described here.

The total particle count nor particle count per cell is given. Thus, a particle spacing is assumed at 0.020 unit length yielding a total particle count of 552. This value is chosen as it produces converged energy results. A coarser resolution yields additional energetic loss. An exact radius for the bodies is not given by Sulsky et al., instead 0.19 unit length is assumed from the figures presented in the publication.

Sulsky et al. [13] do provide a density value of 1000 unit density. However, without a known radius makes this intrinsic quantity's use uncertain. A total mass of 256 unit mass

Table 3.5: Elastic bodies benchmark parameters

Parameter	Value
Total time	1.0
Time step	1e-3
Young's modulus E	1000
Poisson's ratio ν	0.30
Total mass	256
Radius r	0.19
Total material points	552
n	0.05
v_0	$0.1\sqrt{2}$
θ	$\pi/4$

is chosen, which correlates with the total system kinetic energy graphically reported in the reported benchmark.

As the base MLS-MPM code of Hu et al. [66] does not make use of the particle strain energy Ψ during computation, a means to compute it is required. The MLS-MPM code of Hu et al. [66] uses the gradient of the fixed corotated energy density function Ψ [124] to compute force. This is an isotropic hyperelasticty model developed for large deformations [124]. The model is shown in Equation 3.1. See Appendix B for a description of the variables and finite strain decomposition. To follow continuity with the force calculation, this model is used to compute the strain energy.

$$\Psi(\mathbf{F}_E, \mathbf{F}_P) = \mu(\mathbf{F}_P)||\mathbf{F}_E - \mathbf{R}_E||_F^2 + \frac{1}{2}\lambda(\mathbf{F}_P)\,(J_E - 1)^2 \tag{3.1}$$

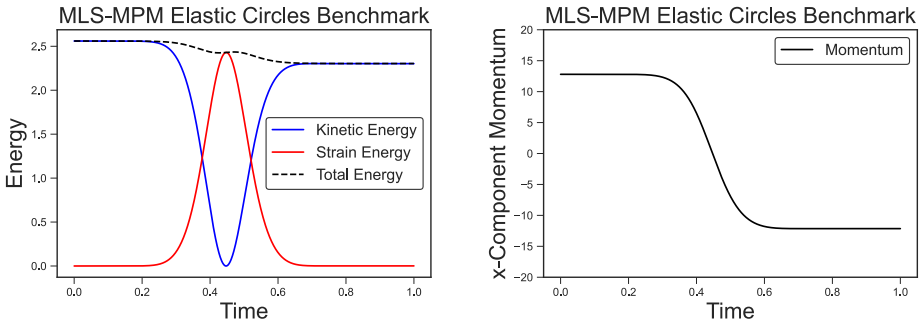
Figure 3.8 shows the results of the benchmark test. Following the original benchmark implementation, the energy and momentum are plotted. Figure 3.8a shows the kinetic, strain and total energy of the system with time. There is a clear drop in the total system energy. Th Figure 3.8b shows the x-component of momentum for the lower left circle with time.

There is a distinct difference in the total collision time between this implementation with MLS-MPM and regular MPM as done by Sulsky et al. [13]. An investigation of this variance is given in Appendix J.

DESIGN OF EXPERIMENTS

The goal of this chapter is to evaluate the GNS model performance when compared to the MLS-MPM method. This is done by creating a design of experiment (DoE) which evaluates the stability property of the numerical method in this benchmark.

Appendix C lists the definitions for the consistency, stability and convergence. However, these definitions assume an approximation function F_n is known. In the case of GNNs,



(a) System Energy

(b) Lower left circle x-component of momentum

Figure 3.8: MLS-MPM Elastic Bodies Benchmark Implementation

the neural network approximation function is fully defined from training. It is not known *a priori*, only its architecture is known. A key component of training is defining the optimization problem to tune the network parameters on some loss, as defined in Equation A.6.

A knowledge gap identified in Chapter 2 is the error metrics do not directly express physical information of the problem. The same logic applies to this scalar loss value. It should be affirmed the optimization problem is consistent with the physical quantities evaluated in this benchmark. In other words, as the training steps increase in the problem, there ought to be an improvement in the benchmark performance.

Stability is defined in Equation C.4. Stability is qualitatively described as small perturbations at the input level yields small variants in the output. Here, it is empirically evaluated by imposing small perturbations to the input data, and observing the resulting outputs. In this case, a simple small input perturbation is changing the particle label associated with the given dataset. As discussed in Appendix D, this is an arbitrary classifying label given to a particle which is meant to represent the type of material. As this label choice is arbitrary, the selection ought to have no effects on the output.

The trained neural network is defined based on the data it is trained to. This means the approximation function is dependent on the data. Varying the underlying data thus alters the definition of the approximation function. It is thus of interest to see if the means to create the approximation function can handle perturbations in the domain data.

GNS IMPLEMENTATION

For the elastic bodies benchmark, two datasets are designed. Figure 3.7 shows the two dataset variants built, v1 and v2. One dataset is the standard benchmark implementation. The second version is the original benchmark, with the bodies rotated at an arbitrary angle ϕ. Table 3.6 lists the dataset parameters held constant between both datasets, and the particular parameters relating to each individual dataset. The variables of x_1, x_2, $\mathbf{v}_0$ and ϕ are all generated from a random uniform distribution according to the bounds

Table 3.6: Elastic bodies GNS model implementation

GNS Hyperparameters		Dataset Parameters	
GNN Architecture	`Interaction Network`	Total time	1.0
Loss function	L_2 Norm Acceleration	Dataset time step	0.001
Learning optimizer	Adam	Train/Test	300/20
Learning rate	exponential decay from 10^{-4} to 10^{-6}	Trajectory length	1000
Training steps	10^5	**Dataset v1**	
Message passing steps	10	x_1 & x_2	$[0.290\sqrt{2}, 0.310\sqrt{2}]$
NN Architecture	2 hidden layers 128 layer neurons	$\mathbf{v}_0$	$\pm v_0(\cos\theta, \sin\theta)$
Noise std	6.7e-4	**Dataset v2**	
Input sequence length h	6	x_1 & x_2	[0.270, 0.300]
Seed	33	ϕ	[-0.1π, 0.1π]
Connectivity radius	0.0565	$\mathbf{v}_0$	$\pm v_0(\cos(\theta+\phi), \sin(\theta+\phi))$

Table 3.7: DoE Set #1 - Elastic Bodies from Setup v1

Experiment	Training steps	Particle label	Acceleration MSE ($\cdot 10^{-3}$)	One step Position MSE ($\cdot 10^{-10}$)
#1.1	10^5	5	0.41187	1.0468
#1.2	10^6	5	0.17930	0.71993
#1.3	10^5	7	0.40105	0.89963
#1.4	10^6	7	0.17973	0.73982
#1.5	10^5	5 & 7	0.34078	0.62333
#1.6	10^6	5 & 7	**0.17448**	**0.44220**

set in Table 3.6.

The particle label is administered as a uniform label between both bodies for all experiments, with the exception of #2.3 and #2.6. In these two experiments, one body is unanimously labeled a 5, and the other body is labeled a 7. The choice of these particle labels are arbitrary. They are only a means to provide a minute perturbation at the input level.

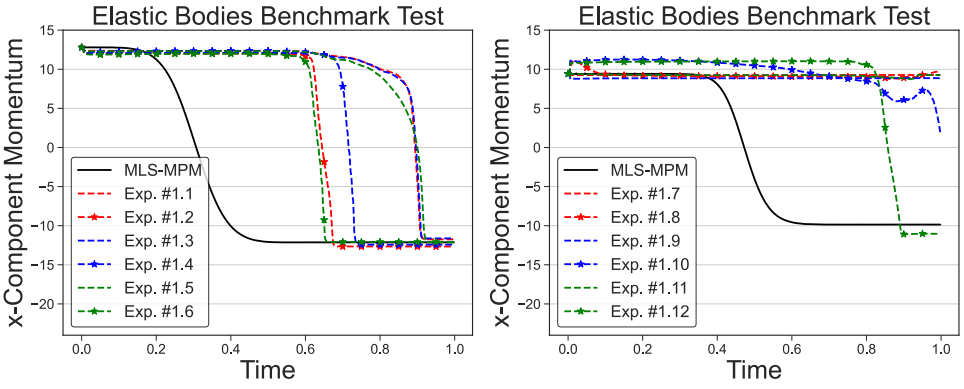
The GNS model used in these experiments predicts solely normalized acceleration. The model was extended to predict directly normalized acceleration and strain energy. This extension yielded grossly inaccurate results at 10^6 training steps. For this reason the standard GNS model which predicts only normalized acceleration is used here.

RESULTS

Table 3.7 and Table 3.8 shows the resultant GNS model error values for the elastic bodies benchmark. These error metrics are defined in Table 3.2. The minimum of each error column is highlighted in **bold**. There is a distinct difference in the magnitude of the loss values between dataset v1 and dataset v2. There is a variance in the reported loss value

Table 3.8: DoE Set #1 - Rotated Elastic Bodies from Setup v2

Experiment	Training steps	Particle label	Acceleration MSE ($\cdot 10^{-3}$)	One step Position MSE ($\cdot 10^{-10}$)
#1.7	10^5	5	1.7356	6.5566
#1.8	10^6	5	0.81389	5.1727
#1.9	10^5	7	1.7583	6.8231
#1.10	10^6	7	0.82089	4.1139
#1.11	10^5	5 & 7	1.6112	5.5408
#1.12	10^6	5 & 7	**0.70821**	**3.0655**



(a) Dataset v1

(b) Dataset v2

Figure 3.9: GNS model predictions for the lower left body momentum x-component

based on the particle type given.

Figure 3.9a shows the trained GNS model predictions for the experiments described in Table 3.7. This is trained on dataset v1 for the original benchmark implementation shown in Figure 3.7a. Figure 3.9 shows the x-component of momentum for the lower left disk. To *pass* this test, predictions must follow the momentum in time defined by the MLS-MPM method.

There is a key recognition that across both plots in Figure 3.9, there is a failure of collision dynamics. If collision dynamics were met, the GNS prediction would match the MLS-MPM curve. Additionally, both plots show variance in the benchmark metric when the particle label is changed. Figure 3.9a does show a conservation of linear momentum for all implementations in the Figure. The linear momentum of the other body and kinetic energy plots are shown in Appendix H.

The training and inference times for the GNS model are listed in Table 3.9[6]. The MLS-

[6]These times are for 12 Intel Xeon CPU E5-2620 0 @ 2.00GHz.

Table 3.9: Training and inference times for the elastic bodies benchmark

Experiment	Training steps	Training time/ step (s)	Inference time/ step (s)
#1.1	10^5	1.469	229.4
#1.2	10^6	1.470	227.7
#1.3	10^5	1.470	227.6
#1.4	10^6	1.469	220.7
#1.5	10^5	1.462	221.4
#1.6	10^6	1.367	217.7
#1.7	10^5	1.469	220.7
#1.8	10^6	1.466	217.0
#1.9	10^5	1.462	221.4
#1.10	10^6	1.469	226.0
#1.11	10^5	1.367	217.7
#1.12	10^6	1.464	215.6

MPM implementation in c++ requires 2.9 seconds[7].

A visualization of Experiment #1.1 is shown in Figure 3.10. The MLS-MPM model shows the two bodies initially start at opposite corners, and move towards each other with equal velocity magnitudes. They then collide, and rebound without touching. The two bodies then move in the opposite direction. This is expected[8]. The GNS prediction shows the two bodies coming into contact, and then moving in unison to the bottom left corner. This is distinctly different than the ground truth dataset.

Figure 3.11 gives a visual of Experiment #1.8. In this case, the MLS-MPM method shows the two bodies approach and rebound in a similar fashion as Figure 3.10. However, a key difference here, is at an offset angle ϕ, the bodies rebound with rotational motion. Strikingly, the GNS model rotates both bodies prior to impact. Appendix I quantifies this difference in rotational motion between the MLS-MPM simulation and the GNS model.

DISCUSSION OF RESULTS

A discussion is now given on why no improvement is seen with an increase in training steps, and why the stability condition is not met. A lack of improvement with training is due to graph construction. The stability condition is failed due to the inadequate data provided during training. Momentum conservation is observed in the results, and is explained by addressing invariants in the domain.

There is no improvement found with an extension of training steps. The GNS model is the approximation function to the true underlying domain described by the benchmark formulation. At an increase of training steps from 10^5 to 10^6, Figure 3.9 shows the collision dynamics continue to fail. This is a clear indication that the chosen optimization

[7] This is on 1 Intel(R) Core CPU i7-6700HQ @ 2.60Ghz.
[8] Appendix J extensively explores and explains this collision in the ground truth model.

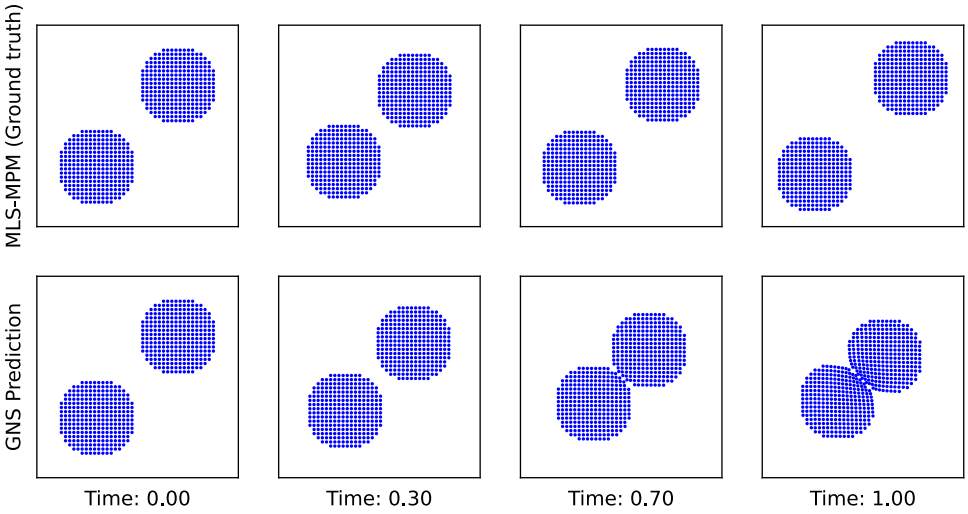


Figure 3.10: Visualization of experiment #1.1

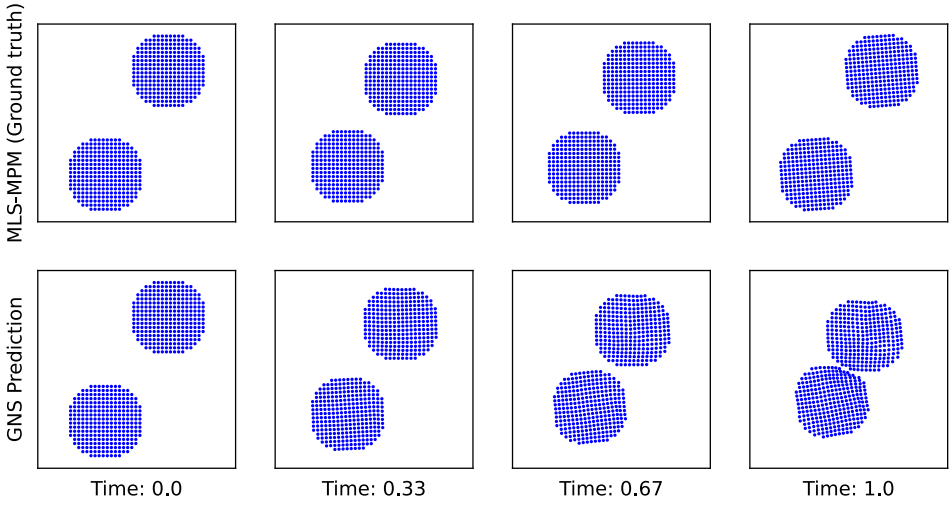
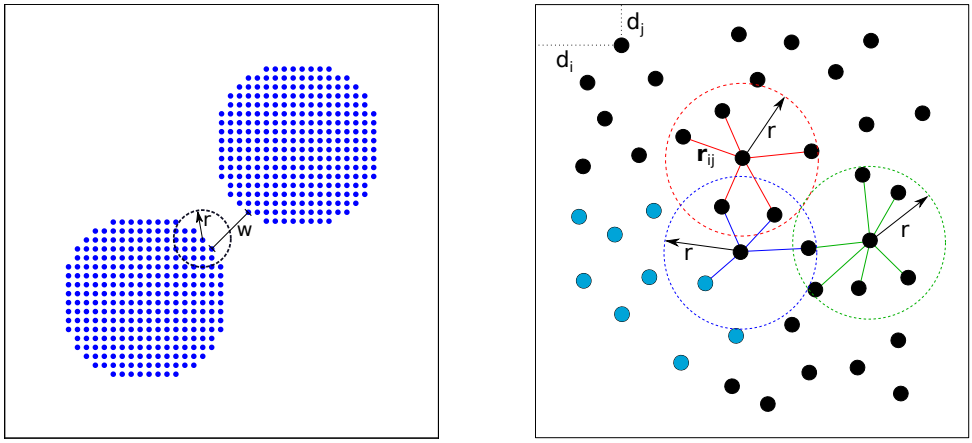


Figure 3.11: Visualization of experiment #1.8



(a) Collision visual

(b) Graph construction

Figure 3.12: Collision dynamics failure visualization

formulation is not consistent with the benchmark performance.

This inability arises from the construction of the graph. GNNs require graphs as inputs to the model. This means that one must express their physical domain, in this case snow, as a graph. Chapter 2 highlights that graphs can naturally represent the Lagrangian formulation of MPM. That is the nodes of the graph represent material points, and the graph edges represent how these material points interact.

Figure D.6 shows the graph construction process, and is described in Appendix D. Figure 3.12a shows a visual of the collision, with the MLS-MPM simulation on the top row, and the GNS model along the bottom row. It is found that the the two bodies do not come into contact during the MLS-MPM simulation. This characteristic of the ground truth simulation is extensively explained in Appendix J. Here, the minimum distance w which the two bodies come together during impact does not decrease below 0.107 unit length. Yet, the connectivity radius is set in Table 3.5 at 0.0565 unit length. Thus, the two bodies are never connected during the training phase. Edges are fundamental for GNNs to learn relations between objects. Without edges, the GNN model is not able to learn the relation between these two bodies. This yields a failure in the collision dynamics.

The original GNS publication emphasized the substantial effects the connectivity radius has on results. The effects of altering the connectivity radius are note explored here. The reason being, this chapter is not to evaluate the GNS model, but to showcase that GNN models can be evaluated on numerical first principles by reenvisioning the error metric. Two potential avenues to explore the error metric here are: (*i*) Extending the radius, which risks *over-squashing*, as discussed in 2.2, (*ii*) include hierarchical message passing between objects [87].

The GNS model is unstable in this benchmark implementation. Figure 3.9b shows significant variance in the benchmark result as the particle label is changed, especially at

10^6 training steps. The basic building block behind this entire approximation function is a neural network. Under sufficient training steps, it is anticipated this tunable function can express any underlying function expressed by the data [125]. In the case of GNNs, this requires the data is properly formulated into a graph which expresses the domain. It is shown the graph construction strategy does not represent the relations between the objects. The flaws in the graph construction described thus also hinder the stability of the model.

The linear momentum conservation for dataset v1 shown in Figure 3.9a is caused by the architecture of the GNN model, the physical root of this conservation, and the provided training data. Section 2.2 introduced the concept of inductive bias. This is the ability of a given ML architecture to exploit symmetries which exist in the domain [126].

The GNS model feature choices are spatially invariant, as shown in Table D.1. They are not dependent on the spatial location of the node.[9] Then, the neural network function is also not dependent on the spatial location of the node, as it has no information on the absolute position of the node. A governing equation of MLS-MPM is conservation of linear momentum. The ground truth data numerically showed this conservation in Figure 3.9a. Physically, linear momentum conservation is a direct result of invariance to spatial translation [127].

The by definition spatial invariance of the node neural network, expectation and existence of linear momentum conservation in the ground truth data, and physical first principles origin of linear momentum conservation all motivate this numerical finding.

By this same logic, the lack of linear momentum conservation in Figure 3.9b is explained. This benchmark problem is described above to include rotational motion after collision. The rotational dependencies seen in Figure 3.11, and quantified in Appendix I, are due to the input features chosen for the GNS model. The desire of this approximation function is that it learns the underlying physical principles, and not just overfit the data for this one benchmark. To facilitate this, features ought to be chosen which are independent of the coordinate axis location, or its rotation, as physical laws are independent to this modeling tool [109, 126].

Appendix D lists the main GNS node feature are velocity vectors. This is a feature choice dependent on the rotation of the coordinate axis. As a result, it is asked of the GNS model to learn that physical laws are independent of the coordinate rotation [87, 128]. In this case, it is shown at these training steps, the GNS model does not learn this independence.

Last, Table 3.9 shows the inference times for the GNS model in this application. The average inference time from the 12 trajectories is 221.9 seconds. This is a 7,650% increase over the c++ code.

This discussion section shows the graph construction process is the failure for both the optimization problem not being consistent with the benchmark performance and the

[9] This is only for internal nodes, where the clipped distance to the boundary is set to 1.

stability failure. The linear momentum conservation is a success of the GNS model to capture expected domain invariants.

3.2. ROLLING DISK BENCHMARK TEST

SETTING

The rolling disk benchmark test is implemented following the description of Bardenhagen et al. [70]. The test is a disk discretized via material points, and a *stick* zero velocity boundary condition is set at the contact surface. The test quantifies the disk center of mass x component with time. The key item of interest is the quadratic relationship of position with time under a uniform gravitational field. Equation 3.2 states the disk center of mass x as a function of its starting position x_0, gravitational magnitude g, time t and the angle of incline θ for a rigid disk assumption. The numerical methods used will be compared with this analytical solution.

$$x(t) = x_0 + \frac{1}{3} g t^2 \sin\theta \tag{3.2}$$

The benchmark authors keep the majority of the physical parameters constant. Some parameters are varied in different test implementations (see Table 3.10).

Table 3.10: Rolling disk benchmark test parameters

Kept Constant		**Variables**	
Disk radius [cm]	50	Time [s]	0.3 - 0.4
Shear modulus [MPa]	2.5	ratio n/r [R]:	0.50R - 0.125R
Bulk modulus [MPa]	10	Incline angle θ [rad]	π / 6 - π / 3
Density [kg/m^3]	3,000		
dt [s]	1e-6*		
Linear elastic	True		
Gravity [m/s^2]	9.8		
Material points per cell	9		

MLS-MPM IMPLEMENTATION

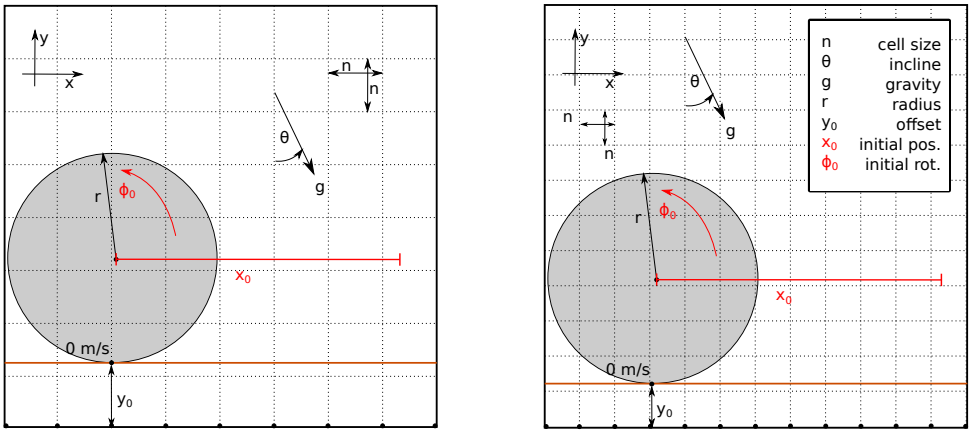
Figure 3.5b shows the incline plane and that the plane does not intersect will all Eulerian nodes. Implementing the zero velocity boundary condition in this reference frame yields a disk rolling along a step-wise roll. This is undesirable. Instead, the Cartesian coordinate system is rotated such that the plane aligns with the Eulerian grid. Figure 3.13 shows the rotated reference frame, and the inclined plane parallel to the Eulerian cell x-direction. Bardenhagen et al. [70] use this same strategy.

The zero velocity boundary condition is applied directly to nodes which align with the plane. This is shown in Figure 3.13 via the black dots along the bottom of the Eulerian grid.

Affine particle in cell methods are known to have contact stick issues when objects and boundaries interact, these issues being proportional to the width of the Eulerian cell [129]. Here, the zero velocity implementation on the bottom nodes results in the disk being fixed in place on the bottom surface. In other words, if the disk is directly placed on the plane (Figure 3.5b), then the disk remains fixed in position. This issue is resolved by augmenting the vertical distance the disk starts at, such that it rolls along the *stick* boundary condition. Figure 3.13a shows this as an orange line. This value is fixed proportional to the cell width for all MLS-MPM implementations at $\frac{5}{4}n$, in accordance with the effects of boundary difficulties being proportional to the Eulerian cell width [129].

The benchmark authors do not specify a means to discretize the disk via material points, in contrast to other benchmarks [13, 17]. A strategy of discretizing the disk is described in Appendix L. The motivation for the discretization method chosen is to mimic the uniform mass distribution defining a disk, and its shape.

Another difference is the extrinsic dimensions used. This formulation of MLS-MPM is written in a unity 1x1 domain. The benchmark authors propose a radius of 0.50m. In this setup, the radius creates a disk the size of the computational domain. For simplicity, instead of scaling all dimensions accordingly, the disk radius is halved.

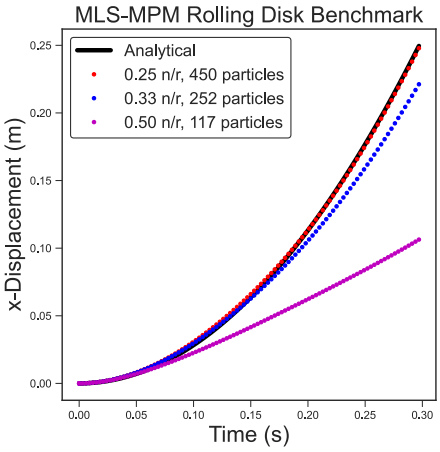


(a) 0.50 n/r ratio

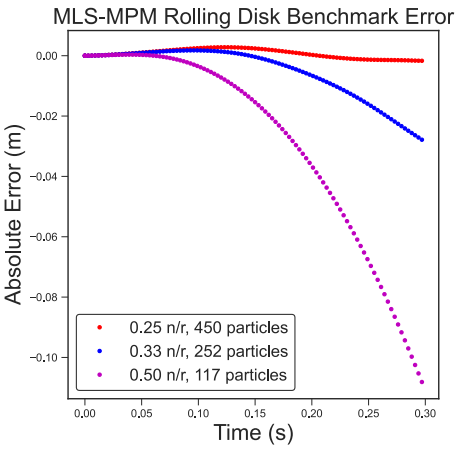
(b) 0.33 n/r ratio

Figure 3.13: Rolling disk MLS-MPM benchmark implementation ($y_0 = \frac{5}{4}n$)

The core ground truth metric used to evaluate the MLS-MPM model in this implementation is the rolling disk center of mass as a function of time, Equation 3.2. Figure 3.14 shows the results of the implementation. There is a clear convergence with increasing particle count, as documented by Bardenhagen et al. [70]. Particle count is directly varied by altering the number of Eulerian cells via the ratio n/r (Table 3.10). As r is constant at 25 cm, altering the ratio varies the total cell count. As the total material point count per cell is constant, this also increases the total material points.



(a) Rollout



(b) Error

Figure 3.14: MLS-MPM Rolling disk benchmark results

DESIGN OF EXPERIMENTS

For this benchmark test, the stability and convergence properties are investigated. Additionally, whether the optimization problem is consistent with the benchmark performance is evaluated. This is done by extending the training steps from 10^5 to 10^6. The magnitude of the total training steps are arbitrary, but only limited to computational time. The intention of this variable is to build an understanding of the approximation function's consistency. This is done by observing the change in the results as the training steps jump an order of magnitude.

Small perturbations to investigate stability are imposed by varying the seed of the GNS model. The DoE generator of `F3DASM` is used to generate a uniform random sequence of seed values. A stable method is expected to produce the same results as the seed is varied.

Last, a convergence study is completed by reducing the time step of the underlying dataset. As with conventional numerical methods, reducing the time step ought to yield a convergence on the true solution. The following range of time steps in seconds are implemented: 3e-05, 6e-05, 3e-04, 6e-04, 3e-03 and 6e-03.

GNS IMPLEMENTATION

The first step to implement the GNS model is creation of a dataset to train the model. To promote dataset variability, two values are initialized at random from a uniform distribution. For the rolling disk setup, the initial rotation ϕ_0 given to the disk and the horizontal distance x_0 it is placed are varied. The reason for these physical quantities being generated at random is the model ought to have no dependency on where the disk starts nor its initial rotation. Figure 3.13a labels these two random initialized values in red. The range of ϕ_0 values allows for a complete rotation. The value range of x_0 are chosen such

Table 3.11: Rolling disk GNS model implementation constants

GNS Hyperparameters		Dataset Parameters	
GNN Architecture	`Interaction Network`	Total time [s]	0.30
Loss function	L_2 Norm Acceleration	θ [rad]	π / 3
Learning optimizer	Adam	Disk radius r [cm]	25
Learning rate	exponential decay from 10^{-4} to 10^{-6}	Density [kg/m^3]	12,000
Training steps	$5{\cdot}10^4$, 10^5, 10^6	Mesh resolution [n/r]	0.33R
Message passing steps	10	Num. particles	252
NN Architecture	2 hidden layers 128 layer neurons	x_0	[0.3, 0.45]
Noise std σ_0	6.7e-4	ϕ_0	[0, 2π]
Input sequence length h	6	Train/test/valid sets	300/20/3
Connectivity radius [m]	0.07		
Particle type	None		

that the disk begins and ends within the computational frame during the simulation. The uniform distribution domain for ϕ_0 and x_0 is given in Table 3.11.

The cell to radius ratio of 0.33 n/r is selected to be implemented. Figure 3.14a shows a ratio of 0.25 n/r produces results nearest the analytical solution. However, the computation cost for training the GNS model at a larger particle count hinders the code development iteration process. For this reason, 0.33 n/r is chosen.

The lower boundary is implemented directly at the offset line required for MLS-MPM. Thus, for the GNS model, no offset is implemented between the disk and the boundary.

The benchmark metric in Figure 3.14a only shows the center of mass as a function of time. This assumes the shape of the body remains in check. Preliminary results showed the GNS model does not always maintain the original shape. Figure 3.15 gives an example of this distortion. This is undesirable. The geometry deformation is quantified via its mass moment of inertia about its own center of mass. If the relative mass moment of inertia between the MLS-MPM model $I_{x,y}^{\text{MPM}}(t)$ and the GNS model $I_{x,y}^{\text{GNS}}(t)$ exceeds a predefined tolerance ε at any time t, the benchmark fails. Equation 3.3 defines this mass moment of inertia $I_{x,y}$ in the x and y directions separately, as a function of the particle mass m, location of a particle p_i and the body center of mass m_c over N particles.

$$\frac{|I_{x,y}^{\text{MPM}}(t) - I_{x,y}^{\text{GNS}}(t)|}{I_{x,y}^{\text{MPM}}(t)} > \varepsilon \tag{3.3}$$

$$I_{x,y}(t) = \sum_{i}^{N} m * (p_i(t)_{x,y} - m_c(t)_{x,y})^2$$

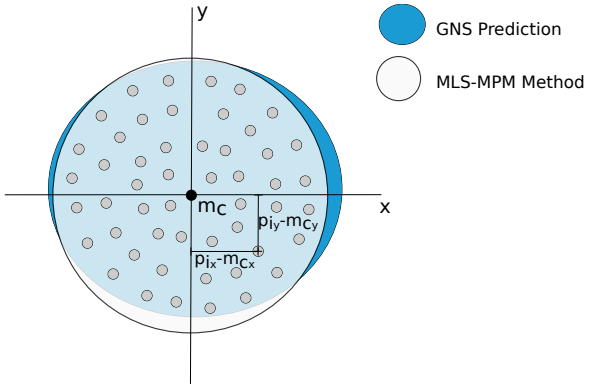


Figure 3.15: Example rolling disk distortion

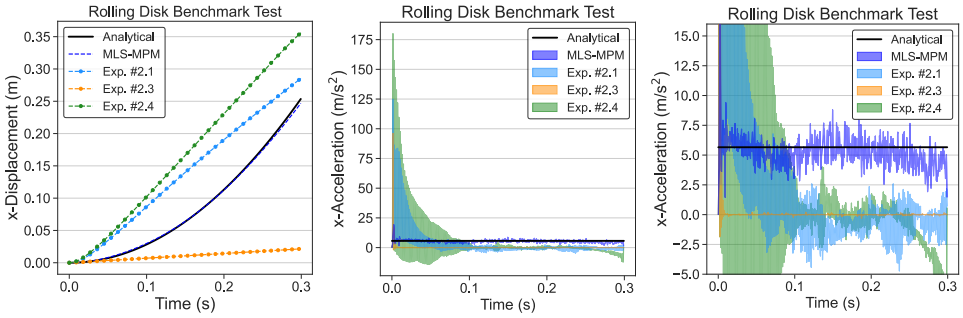
Table 3.12: DoE for Set #1 - Rolling Disk

Experiment	Training steps	Seed	Acceleration MSE ($\cdot 10^{-3}$)	One step Position MSE ($\cdot 10^{-10}$)	Shape tolerance $\varepsilon = 5\%$	$\sigma(\ddot{x}_{m_c})$ (m/s^2)
MLS-MPM	-	-	-	-	-	1.3839
#2.1		51	0.89080	1.6827	Pass	12.008
#2.2		92	1.1206	1.4856	Fail	-
#2.3	10^5	14	1.1663	1.8392	Pass	3.7673
#2.4		71	0.86577	1.9601	Pass	17.517
#2.5		60	0.78931	1.1205	Fail	-
#2.6		51	0.80020	1.3125	Fail	-
#2.7		92	0.54033	**0.59045**	Pass	27.197
#2.8	10^6	14	0.59219	0.79645	Pass	15.748
#2.9		71	0.60952	0.60479	Pass	8.9292
#2.10		60	**0.51366**	0.69112	Pass	13.457

For the convergence study with time, the model is trained to the same rolling disk hyperparameters described in 3.11. As a reminder, this is using the standard features and targets for the model, described in Appendix D. The open-source GNS model is written such that the time step is not used. Instead, the given set of data is assumed to have a uniform time step between all frames. To accommodate this constraint, 6 separate datasets are created for the chosen time steps. All 6 datasets are derived from the same set of simulations. From this set, position entities are exported at intervals respective to the desired time step.

GNS RESULTS

Table 3.12 shows two total training steps to determine if the optimization problem is consistent with the benchmark performance. Second, seed variations are to investigate stability. Table 3.12 lists the computed loss values for the trained model to the unseen test dataset. The minimum magnitudes in each column are highlighted in **bold**. A de-



(a) Center of mass displacement (b) Center of mass acceleration (c) Center of mass acceleration

Figure 3.16: GNS model benchmark predictions at 10^5 training steps

scription of both the error metric formulations, and the computational complexity of the model are shown in Appendix D.

Figure 3.16a shows the GNS model predictions at 10^5 training steps, the MLS-MPM method used to generate the dataset, and the true analytical solution (Equation 3.2). At 10^5 training steps, the GNS model shows a wide range of predictions made as the seed is varied.

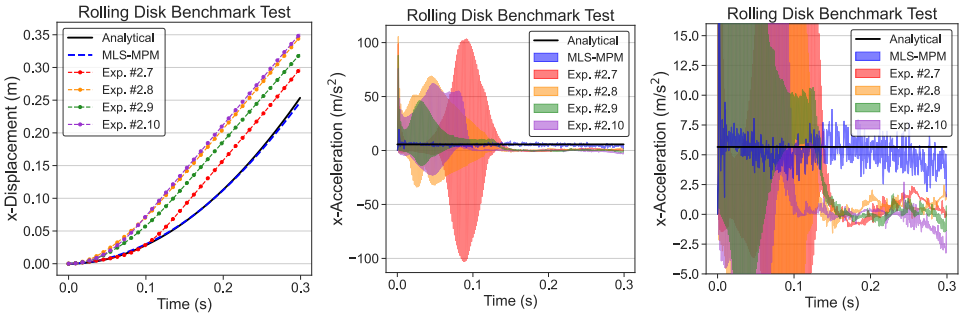
Figure 3.17a shows the GNS predictions for 10^6 training steps. From the predictions at 10^5 training steps to 10^6 training steps, it is seen the GNS predictions go from a linear to non-linear nature. There continues to be a distribution in the results as the seed is varied.

The MSE is evaluated on normalized per particle acceleration. A reminder, the equation for this error metric is given in Table 3.2. To build understanding of the error metric magnitudes shown in Table 3.12, the acceleration values of the respective curves are computed for the body center of mass $\ddot{x}_{m_c}$ in the x direction using twice second order central differences via `numpy.gradient`. No means to filter the noise are implemented. This is to elucidate the noise present.

These accelerations are shown in Figure 3.16b and Figure 3.17b. Figure 3.16c and Figure 3.17c provide the same acceleration data, on a zoomed in visual. This shows the noise in the MLS-MPM data, and the true analytical solution. There is a significant amount of noise seen for the GNS predictions compared to the MLS-MPM ground truth model. This is quantified with the standard deviation in the center of mass acceleration $\sigma(\ddot{\mathbf{x}}_{m_c})$, shown in Table 3.12. Velocity plots for the center of mass are shown in Appendix K.

The training and inference times are reported in Table 3.13.[10] There is a distinct difference in the training time per step between the experiments at 10^5 training steps and 10^6 training steps. This is due to the time including both the training and validation calculations. All setups use a constant 1000 validation checks. The reason for using a constant

[10] These times are for 12 Intel Xeon CPU E5-2620 0 @ 2.00GHz.



(a) Center of mass displacement (b) Center of mass acceleration (c) Center of mass acceleration

Figure 3.17: GNS model benchmark predictions at 10^6 training steps

Table 3.13: Training and inference times for the rolling disk benchmark

Experiment	Training steps	Training time/ step (s)	Inference time/ step (s)
#2.1		1.004	104.5
#2.2		0.9948	104.2
#2.3	10^5	0.9962	103.7
#2.4		0.9899	104.5
#2.5		0.9893	104.6
#2.6		0.6185	103.7
#2.7		0.6238	105.1
#2.8	10^6	0.6213	102.7
#2.9		0.6199	104.5
#2.10		0.6243	104.7

checkpoint count are to reduce computational costs. Appendix N shows the validation curves, and an adequate training process. If no validation is used, the average training time per step is 0.57 s. The MLS-MPM model requires 60 s [11] to compute and store the data of an entire trajectory. The GNS model inference has a 73% time increase over the MLS-MPM method.

A visualization of one trajectory is shown in Figure 3.18. It shows the disk rolling along the boundary with time. The vector field and rotational energy is given in Appendix K.

The second numerical study performed evaluates convergence with an increase in the time resolution. As with conventional numerical methods, there is an expectation that the numerical method ought to converge on the true solution as the time step is reduced. Table 3.14 shows the resultant error metric magnitudes for the test, with the minimum of each column highlighted in **bold**. Table 3.14 shows the GNS model fails the shape

[11] This time is for 1 Intel(R) Core CPU i7-6700HQ @ 2.60Ghz.

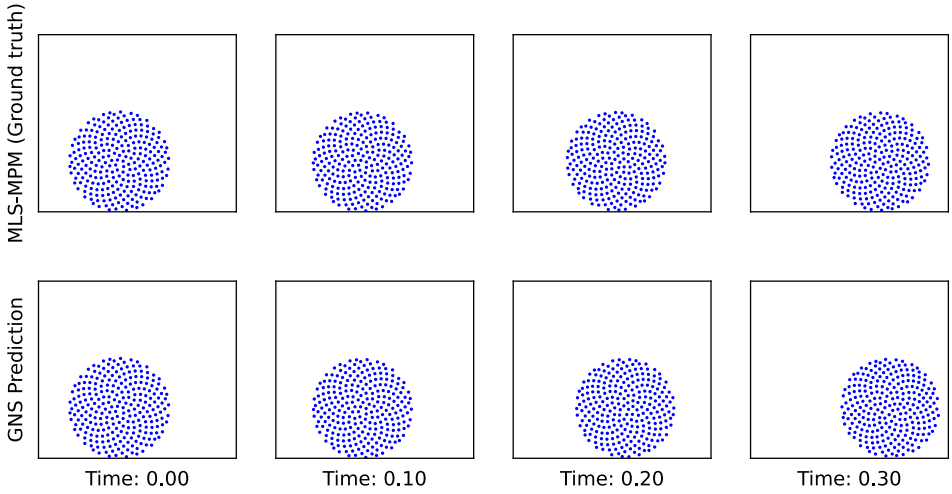


Figure 3.18: Visualization of experiment #2.7

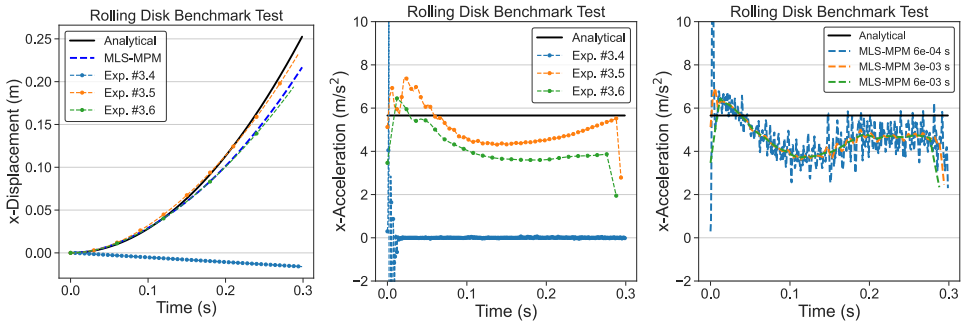
Table 3.14: Rolling Disk Time Convergence DoE (seed = 33, training steps = $5{\cdot}10^4$)

Experiment	Time step (s)	MLS-MPM $\sigma(\ddot{x}_{m_c})$ (m/s^2)	Acceleration MSE ($\cdot 10^{-3}$)	One step Position MSE ($\cdot 10^{-10}$)	Shape tolerance $\varepsilon = 5\%$	GNS $\sigma(\ddot{x}_{m_c})$ (m/s^2)	Inference time / trajectory (s)
#3.1	3e-05	13.160	**0.31762**	**1.0150**	Fail	-	821
#3.2	6e-05	3.9157	0.74283	1.9920	Fail	-	406
#3.3	3e-04	1.3468	1.1624	3.3838	Fail	-	105
#3.4	6e-04	1.1469	2.0290	6.3991	Pass	2.5649	52.8
#3.5	3e-03	0.7617	5.6269	18.107	Pass	0.8440	10.2
#3.6	6e-03	0.7576	18.518	59.483	Pass	0.8541	4.86

criteria as the time step is refined, and the MSE magnitudes increase as the time step increases. Table 3.14 quantifies an increase in the noise in the MLS-MPM acceleration data as the time step is reduced. This is visualized in Figure K.4.

Figure 3.19a shows the center of mass displacement for the GNS predictions which pass the shape tolerance criteria, the MLS-MPM solution trained on, and the analytical solution. At the two coarsest time steps of 3e-03 and 6e-03, the GNS prediction closely matches the ground truth model. Figure 3.19c shows that as the time step is reduced, more noise appears in the MLS-MPM center of mass acceleration. Figure 3.19b and Figure 3.19c show a decay of the acceleration magnitudes at the first two and final two time steps. This is caused by truncation errors at the bounds from `numpy.gradient`.

Last, Table 3.14 shows as the time step increases, the inference time per trajectory reduces. At its shortest inference time of 4.86 s, there is a 92% time decrease compared to the time cost of the MLS-MPM method.



(a) Center of mass displacement (b) Center of mass acceleration (c) Center of mass acceleration

Figure 3.19: Benchmark plots for a time convergence study

DISCUSSION OF RESULTS

The results of these tests are analyzed in the following paragraphs to make conclusions on these numerical method core properties. It is shown that the optimization problem is consistent with the benchmark performance, while stability and convergence fail. The success with training comes from the ability of the GNS model to approximate the benchmark domain, while the stability and convergence failures are due to noise in the feature and target domains. Last, the discrepancy between the magnitudes of the error values in Table 3.14 and the benchmark performance is discussed.

From 10^5 training steps in Figure 3.16a, to 10^6 training steps in Figure 3.17a, there is an improvement in the GNS model to predict the center of mass displacement. This is indicative that the chosen optimization problem is consistent with the benchmark performance.

The same two benchmark metrics of Figure 3.16a and Figure 3.17a show that with this increase in training steps, comes a closer grouping of the provided benchmarks with the varied seed. However, Figure 3.17b shows and Table 3.12 quantifies there is a significant variance of noise in the center of mass acceleration $\ddot{x}_{m_c}$. Noise is quantified on the standard deviation in the center of mass acceleration $\ddot{x}_{m_c}$. This is a failure of the model stability in this application.

The GNS authors describe a velocity random walk noise sequence used to perturb input position data [85]. The motivation for this is to force the GNS model to correct for errors it produces. In the open source code model, the magnitude of this noise is fixed. Algorithm D.4 shows this implemented as σ_0. This σ_0 is fixed at 0.00067.[12]

The added noise to the position value alters the definition of acceleration. The GNS model trains on acceleration values computed from twice central differences of position data. To compute the acceleration $\ddot{\mathbf{x}}(t)$, the random walk noise perturbed positions of $\mathbf{x}(t)$, $\mathbf{x}(t-1)$ and $\mathbf{x}(t+1)$ are used. As noted in the paper, the noise values in $\mathbf{x}(t)$ and $\mathbf{x}(t+1)$ are equal, and cancel out. This yields a next time step velocity $\dot{\mathbf{x}}(t+\frac{1}{2})$ with no

[12] No units were given in the publication of Sanchez-Gonzalez et al. on this quantity.

noise. However, there is noise in $\mathbf{x}(t)$ and $\mathbf{x}(t-1)$, when computing $\dot{\mathbf{x}}(t-\frac{1}{2})$. This noise persists in computing the target $\ddot{\mathbf{x}}(t)$ from the ground truth position data. This is shown in Equation 3.4.

$$\tilde{\ddot{\mathbf{x}}}(t) = \frac{\mathbf{x}(t+1) - \mathbf{x}(t) - \tilde{\mathbf{x}}(t) + \tilde{\mathbf{x}}(t-1)}{\Delta t^2} \tag{3.4}$$

Figure 3.16b and Figure 3.17b show substantially more noise in the center of mass acceleration compared to the ground truth MLS-MPM benchmark. The noise in the MLS-MPM benchmark is highlighted in Figure 3.16c and Figure 3.17c.

Additionally, Figure 3.16b and Figure 3.17b show noise for acceleration values for the center of mass m_c in the MLS-MPM benchmark. This is noise averaged over the entire body. It is expected there is more severe noise in each individually computed particle. The noise in the random walk sequence, and the noise present in the position data yields a poorly defined target for the model. The result is asking the ML model to learn through a substantial amount of noise.

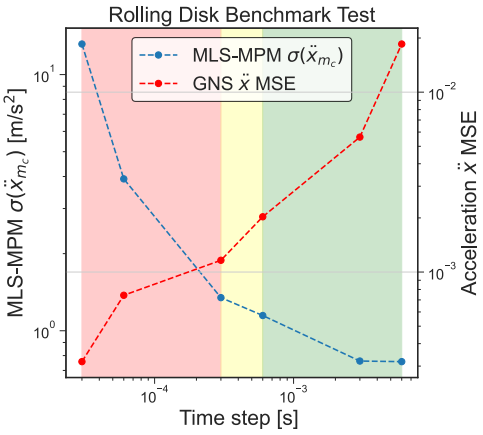
The convergence test with time shows a failure of the GNS model. Figure 3.19a shows that a reduction in the time step yields poorer benchmark results. Table 3.14 shows the 3 experiments with the smallest time steps all fail the shape tolerance. This comes from a limitation of the data provided to the model and the choice of noise added at the input level.

First, the convergence study highlights the consequence of a fixed random walk noise magnitude. As one reduces the time step, this fixed magnitude σ_0 which one perturbs the position values becomes more significant. A reminder, these perturbed positions are then used to compute the velocities of the particles using finite differences to be used as node feature. As the time step decreases, this fixed noise magnitude has a more substantial effect on the perturbations applied to positional data. This quickly changes to incomprehensible noise at the input as the time step decreases.

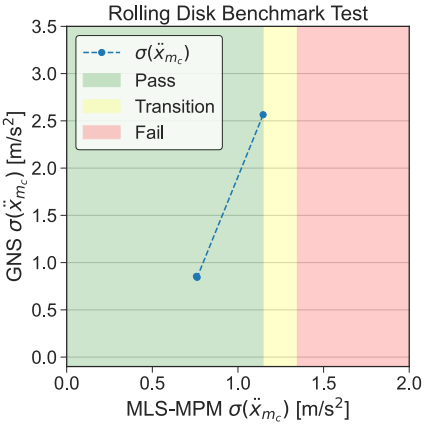
Next, noise in computing acceleration through finite difference of position hinders the model. Shown in Figure 3.20a, as one reduces the time step, more noise appears in the ground truth acceleration data. The GNS implementation section states the varying time step datasets are derived from the same simulation data. The only difference is the time step frequency which this data is organized. Thus, as the time step between this data reduces, the noise in the second finite difference derivative grows.

Figure 3.20b shows that with an increase in the noise in the MLS-MPM acceleration data, a noise increase in the prediction by the GNS model is seen. This is expected, as the MLS-MPM acceleration is by definition set as the goal in the loss term. The acceleration loss is evaluated for N particles, over k time steps, j dimensions, their position $\mathbf{x}$, normalized acceleration $\hat{\ddot{\mathbf{x}}}$ and GNN acceleration prediction $d_\theta(\mathbf{x}_i^{t_k})$. This is shown in Equation 3.5.

$$\mathscr{L} = \frac{1}{K}\frac{1}{N}||d_\theta(\mathbf{x}_i^{t_k}) - \hat{\tilde{\ddot{\mathbf{x}}}}_i^{t_k}||^2 \tag{3.5}$$



(a) Time dependencies



(b) Standard deviation dependencies. 3 data points are shown, but two lie near the other in the bottom left of the line.

Figure 3.20: Relations of noise with the time step. Shape tolerance pass/fail zones are highlighted

The normalized target acceleration is computed from both the dataset statistics and the noise magnitude σ_0. Further, it is normalized in time by assuming $\Delta t = 1$ in Equation 3.4.

$$\hat{\tilde{\mathbf{x}}}_i^{t_k} = \frac{\tilde{\ddot{\mathbf{x}}}_i^{t_k} - \mathbf{a}_{\text{mean}}}{(\mathbf{a}_{\text{std}}^2 + \sigma_0^2)^{1/2}} \tag{3.6}$$

Figure 3.20a shows that as the noise in the MLS-MPM data increases, the prediction fails the shape tolerance. Figure 3.19b shows that the 2 experiments (#3.5 & #3.6) with the lowest noise in the ground truth data adequately meet the benchmark. These results show that noise at the input level significantly affects the performance of the GNS model.

There is a clear discrepancy between the acceleration MSE and the performance of the benchmark. Figure 3.20a shows that at the highest MSE values, largest error, the predictions pass the shape tolerance. This is a result of normalizing the data. As the time step decreases, the displacement between each each step decreases. This yields smaller values in the numerator of equation 3.4. Yet, the denominator remains fixed at $\Delta t = 1$. This biases the magnitude of the acceleration magnitudes based on the time step.

However, the target $\hat{\tilde{\mathbf{x}}}_i^{t_k}$ in the loss Equation 3.5 is normalized. Equation 3.6 shows this normalization. The magnitude of σ_0 is fixed, and as the time step decreases, σ_0 has much heavier influence on the normalized values. Figure 3.21 shows that as the time step decreases, a_{std}, yet σ_0 remains the same.

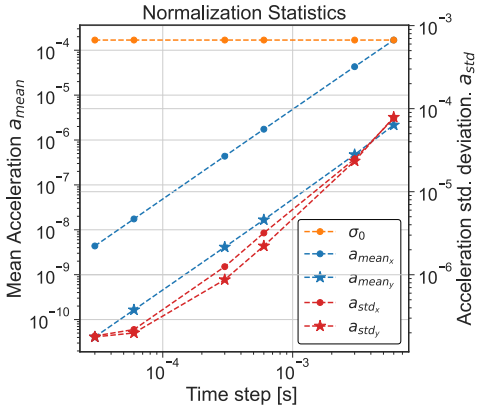


Figure 3.21: Normalization statistics

The result of this normalization, is that the acceleration magnitude decreases in time due to the normalization. The MSE magnitude is dependent on the absolute value of the quantity it evaluates. As the acceleration magnitudes decrease, due to normalization, this is reflected in the MSE metrics dependency on the absolute value it evaluates. Section 4.1 will show the results when normalizing to only standard deviation in the data, and not σ_0.

A closing remark in this discussion section are the computational times. The elastic bodies benchmark showed a time increase of 7,650% for the GNS model compared to the ground truth method. This rolling disk implementation reduced this to 73%. This arises from the time step which the ground truth method must take. The elastic bodies ground truth method uses a time step of 1e-03 s, while the rolling disk uses a time step of 1e-06 s. This is a requirement of the explicit numerical method to meet the Courant Number. For the time being, the GNS model is able to take coarse time steps, and achieve representative results of the benchmark. In making coarse time steps, it has the potential to reduce the computational cost compared to the ground truth method. This is shown with the rapid 92% reduction in computation cost for the GNS prediction compared to the MLS-MPM method at the coarsest time step.

3.3. CONCLUSION

This thesis chapter shows reimagining the error metric allows a state of the art GNN model to be evaluated on isolated physical laws. Each DoE is defined to evaluate the GNN model on basic numerical method properties.

The elastic bodies benchmark showcased the influence of graph construction choice, as it limits stability. Likewise, the rolling disk benchmark failed stability, due to the severe noise present rooted in the random walk noise sequence. Time convergence continues to allude the model in the rolling disk benchmark.

The metrics of consistency, stability and convergence and implementations of bench-

marks were fundamental in developing the finite difference method as a widely used numerical method to solve PDEs [110, 111]. The connection with this work drawn here places the GNN model on a stage to be evaluated via first principles.

4

MODEL INVESTIGATION

The hope of this numerical method is it will capture a range of snow simulations. This requires it is robust to a breadth of problems. Chapter 3 showed the method successes and failures in two examples. This chapter identifies limiting factors of the model, motivates remedies to these shortcomings, and analyzes results of implementing these alterations. Key alterations to the method are made with intentions to improve its generalization capabilities. Specifically, this investigates the choice of node features and how the prediction loss is evaluated. It is found evaluating the loss term on position, instead of acceleration, improves performance when using unnormalized features and targets.

4.1. NORMALIZING EFFECTS

The use of normalization statistics is the first limiting factor identified.[1] By normalizing features and targets to scalar quantities descriptive of the training dataset, the model is inherently constrained to physical setups in the training dataset. This is undesirable. Ad-

[1] Appendix D describes what these scalar quantities are, where they come from and how they are used.

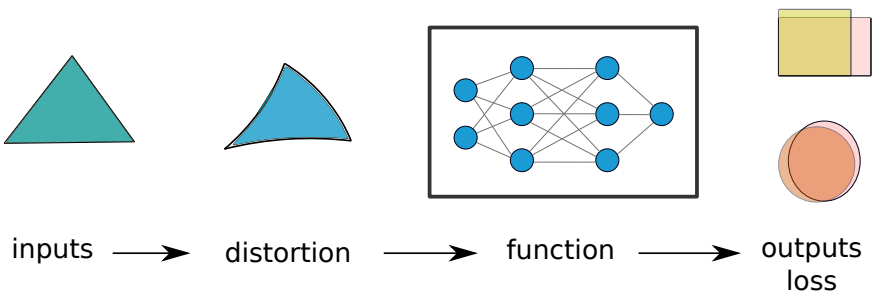


Figure 4.1: A system model of inputs and outputs

ditionally, the model is written normalizing all velocities and accelerations to a time step Δt of 1. This is claimed in the original publication to be done for simplicity [85].

The second shortcoming identified is using a fixed noise magnitude for the random walk noise sequence. Section 3.2's discussion highlights this constricts the model when reducing the time step in the dataset. Specifically, that it supplements too aggressive of noise as the time step is reduced.

Figure 3.20a showed the MSE magnitude is not conclusive to the benchmark performance when varying the time step. Section 3.2's discussion identifies this comes from normalizing the acceleration variance to both the dataset statistics and the random walk noise magnitude σ_0. Normalizing to the random walk noise biases the magnitude in the acceleration data, causing the MSE magnitude to be misleading.

Key model alterations are made to address the issues identified in these first paragraphs. In Figure 4.1 this is the choice of how the inputs are distorted, and how the outputs are evaluated. The normalization statistics are removed to promote generalization capabilities. These scalar values, computed in Algorithm D.5, may be nullified by setting the standard deviation to unity, and the mean to zero. This yields the values non-effective in the code.

The real time step is introduced as it is physically relevant. The time step is added by careful evaluation of where time integration or differentiation takes place, and introduced accordingly. For example, in Equation 3.4, Δt becomes the real time step associated with the dataset, instead of unity. The time differentiation and integration is assured to follow a central finite difference scheme [130] and consistent units are used in the model.

A solution to the fixed noise value σ_0 is not trivial. The following decision is made. Each training step makes use of a batch size of n graphs from n random time steps, characterized by a batch of position values $\mathbf{x}_i^{t_k}$. Sanchez-Gonzalez et al. [85] use a batch size of 2 graphs. In this case the batch of position values $\mathbf{x}_i^{t_k}$ act as input to the random walk noise sequence in Algorithm D.4. This batch of $\mathbf{x}_i^{t_k}$ is used to compute a standard deviation in the velocity values. The standard deviation in this batch alone is used as σ_0. This σ_0 is for each batch of graphs selected. Equation 4.1 shows the calculation of σ_0 via `TensorFlow`'s `reduce_std` function.

For a batch of position values $\mathbf{x}_i^{t_k}$, the velocity at these points is computed according to the finite difference at that point.

$$\dot{\mathbf{x}}_i^{t_k} = \frac{\mathbf{x}_i^{t_k} - \mathbf{x}_i^{t_{k-1}}}{\Delta t}$$

$$\sigma_0 = \alpha \cdot \texttt{tf.math.reduce\_std}(\dot{\mathbf{x}}_i^{t_k}) \tag{4.1}$$

DESIGN OF EXPERIMENTS

Equation 4.1 shows a new hyperparameter α. This value determines the magnitude of noise added based on the standard deviation in the batch of velocity values $\dot{\mathbf{x}}_i^t$. This requires a numerical study to understand the effects of this hyperparameter. This is done over a logarithmic search of the value magnitude.

Second, the effects of normalizing the data are of interest. Sanchez-Gonzalez et al. [85] motivate the use of normalizing the features and targets as it reduces the test MSE magnitude. However, Chapter 2 and Chapter 3 highlight these MSE magnitudes give little information over the underlying physics. The use of the features and targets being normalized or unnormalized are thus investigated.

GNS IMPLEMENTATION

This implementation uses the same rolling disk benchmark implementation described in Section 3.2. The hyperparameters for the GNS model are listed in Table 4.1. The GNS model is implemented on a time step of 6e-04 seconds. The reason being, Figure 3.19 showed that this time step is the transition between acceptable and unacceptable results. To elucidate the changes made, the equations defining the features and targets edited are included here.

The velocity features are defined according to Equation 4.2. The output acceleration $\tilde{\ddot{\mathbf{x}}}(t)$ is defined by Equation 3.4.

$$\tilde{\dot{\mathbf{x}}}_i^{t_k} = \frac{\tilde{\mathbf{x}}_i^{t_k} - \tilde{\mathbf{x}}_i^{t_k-1}}{\Delta t} \tag{4.2}$$

The node feature velocity $\hat{\tilde{\dot{\mathbf{x}}}}_i^{t_k}$ and target acceleration $\hat{\tilde{\ddot{\mathbf{x}}}}_i(t)$ definitions are shown in Equation 4.3 and Equation 4.4. The denominators contain only $\mathbf{v}_{\text{std}}$ and $\mathbf{a}_{\text{std}}$, and not σ_0, as used in Equation 3.6. For the unnormalized experiments, setting the $\mathbf{a}_{\text{mean}}$ to zero and $\mathbf{a}_{\text{std}}$ to one, renders this normalization null.

$$\hat{\tilde{\dot{\mathbf{x}}}}_i^{t_k} = \frac{\tilde{\dot{\mathbf{x}}}_i(t) - \mathbf{v}_{\text{mean}}}{\mathbf{v}_{\text{std}}} \tag{4.3}$$

$$\hat{\tilde{\ddot{\mathbf{x}}}}_i(t) = \frac{\tilde{\ddot{\mathbf{x}}}_i(t) - \mathbf{a}_{\text{mean}}}{\mathbf{a}_{\text{std}}} \tag{4.4}$$

In summary, this study implements a new means to define the magnitude σ_0 which is not fixed. Instead, it is defined relative to the standard deviation in the input velocity set. Second, normalization is investigated. A normalized `True` study uses $\Delta t = 1$, the normalization statistics defined by Algorithm D.5, and their implementation in Equation 4.3 and Equation 4.4. A normalized `False` study uses Δt the dataset is defined on, a mean normalization statistic of $\mathbf{0}$ and a standard deviation normalizing statistic of $\mathbf{1}$ to yield them ineffective. The loss continues to be that defined by Equation 3.5.

Table 4.1: Rolling disk GNS model implementation constants

GNS Hyperparameters		Dataset Parameters	
GNN Architecture	`Interaction Network`	Total time [s]	0.30
Loss function	L_2 Norm Acceleration	θ [rad]	π / 3
Learning optimizer	Adam	Disk radius r [cm]	25
Learning rate	exponential decay from 10^{-4} to 10^{-6}	Density [kg/m^3]	12,000
Training steps	$5{\cdot}10^4$	Mesh resolution [n/r]	0.33R
Message passing steps	10	Num. particles	252
NN Architecture	2 hidden layers 128 layer neurons	x_0	[0.3, 0.45]
Noise std	Adaptive	ϕ_0	$[0, 2\pi]$
Input sequence length h	6	Train/test sets	300/20
Connectivity radius [m]	0.07		
Particle type	None		
Seed	33		

GNS RESULTS

Table 4.2 shows the results when investigating the α hyperparameter for normalized and unnormalized features and targets. The first key item seen is the shape tolerance set. This magnitude is relaxed from 5%, to 10% as no predictions pass at 5%. Second, there is a clear increase in the acceleration and position MSE magnitudes with an increase in α. Third, even though the acceleration and position MSE values are smaller, at lower α values, the shape tolerance criteria fails.

Figure 4.2 shows the results of the two benchmarks which do indeed pass the shape tolerance criteria in Table 4.2. Figure 4.2a shows both trajectories are near the ground truth displacement. Figure 4.2b shows there is a significant amount of noise in the acceleration data for the unnormalized trajectory, Exp. #4.10. Figure 4.2b shows the single normalized experiment which passes, Exp. #4.4. This trajectory has less noise than the ground truth model, as quantified with the standard deviation in the center of mass acceleration $\ddot{x}_{m_c}$ in Table 4.2.

DISCUSSION OF RESULTS

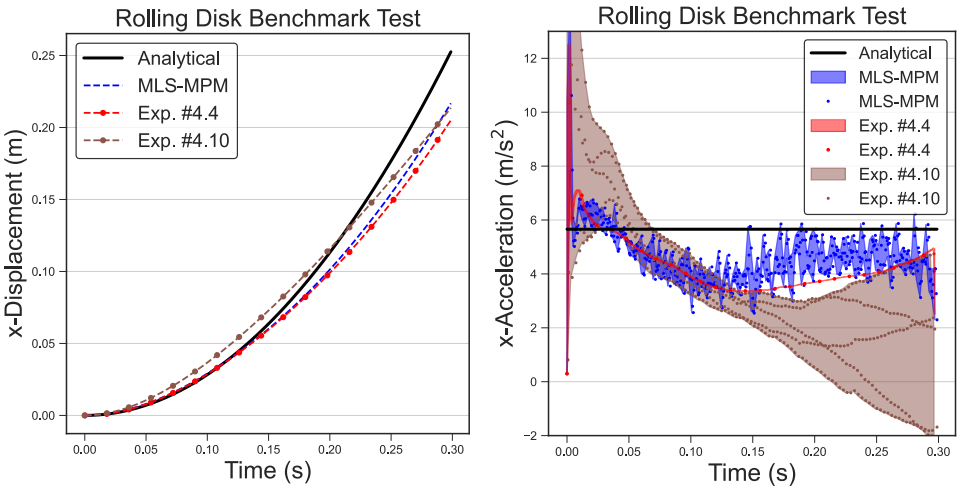
The motivation to investigate the normalization of features and targets stems from the lack of physical basis known when normalization is used. This is explored together with the introduction of a new hyperparameter to avoid a fixed noise value. This discussion section highlights that normalizing these values reduces the noise in the predictions.

Table 4.2 shows the introduction of this hyperparameter yields a significant degradation of the ability for the GNS model to maintain the shape of the item it predicts. This entails the model is sensitive to the manner which noise is added at the input level.

Figure 4.2 shows that unnormalized features and labels yields significant noise in the predicted rollouts. It is also striking to see Exp. #4.4 in Figure 4.2b. The GNS model,

Table 4.2: DoE for Set #4 - Rolling Disk (Δt=6e-04s)

Experiment	Normalize	alpha α [%]	Acceleration MSE	One step Position MSE ($\cdot 10^{-12}$)	Shape tolerance ε = 10%	GNS $\sigma(\ddot{x}_{m_c})$ (m/s^2)
MLS-MPM	-	-	-	-	-	1.1469
#4.1		0.0	**1.4266**	**9.5014**	Fail	-
#4.2		0.1	1.4678	9.8081	Fail	-
#4.3	True	1.0	1.7199	10.204	Fail	-
#4.4		10	6.7167	27.354	Pass	1.0340
#4.5		100	54.576	163.91	Fail	-
#4.6		0.0	75.504	19.786	Fail	-
#4.7		0.1	77.493	9.9301	Fail	-
#4.8	False	1.0	95.311	10.028	Fail	-
#4.9		10	380.56	29.535	Fail	-
#4.10		100	3032.6	141.74	Pass	2.9191



(a) Center of mass displacement

(b) Center of mass acceleration

Figure 4.2: Results of searching for a new alpha hyperparameter

for normalized features and prediction, produces an acceleration rollout with significant less noise than the data it is trained on. This indicates the robustness added to the model when normalizing input and outputs.

Table 4.2 shows that as α increases, there is a corresponding increase in the acceleration MSE. This is a result of the target acceleration being partially defined from the position sequence with a random walk noise added. Section 3.2's discussion addresses how the random walk noise appears in the target acceleration. As the noise magnitude increases with α, there is a corollary increase in the noise in the target value. Section 3.2's discussion details how the target acceleration is dependent on the random walk noise added. This means the GNS model is trained on noisier and noisier target acceleration values as α increases.

The three main conclusions of this discussion section are: (*i*) The GNS model tends to produce stabilized predictions for normalized features and targets, (*ii*) the GNS model is sensitive to the means by which noise is added at the input level, (*iii*) there is a clear limitation by noise for computing the targets through twice finite differences of position with a random walk noise added. In continuing the investigation of normalizing the inputs and outputs, the effects of a time convergence study is now completed.

DESIGN OF EXPERIMENTS

The numerical study of α showed that the only prediction which passes the shape tolerance for unnormalized values is experiment #4.10. This is at an α value of 100%. This value will be further explored. The convergence study of Table 3.19 is repeated. Section 3.2 showed the limitations of a fixed noise value has at the input level. The intention is to understand the effect of this new hyperparameter as time is varied.

The continued search of unnormalized features and targets are of interest, as they do not require the use of scalar normalizing values defined on the training dataset.

GNS IMPLEMENTATION

The same datasets from the convergence study in Section 3.2 are used. The same GNS hyperparameters as the α value study are used. These are listed in Table 4.1.

GNS RESULTS

Table 4.3 shows the results of varying the time step under a single α value of 100%. As the time step increases, there is a reduction in the acceleration MSE magnitude and an increase in the one step position MSE magnitude. Experiment #5.1 is an exception to this. The unnormalized experiments all fail the shape tolerance of 5%, accept at the smallest time step. See Table 3.14 for the standard deviation in the MLS-MPM acceleration data.

Figure 4.3 plots the results of these implementations. Figure 4.3a shows at a coarse time step for normalized values, the altered GNS model comes near the ground truth model. Figure 4.3a also shows for the single unnormalized experiment which passes the shape tolerance, a prediction with a distinct linear trend is seen.

Figure 4.3b shows the normalized experiments at a coarser time step have similar predictions to the respective ground truth model in Figure K.4. Figure 4.3b shows significant

Table 4.3: DoE for Set #5 - Rolling Disk (α = 100%)

Experiment	Normalize	Time step (s)	Acceleration MSE	One step Position MSE ($\cdot 10^{-12}$)	Shape tolerance ε = 5%	GNS $\sigma(\ddot{x}_{m_c})$ (m/s^2)
#5.1	True	3e-05	35.166	**0.77730**	Pass	2.7880
#5.2		6e-05	94.199	3.3619	Fail	-
#5.3		3e-04	84.928	91.470	Fail	-
#5.4		6e-04	54.576	163.91	Fail	-
#5.5		3e-03	30.802	4692.0	Pass	0.67588
#5.6		6e-03	**6.0854**	25175	Pass	1.1451
#5.7	False	3e-05	2832363	0.83939	Pass	10.586
#5.8		6e-05	326235	1.9817	Fail	-
#5.9		3e-04	13170	47.797	Fail	-
#5.10		6e-04	3032.6	141.74	Fail	-
#5.11		3e-03	141.08	4590.3	Fail	-
#5.12		6e-03	32.276	19360	Fail	-

more noise for Experiment #5.7 (unnormalized), which uses the same time step as Experiment #5.1 (normalized). Figure 3.19c shows the corresponding ground truth data for this small time step of 3e-05 with a significant amount of noise. This noise is quantified in Table 4.3. The velocity plots for these respective setups are shown in Appendix K.

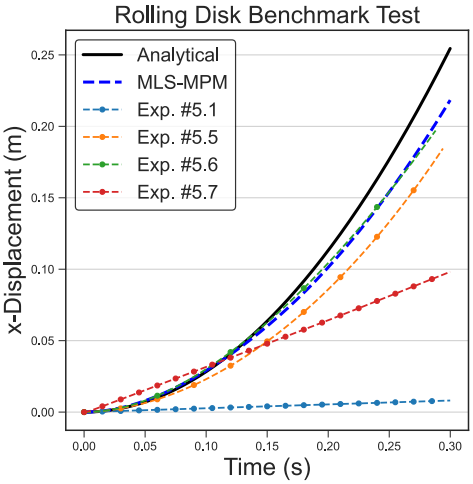
DISCUSSION OF RESULTS

Table 4.3 shows the convergence study, with the reformulation of the noise at the input level. Without the normalization by the noise, as done in Table 3.14, there is the expected acceleration MSE values on a similar order of magnitude for the normalized experiments. The reason being, although different amounts of noise appear in the ground truth data, dependent on the time step, all models use use their respective normalization statistics. These statistics bring the target and prediction values to a range of zero mean and unit variance. Thus, all normalized experiment quantities are evaluated are on a similar magnitude scale.

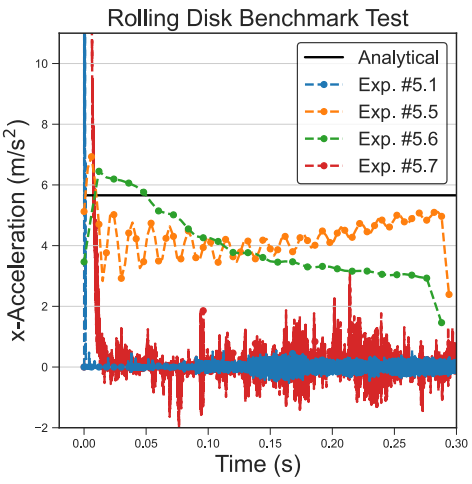
In contrast, the unnormalized predictions showcase the additional noise in the ground truth data as the time step is varied. As discussed, as the time step reduces, more noise appears in the ground truth data. This results in the target prediction magnitudes increase. The dependency of the MSE value on the absolute quantity it evaluates is reflected here. This phenomena is further explored in the next Section 4.2, including for the one step position MSE values.

The normalized experiments in Table 4.3 which pass the shape tolerance, have the lowest acceleration MSE values in the normalized category. This supports that although significant noise in acceleration as a loss quantity is reflected in the GNS performance, as an error metric quantity, it appears to be well indicative.

Last, it is striking to see for unnormalized experiments which pass the shape tolerance,



(a) Center of mass displacement



(b) Center of mass acceleration

Figure 4.3: Results of evaluating with acceleration

Exp. #4.10 and Exp. #5.7. These experiments are both the highest α value of 100% in Table 4.2 and the smallest time step in Table 4.3. These two experiments respectively have the largest amount of noise in their respective unnormalized sets. At the largest α value, there is the most amount of noise applied. At the smallest time step, there is the most amount of noise in the ground truth data. This is indicative that the neural network is robust to added noise.

4.2. LOSS QUANTITY

The noise limitations identified by using acceleration to compute the target motivate a reformulation of the target value. Current literature encourages the use of using acceleration as the target quantity to leverage the symmetry geometric prior[2]. However, the current noise found in using acceleration from position suggests a reformulation of this.

Although there is a clear motivation to use acceleration as the predicted quantity [126, 128], there are prominent examples where this is not the case [109, 131]. This uncertainty in literature, and numerical results seen here suggest further investigation is required. This is done by continuing to predict acceleration, but now shifting the focus to evaluate the loss on position. The motivation is to leverage: (*i*) the symmetry geometric prior by predicting acceleration, (*ii*) compute the loss on a target quantity which has minimal noise.

[2] Bronstein et al. [126] describe this Bayesian perspective in depth to motivate its ability to reduce a high dimensional space to a lower dimension. This is to facilitate universal approximation at a reduced count of training steps. This is explained in Appendix E

DESIGN OF EXPERIMENTS

Section 3.2 highlighted that during a convergence study of the rolling disk benchmark, the noise in the acceleration target hinders the ability of the GNS model to learn the underlying function. This hindrance of noise makes the experimental setup ideal for testing a resolution which intends to alleviate this noise. Thus, the convergence study of Section 3.2 is repeated.

As current understanding in using unnormalized values is not complete, this experiment will look at unnormalized values. As an α value of 100% showed promising results in Table 4.2, it is used here.

GNS IMPLEMENTATION

The GNS model is implemented the same as shown in Table 4.1, with the single difference that the loss is now evaluated on the L_2 norm of position instead of acceleration. Further, the use of unnormalized values is identical to that used in Section 4.1.

Equation 4.6 shows this L_2 norm between the batch of ground truth positions $\mathbf{x}^{t_k}(t+1)$, and the predicted position $\mathbf{x}^{t_k\prime}(t+1)$. The model continues to predict acceleration $\ddot{\mathbf{x}}^{t_k\prime}(t)$, shown in Equation 4.5. However, through integration a position value is now obtained. This predicted position is then compared against the ground truth position value with no noise added. This is shown in the following integration scheme:

$$\ddot{\mathbf{x}}^{t_k\prime}(t) = d_\theta(\mathbf{x}_i^{t_k}) \tag{4.5}$$

$$\dot{\mathbf{x}}^{t_k\prime}(t+\frac{1}{2}) = \dot{\mathbf{x}}^{t_k}(t-\frac{1}{2}) + \ddot{\mathbf{x}}^{t_k\prime}(t)\cdot\Delta t$$

$$\mathbf{x}^{t_k\prime}(t+1) = x^{t_k}(t) + \dot{\mathbf{x}}^{t_k\prime}(t+\frac{1}{2})\cdot\Delta t$$

$$\mathscr{L} = \frac{1}{K}\frac{1}{N}||\mathbf{x}^{t_k\prime}(t+1) - \mathbf{x}^{t_k}(t+1)||^2 \tag{4.6}$$

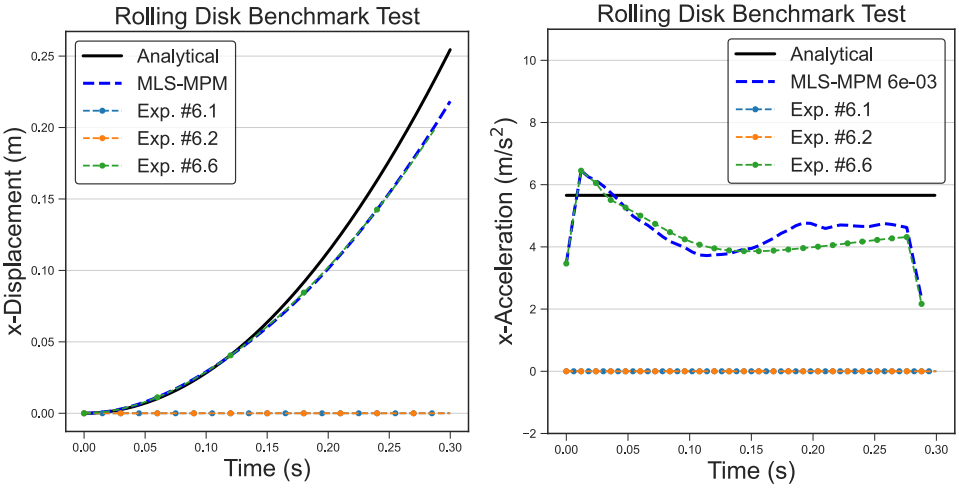
GNS RESULTS

Table 4.4 shows the results of the numerical convergence study. There is a decrease in the acceleration MSE magnitudes and increase in the position MSE magnitudes as the time step increases.

Figure 4.4 shows the benchmark predictions for those which passed the shape tolerance. At a coarse time step, the GNS prediction matches the ground truth model. Once again, Figure 4.4b shows truncation errors from `numpy.gradient` at the first two and final two time steps. The two smallest time steps pass the benchmark, and show stationary, noiseless results.

Table 4.4: DoE for Set #6 - Rolling Disk (α = 100%, Normalize = False)

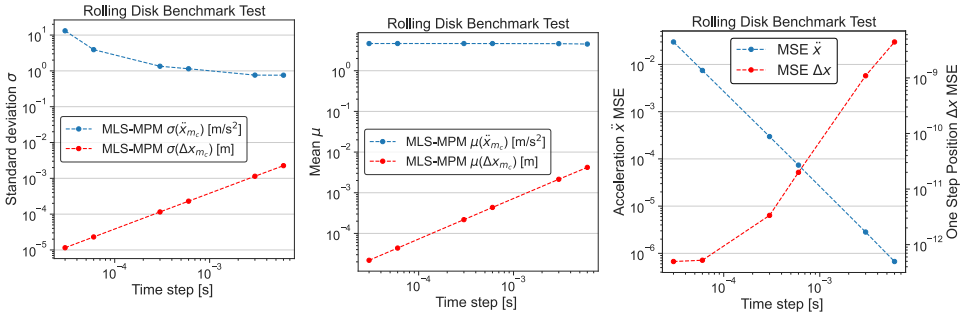
Experiment	Time step (s)	Acceleration MSE ($\cdot 10^{+6}$)	One step Position MSE ($\cdot 10^{-12}$)	Shape tolerance $\varepsilon = 5\%$	GNS $\sigma(\ddot{x}_{m_c})$ (m/s^2)
#6.1	3e-05	299.27	**0.49654**	Pass	0.0
#6.2	6e-05	74.616	0.52337	Pass	0.0
#6.3	3e-04	2.9712	3.3570	Fail	-
#6.4	6e-04	0.73887	19.966	Fail	-
#6.5	3e-03	0.028257	1092.8	Fail	-
#6.6	6e-03	**0.0067159**	4433.9	Pass	0.76270



(a) Center of mass displacement

(b) Center of mass acceleration

Figure 4.4: Results of evaluating on position



(a) Center of mass standard deviation values

(b) Center of mass mean values

(c) MSE error metric comparison

Figure 4.5: Ground truth data magnitude and the error metric

4

DISCUSSION OF RESULTS

The difference between Experiments #5.7-#5.12, and #6.1-#6.6 is solely a choice of the quantity evaluated in the loss term. The contrast between #5.12 and #6.6 shows that by evaluating on position, the GNS model passes the shape tolerance and performs well on the benchmark. Further, Figure 4.4b shows stable acceleration values are computed, even when using unnormalized data. This is supportive numerical evidence that the instability found when evaluating with unnormalized features and targets may be resolved by evaluating the loss quantity on position.

Figure 4.4a shows only a single experiment accurately meets the benchmark prediction. The intention of this numerical study is to see if evaluating the loss on position can help break through this barrier found when reducing the time step. The results show this barrier continues to persist.

The cause of this is an artifact of the magnitude in the unscaled data. Figure 4.5c shows that as the time step reduces, the one step position MSE reduces. When position is used in the L_2 norm, this loss magnitude is of the order of magnitude 10^{-12}, as conveyed by the one step position MSE in Table 4.4. This minute magnitude hinders the training of the neural networks' weights and biases.

Experiments #6.1 and #6.2 show the limiting case of this, at the smallest time steps. In this case the results show a perfectly stagnant disk with out any movement. The hindrance of network training from minute loss magnitudes yields an approximation function which produces static results.

Figure 4.5c shows a clear increase in the position MSE, and decrease in the acceleration MSE as the time step is increased. This is caused by the MSE value being dependent on the absolute magnitude of the value it evaluates.

As expected, the acceleration value remains the same regardless of the time step, as shown in Figure 4.5b. However, as the time step decreases, there is more noise in the acceleration data, as shown in Figure 4.5a. This translates to very high acceleration MSE values.

As the time step increases, the displacement between each time step increases, as shown in Figure 4.5b. This is reflected in a greater magnitude seen in the position MSE values. Figure 4.5a illustrates the key advantage of evaluating on position, that the standard deviation in displacement $\sigma(\Delta x)$ decreases with time.

In conclusion, evaluating the loss with position shows the ability for the GNS model to produce a noiseless response when using unnormalized features and targets. However, there continues to persist an inability for the model to make accurate predictions as the time step reduces.

4.3. CONCLUSION

Investigation of the GNS model and motivated alterations show the information limitations of the error metric magnitudes, depict the training hindrance caused by noise and showcase that evaluating on position allows unnormalized features and targets to be used in predicting noiseless results at a coarse time step. Sanchez-Gonzalez et al. [85] motivate the use of normalized features and targets, which require dataset statistics to normalize. This chapter shows the ability to do away with these scalar statistics. Instead, an alternative to use unnormalized features and targets, continue to predict acceleration, but instead compute the loss on position.

5

DISCUSSION

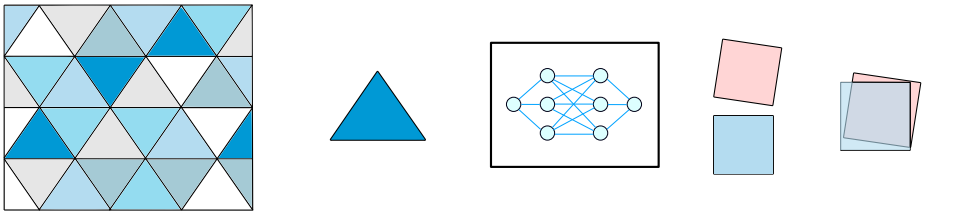


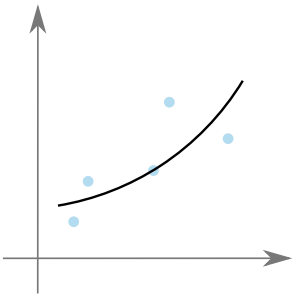
Figure 5.1: System visualization

The focus of this discussion is to wrap together Chapters 2, 3 and 4 in identifying new knowledge for literature. This is broken into: (*i*) limitations of the loss term, (*ii*) utility of the consistency, stability and convergence properties pioneered by the finite differences community, (*iii*) the numerical evidence of the loss quantity on stability when using unnormalized features and targets. This chapter concludes with future research suggestions.

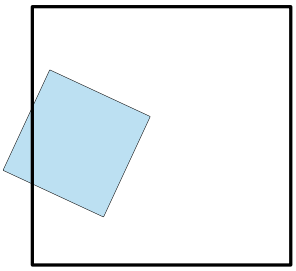
Figure 5.1 gives a conceptual abstraction of the context explored in this thesis project[1]. The identified benchmark simulations (domain) are expressed as feature inputs (signals) to the ML model (function) which is evaluated on a set goal (loss).

Chapter 2 explored the current practice in snow simulations, and the promising capabilities GNNs in the ML community offered. The key advantage of GNNs over both the conventional DEM and MPM models is the ability to make use of both global and local information in making local updates. However, there is a clear lack of physical basis

[1] This figure is derived from the GDL framework [126].



(a) Curve fitting



(b) Exceeding boundary

Figure 5.2: Contrasting the intentions of minimizing error and assuring basic physics

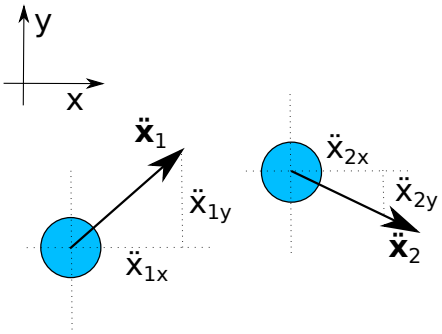
Table 5.1: Current models in literature and their loss quantity.

Model	**Loss quantity**
GNS [85]	Acceleration
C-GNS [90]	Velocity
MeshGraphNets [103]	Position or momentum
EGNNs [109]	Position
DPI-Nets [82]	Position or Velocity
MultiScale MeshGraphNets [106]	Position or momentum
Hamiltonian GNNs [132]	Position & momenta
Lagrangian Neural Networks [133]	Acceleration

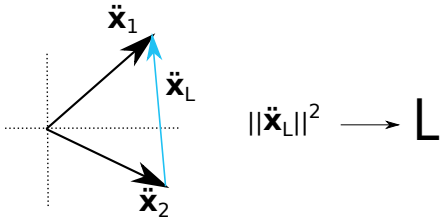
knowledge covered up when quantifying these GNN models on single scalar error metrics. Figure 5.2 highlights this difference between fitting data and meeting basic physics, such as boundary conditions. By definition, the MSE value quantifies how well an approximation function fits the test data. It does not inform of the underlying physics. This conclusion yielded the thesis research questions.

The true power of Chapter 3 is that reimagining this error metric by inspiration of the patch test allows one to evaluate a state of the art GNN model on core properties any numerical method ought to meet. In Chapter 3, the GNS model is simply used as a means to demonstrate this research idea. The importance of this for literature is it sheds light on the fact that GNNs to learn complex system apparently skipped the basics. To the knowledge of the author, the first application of GNNs to learn physical simulations jumped straight to learning an n-body dynamic system [81]. From here, GNNs were applied to continually complex systems [82, 83, 85, 103].

Chapter 3 showed that a reduction in the MSE magnitude does not affirm the underlying physics are improved, or if the underlying physics are even met for a small MSE magnitude. To improve on this limitation, first principles of consistency, stability and convergence form the bedrock to evaluate a GNN model in learning physics simulations. Further, through benchmark tests, they allow research to evaluate these GNN models on the physical laws of interest. Nevertheless, it answers the 1^{st} research question. By eval-



(a) Example accelerating particles



(b) Loss term visualization

Figure 5.3: Vector differences and its compression to the loss term

uating the GNN model with curated benchmarks, it can be shown if the model is, or is not, learning the underlying physics. Whether the physical laws of interest are learned is dependent on the problem formulation.

Chapter 4 narrows the focus to looking at the performance of the GNS model alone. Currently in literature there exists a spread of physical quantities which are used in the error metric, as shown in Table 5.1. There is also an apparent use of normalizing feature and targets sometimes, and other times not [85, 109]. Figure 5.3 visualizes the fact that this scalar loss value is in reality the expression of a vector difference.

The first numerical finding of Chapter 4 is that normalizing the features and targets results in a noise reduction for the predictions made by the GNS model. However, it also found this noise can be stabilized by choosing to evaluate the loss on position, instead of acceleration. These finding are made possible by implementation of the benchmark tests, and can not be seen in the single scalar loss metrics alone. They also answer the 2^{nd} research question set at the end of Chapter 2. This GNN model is sensitive to its inputs.

5.1. FUTURE RESEARCH

The popularity of the GNS model has inspired many papers, and this MSc thesis. Through work like this, this field can continue to grow. To nurture additional research, the following directions are proposed.

- In drawing another parallel with the finite difference method is the *Equivalence Theorem* [110]:

  > For a finite approximation function $F(x,d)$ which meets the consistency condition, stability is a necessary and sufficient condition for $F(x,d)$ to be a convergent approximation.

  This entails a relationship between consistency, stability and convergence exists. The work of Chapter 3 showed that for the rolling disk benchmark, where consis-

tency and stability are both met. In contrast, for the elastic bodies setup, neither consistency nor stability are met. The relationship for GNNs, if it exists, appears an open research direction.

- There is a clear inability to impose boundary conditions on this GNN model, and others in literature. For example, all models listed in Table 5.1. However, there is a known ability to connect spatial graphs with temporal graphs thorough the Kronecker graph product [134, 135]. This same connection can potentially be made, except with a graph representing boundary conditions, and the conventional graph representing the problem domain.
- The use of absolute error metrics appears to be misleading, especially when normalizing to different values. The use of relative error metrics, such as the r^2 value or mean absolute percentage error (MAPE) may give better insight when quantifying model performance on these single scalar error metrics.
- A clear barrier faced by the GNS model in all time convergence studies is a minimum time step which the model is able to approximate the rolling disk benchmark. The noise attributed to quantities derived though finite difference with reducing the time is believed to be the cause of this. Evaluating on position, a quantity not derived through finite differences, appears to aid this. However, the minute magnitude when using unnormalized values is believed to hinder network training.
- Section 3.1 highlights the inability of the GNS model to capture the rotational domain symmetry. This continues to hinder the model to this day [128]. Reducing the dimension of the dataset to a size for the neural network to learn in a tangible amount of training steps is vital to the efficacy of this method. Ongoing work shows this importance of including all domain symmetries [136].

6

CONCLUSION

The last decade introduced MPM to the snow science community to reduce computational costs. Simultaneously, the ML field progressed into learning complex physical simulations. GNN's efficacy to leverage physical domain symmetries makes it the current optimal ML model in this application.

The GNN model allows two key advantages to MPM: (*i*) Use of both local and global information in making local updates, (*ii*) directly implementing data to bias structure in the learned function. However, there are shortcomings in the GNN field which need to be addressed.

This thesis focuses on extending beyond fitting data, instead evaluating if boundary conditions, initial conditions and conservation laws are met. The main contributions of this work are: (*i*) through curated benchmark tests, may physical phenomena be isolated to determine if the GNN model is truly learning the underlying physics (*ii*) these benchmark tests allow a GNN model to be investigated on core properties any numerical method should meet, (*iii*) training the GNS model on unnormalized data can maintain noiseless predictions when evaluating the loss on position, (*iv*) a small loss value does not affirm physical quantities of interest are met. Future research directions are proposed to continue developing this numerical method such that it can truly be applied in the snow science community.

REFERENCES

[1] W. Bragg, *The crystal structure of ice,* Proceedings of the Physical Society of London **34**, 98 (1921).

[2] W. A. Bentley and W. J. Humphreys, *Snow crystals* (Courier Corporation, 2013).

[3] U. Nakaya and Y. Sekido, *General classification of snow crystals and their frequency of occurrence,* **1**, 243 (1936).

[4] H. Bader and D. Kuroiwa, *The physics and mechanics of snow as a material,* Cold Regions Research and Engineering Laboratory (US) (1962).

[5] M. Mellor, *A review of basic snow mechanics* (US Army Cold Regions Research and Engineering Laboratory Hanover, NH, 1974).

[6] F. Nicot, *Constitutive modelling of snow as a cohesive-granular material,* Granular Matter **6**, 47 (2004).

[7] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle, *A material point method for snow simulation,* ACM Trans. Graph. **32** (2013), 10.1145/2461912.2461948.

[8] J. Gaume, T. Gast, J. Teran, A. van Herwijnen, and C. Jiang, *Dynamic anticrack propagation in snow,* Nature Communications **9**, 3047 (2018).

[9] N. Mitarai and F. Nori, *Wet granular materials,* Advances in Physics **55**, 1 (2006), https://doi.org/10.1080/00018730600626065 .

[10] S. Li and W. K. Liu, *Meshfree particle methods* (Springer Science & Business Media, 2007).

[11] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl, *Meshless methods: An overview and recent developments,* Computer Methods in Applied Mechanics and Engineering **139**, 3 (1996).

[12] G.-R. Liu, *Meshfree methods: moving beyond the finite element method* (CRC press, 2009).

[13] D. Sulsky, Z. Chen, and H. Schreyer, *A particle method for history-dependent materials,* Computer Methods in Applied Mechanics and Engineering **118**, 179 (1994).

[14] M. W. Evans, F. H. Harlow, and E. Bromberg, *The particle-in-cell method for hydrodynamic calculations,* Tech. Rep. (1957).

[15] J. Fern, A. Rohe, K. Soga, and E. Alonso, *The material point method for geotechnical engineering: a practical guide* (CRC Press, 2019).

[16] W. T. Sołowski, M. Berzins, W. M. Coombs, J. E. Guilkey, M. Möller, Q. A. Tran, T. Adibaskoro, S. Seyedan, R. Tielen, and K. Soga, *Chapter two - material point method: Overview and challenges ahead,* (Elsevier, 2021) pp. 113–204.

[17] D. Sulsky, S.-J. Zhou, and H. L. Schreyer, *Application of a particle-in-cell method to solid mechanics,* Computer Physics Communications **87**, 236 (1995), particle Simulation Methods.

[18] V. P. Nguyen, A. de Vaucorbeil, and S. Bordas, *The Material Point Method: Theory, Implementations and Applications* (Springer Nature, 2023).

[19] J. Gaume, L. Blatny, X. Li, and B. Trottet, *Modeling snow and avalanches with the Material Point Method and finite strain elastoplasticity,* Tech. Rep. (The Alliance of Laboratories in Europe for Education, Research and Technology, 2020).

[20] J. Gaume, A. van Herwijnen, T. Gast, J. Teran, and C. Jiang, *Investigating the release and flow of snow avalanches at the slope-scale using a unified model based on the material point method,* Cold Regions Science and Technology **168**, 102847 (2019).

[21] X. Li, B. Sovilla, C. Jiang, and J. Gaume, *Three-dimensional and real-scale modeling of flow regimes in dense snow avalanches,* Landslides **18**, 3393 (2021).

[22] K. Kumar, K. Soga, and J.-Y. Delenne, *Multi-scale modelling of granular avalanches,* (American Institute of Physics, 2013) pp. 1250–1253.

[23] M. Jebahi, D. André, I. Terreros, and I. Iordanoff, *Discrete element method to model 3D continuous materials* (John Wiley & Sons, 2015).

[24] P. A. Cundall and O. D. L. Strack, *A discrete numerical model for granular assemblies,* Géotechnique **29**, 47 (1979), https://doi.org/10.1680/geot.1979.29.1.47 .

[25] Y. Guo and J. S. Curtis, *Discrete element method simulations for complex granular flows,* Annual Review of Fluid Mechanics **47**, 21 (2015), https://doi.org/10.1146/annurev-fluid-010814-014644 .

[26] H. Kruggel-Emden, E. Simsek, S. Rickelt, S. Wirtz, and V. Scherer, *Review and extension of normal force models for the discrete element method,* Powder Technology **171**, 157 (2007).

[27] P. Wriggers and B. Avci, *Discrete element methods: Basics and applications in engineering,* in *Modeling in Engineering Using Innovative Numerical Methods for Solids and Fluids*, edited by L. De Lorenzis and A. Düster (Springer International Publishing, Cham, 2020) pp. 1–30.

[28] G. Bobillier, B. Bergfeld, A. Capelli, J. Dual, J. Gaume, A. van Herwijnen, and J. Schweizer, *Micromechanical modeling of snow failure,* The Cryosphere **14**, 39 (2020).

[29] J. Gaume, A. van Herwijnen, G. Chambon, K. W. Birkeland, and J. Schweizer, *Modeling of crack propagation in weak snowpack layers using the discrete element method,* The Cryosphere **9**, 1915 (2015).

[30] P. Hagenmuller, G. Chambon, and M. Naaim, *Microstructure-based modeling of snow mechanics: a discrete element approach,* The Cryosphere **9**, 1969 (2015).

[31] K. Kumar, K. Soga, J.-Y. Delenne, and F. Radjai, *Modelling transient dynamics of granular slopes: Mpm and dem,* Procedia Engineering **175**, 94 (2017), proceedings of the 1st International Conference on the Material Point Method (MPM 2017).

[32] R. A. Gingold and J. J. Monaghan, *Smoothed particle hydrodynamics: theory and application to non-spherical stars,* Monthly notices of the royal astronomical society **181**, 375 (1977).

[33] L. B. Lucy, *A numerical approach to the testing of the fission hypothesis.* aj **82**, 1013 (1977).

[34] M. B. Liu and G. R. Liu, *Smoothed particle hydrodynamics (sph): an overview and recent developments,* Archives of Computational Methods in Engineering **17**, 25 (2010).

[35] A. M. Abdelrazek, I. Kimura, and Y. Shimizu, *Numerical simulation of snow avalanches as a bingham fluid flow using sph method,* River Flow , 1581 (2014).

[36] Ø. E. Krog and A. C. Elster, *Fast gpu-based fluid simulations using sph,* (Springer, 2010) pp. 98–109.

[37] C. Gissler, A. Henne, S. Band, A. Peer, and M. Teschner, *An implicit compressible sph solver for snow simulation,* ACM Trans. Graph. **39** (2020), 10.1145/3386569.3392431.

[38] Ø. E. Krog, *Gpu-based real-time snow avalanche simulations,* Master's thesis (2010).

[39] G.-R. Liu and M. B. Liu, *Smoothed particle hydrodynamics: a meshfree particle method* (World scientific, 2003).

[40] S. Koshizuka and Y. Oka, *Moving-particle semi-implicit method for fragmentation of incompressible fluid,* Nuclear science and engineering **123**, 421 (1996).

[41] S. Koshizuka, A. Nobe, and Y. Oka, *Numerical analysis of breaking waves using the moving particle semi-implicit method,* International Journal for Numerical Methods in Fluids **26**, 751 (1998).

[42] S. Koshizuka, K. Shibata, M. Kondo, and T. Matsunaga, *Moving particle semi-implicit method: a meshfree particle method for fluid dynamics* (Academic Press, 2018).

[43] H. Kato, M. Otsuki, Y. Saito, Y. Shimizu, and I. Kimura, *Refinement of mps method for practical application on large scale snow avalanches,* Journal of Japan Society of Civil Engineers, Ser. B1 (Hydraulic Engineering) **67**, I (2011).

[44] T. Otsuka, Y. Shimizu, I. Kimura, M. Otsuki, and Y. Saito, *Fundamental studies on applications of mps method for computing snow avalanches,* in *International Snow Science Workshop, Davos* (2009).

[45] Y. Tao and S. Koshizuka, *Validation of a bingham snow model by the mps method,* in *2nd International Workshop on Structural Health Monitoring for Railway System (IWSHM-RS 2018)* (2018).

[46] S. Nohara, H. Suenaga, and K. Nakamura, *Large deformation simulations of geo-materials using moving particle semi-implicit method,* Journal of Rock Mechanics and Geotechnical Engineering **10**, 1122 (2018).

[47] M. Stoffel, *Numerical modelling of snow using finite elements*, 291 (vdf Hochschulverlag AG, 2006).

[48] E. Podolskiy, G. Chambon, M. Naaim, and J. Gaume, *A review of finite-element modelling in snow mechanics,* Journal of Glaciology **59**, 1189–1201 (2013).

[49] D. L. Logan, *A first course in the finite element method* (Cengage Learning, 2016).

[50] G. Meschke, C. Liu, and H. A. Mang, *Large strain finite-element analysis of snow,* Journal of Engineering Mechanics **122**, 591 (1996).

[51] S. Shoop, K. Kestler, and R. Haehnel, *Finite Element Modeling of Tires on Snow2,* Tire Science and Technology **34**, 2 (2006).

[52] E. Seta, T. Kamegawa, and Y. Nakajima, *Prediction of Snow/Tire Interaction Using Explicit FEM and FVM,* Tire Science and Technology **31**, 173 (2003).

[53] S. Rajendran, *A technique to develop mesh-distortion immune finite elements,* Computer Methods in Applied Mechanics and Engineering **199**, 1044 (2010).

[54] P. Bartelt and M. Lehning, *A physical snowpack model for the swiss avalanche warning: Part i: numerical model,* Cold Regions Science and Technology **35**, 123 (2002).

[55] M. Lehning, P. Bartelt, B. Brown, C. Fierz, and P. Satyawali, *A physical snowpack model for the swiss avalanche warning: Part ii. snow microstructure,* Cold Regions Science and Technology **35**, 147 (2002).

[56] M. Lehning, P. Bartelt, B. Brown, and C. Fierz, *A physical snowpack model for the swiss avalanche warning: Part iii: meteorological forcing, thin layer formation and evaluation,* Cold Regions Science and Technology **35**, 169 (2002).

[57] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, *A view of the parallel computing landscape,* Commun. ACM **52**, 56–67 (2009).

[58] X. Wang, Y. Qiu, S. R. Slattery, Y. Fang, M. Li, S.-C. Zhu, Y. Zhu, M. Tang, D. Manocha, and C. Jiang, *A massively parallel and scalable multi-gpu material point method,* ACM Trans. Graph. **39** (2020), 10.1145/3386569.3392442.

[59] Y. Shigeto and M. Sakai, *Parallel computing of discrete element method on multi-core processors,* Particuology **9**, 398 (2011), multiscale Modeling and Simulation of Complex Particulate Systems.

[60] E. Yang, H. H. Bui, H. De Sterck, G. D. Nguyen, and A. Bouazza, *A scalable parallel computing sph framework for predictions of geophysical granular flows,* Computers and Geotechnics **121**, 103474 (2020).

[61] M. Ihmsen, N. Akinci, M. Becker, and M. Teschner, *A parallel sph implementation on multi-core cpus,* Computer Graphics Forum **30**, 99 (2011), https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2010.01832.x .

[62] M. B. Cil and K. A. Alshibli, *3d assessment of fracture of sand particles using discrete element method,* Géotechnique Letters **2**, 161 (2012), https://doi.org/10.1680/geolett.12.00024 .

[63] Z.-P. Chen, X. Zhang, K. Y. Sze, L. Kan, and X.-M. Qiu, *v-p material point method for weakly compressible problems,* Computers & Fluids **176**, 170 (2018).

[64] K. Schürholt, J. Kowalski, and H. Löwe, *Elements of future snowpack modeling – part 1: A physical instability arising from the nonlinear coupling of transport and phase changes,* The Cryosphere **16**, 903 (2022).

[65] A. Simson, H. Löwe, and J. Kowalski, *Elements of future snowpack modeling – part 2: A modular and extendable eulerian–lagrangian numerical scheme for coupled transport, phase changes and settling processes,* The Cryosphere **15**, 5423 (2021).

[66] Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang, *A moving least squares material point method with displacement discontinuity and two-way rigid body coupling,* ACM Trans. Graph. **37** (2018), 10.1145/3197517.3201293.

[67] I. Farmaga, P. Shmigelskyi, P. Spiewak, and L. Ciupinski, *Evaluation of computational complexity of finite element analysis,* in *2011 11th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)* (IEEE, 2011) pp. 213–214.

[68] S. Koshizuka, K. Shibata, M. Kondo, and T. Matsunaga, *Moving particle semi-implicit method: a meshfree particle method for fluid dynamics* (Academic Press, 2018).

[69] Q. Zhu, L. Hernquist, and Y. Li, *NUMERICAL CONVERGENCE IN SMOOTHED PARTICLE HYDRODYNAMICS,* The Astrophysical Journal **800**, 6 (2015).

[70] S. Bardenhagen, J. Brackbill, and D. Sulsky, *The material-point method for granular materials,* Computer Methods in Applied Mechanics and Engineering **187**, 529 (2000).

[71] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems* (" O'Reilly Media, Inc.", 2019).

[72] K. P. Murphy, *Probabilistic machine learning: an introduction* (MIT press, 2022).

[73] K. O'Shea and R. Nash, *An introduction to convolutional neural networks,* arXiv preprint arXiv:1511.08458 (2015).

[74] S. Albawi, T. A. Mohammed, and S. Al-Zawi, *Understanding of a convolutional neural network,* in *2017 International Conference on Engineering and Technology (ICET)* (2017) pp. 1–6.

[75] K. Stachenfeld, D. B. Fielding, D. Kochkov, M. Cranmer, T. Pfaff, J. Godwin, C. Cui, S. Ho, P. Battaglia, and A. Sanchez-Gonzalez, *Learned coarse models for efficient turbulence simulation,* (2021).

[76] S. Wiewel, M. Becher, and N. Thuerey, *Latent space physics: Towards learning the temporal evolution of fluid flow,* Computer Graphics Forum **38**, 71 (2019), https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13620 .

[77] B. Ummenhofer, L. Prantl, N. Thuerey, and V. Koltun, *Lagrangian fluid simulation with continuous convolutions,* in *International Conference on Learning Representations* (2020).

[78] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, *Machine learning on graphs: A model and comprehensive taxonomy,* arXiv preprint arXiv:2005.03675 , 1 (2020).

[79] M. Gori, G. Monfardini, and F. Scarselli, *A new model for learning in graph domains,* (2005) pp. 729–734.

[80] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, *The graph neural network model,* IEEE Transactions on Neural Networks **20**, 61 (2009).

[81] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and k. kavukcuoglu, *Interaction networks for learning about objects, relations and physics,* in *Advances in Neural Information Processing Systems*, Vol. 29, edited by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Curran Associates, Inc., 2016).

[82] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, *Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids,* CoRR **abs/1810.01566** (2018), 1810.01566 .

[83] Y. Li, J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake, *Propagation networks for model-based control under partial observation,* in *2019 International Conference on Robotics and Automation (ICRA)* (2019) pp. 1205–1211.

[84] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, *Relational inductive biases, deep learning, and graph networks,* CoRR **abs/1806.01261** (2018), 1806.01261 .

[85] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, *Learning to simulate complex physics with graph networks,* in *Proceedings of the 37th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 119, edited by H. D. III and A. Singh (PMLR, 2020) pp. 8459–8468.

[86] J. Klimesch, P. Holl, and N. Thuerey, *Simulating liquids with graph networks,* arXiv preprint arXiv:2203.07895 (2022).

[87] J. Han, W. Huang, H. Ma, J. Li, J. Tenenbaum, and C. Gan, *Learning physical dynamics with subequivariant graph neural networks,* Advances in Neural Information Processing Systems **35**, 26256 (2022).

[88] Z. Li and A. B. Farimani, *Graph neural network-accelerated lagrangian fluid simulation,* Computers & Graphics **103**, 201 (2022).

[89] S. Yang, X. He, and B. Zhu, *Learning physical constraints with neural projections,* in *Advances in Neural Information Processing Systems*, Vol. 33, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin (Curran Associates, Inc., 2020) pp. 5178–5189.

[90] Y. Rubanova, A. Sanchez-Gonzalez, T. Pfaff, and P. W. Battaglia, *Constraint-based graph network simulator,* CoRR **abs/2112.09161** (2021), 2112.09161 .

[91] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations,* CoRR **abs/1711.10561** (2017), 1711.10561 .

[92] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, *Physics-informed neural networks (pinns) for fluid mechanics: a review,* Acta Mechanica Sinica **37**, 1727 (2021).

[93] K. Shukla, M. Xu, N. Trask, and G. E. Karniadakis, *Scalable algorithms for physics-informed neural and graph networks,* Data-Centric Engineering **3**, e24 (2022).

[94] P. Kay and W. Kempton, *What is the sapir-whorf hypothesis?* American Anthropologist **86**, 65 (1984), https://anthrosource.onlinelibrary.wiley.com/doi/pdf/10.1525/aa.1984.86.1.02a00050 .

[95] M. Cranmer, A. Sanchez Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, *Discovering symbolic models from deep learning with inductive biases,* in *Advances in Neural Information Processing Systems*, Vol. 33, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin (Curran Associates, Inc., 2020) pp. 17429–17442.

[96] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, *A gentle introduction to graph neural networks,* Distill (2021), 10.23915/distill.00033, https://distill.pub/2021/gnn-intro.

[97] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, *Neural message passing for quantum chemistry,* in *Proceedings of the 34th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 70, edited by D. Precup and Y. W. Teh (PMLR, 2017) pp. 1263–1272.

[98] U. Alon and E. Yahav, *On the bottleneck of graph neural networks and its practical implications,* CoRR **abs/2006.05205** (2020), 2006.05205 .

[99] M. Lino, C. D. Cantwell, A. A. Bharath, and S. Fotiadis, *Simulating continuum mechanics with multi-scale graph neural networks,* CoRR **abs/2106.04900** (2021), 2106.04900 .

[100] W. Hamilton, Z. Ying, and J. Leskovec, *Inductive representation learning on large graphs,* in *Advances in Neural Information Processing Systems*, Vol. 30, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017).

[101] B. Rozemberczki, O. Kiss, and R. Sarkar, *Little ball of fur: A python library for graph sampling,* in *Proceedings of the 29th ACM International Conference on Information & amp; Knowledge Management*, CIKM '20 (Association for Computing Machinery, New York, NY, USA, 2020) p. 3133–3140.

[102] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, *Graphsaint: Graph sampling based inductive learning method,* CoRR **abs/1907.04931** (2019), 1907.04931 .

[103] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, *Learning mesh-based simulation with graph networks,* CoRR **abs/2010.03409** (2020), 2010.03409 .

[104] Z. Kolter, D. Abed Duvenaud, and M. Johnson, *Deep implicit layers - neural odes, deep equilibirum models, and beyond,* NeurIPS 2020 tutorial (2020).

[105] K. Martinkus, A. Lucchi, and N. Perraudin, *Scalable graph networks for particle simulations,* Proceedings of the AAAI Conference on Artificial Intelligence **35**, 8912 (2021).

[106] M. Fortunato, T. Pfaff, P. Wirnsberger, A. Pritzel, and P. Battaglia, *Multiscale meshgraphnets,* in *ICML 2022 2nd AI for Science Workshop*.

[107] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, *OGB-LSC: A large-scale challenge for machine learning on graphs,* CoRR **abs/2103.09430** (2021), 2103.09430 .

[108] M. Naser and A. H. Alavi, *Error metrics and performance fitness indicators for artificial intelligence and machine learning in engineering and sciences,* Architecture, Structures and Construction , 1 (2021).

[109] V. G. Satorras, E. Hoogeboom, and M. Welling, *E (n) equivariant graph neural networks,* in *International conference on machine learning* (PMLR, 2021) pp. 9323–9332.

[110] P. D. Lax and R. D. Richtmyer, *Survey of the stability of linear finite difference equations,* Communications on pure and applied mathematics **9**, 267 (1956).

[111] R. Courant, K. Friedrichs, and H. Lewy, *On the partial difference equations of mathematical physics,* IBM journal of Research and Development **11**, 215 (1967).

[112] A. K. Aziz, *The mathematical foundations of the finite element method with applications to partial differential equations* (Academic Press, 2014).

[113] O. Zienkiewicz and R. Taylor, *The finite element patch test revisited a computer test for convergence, validation and error estimates,* Computer Methods in Applied Mechanics and Engineering **149**, 223 (1997), containing papers presented at the Symposium on Advances in Computational Mechanics.

[114] J. Dolbow and T. Belytschko, *Numerical integration of the galerkin weak form in meshfree methods,* Computational Mechanics **23**, 219 (1999).

[115] G. R. Liu and Y. T. Gu, *A meshfree method: meshfree weak-strong (mws) form method, for 2-d solids,* Computational Mechanics **33**, 2 (2003).

[116] Q. Duan, X. Li, H. Zhang, and T. Belytschko, *Second-order accurate derivatives and integration schemes for meshfree methods,* Int. J. Numer. Meth. Engng **92**, 399 (2012).

[117] T. Belytschko, Y. Y. Lu, and L. Gu, *Element-free galerkin methods,* International Journal for Numerical Methods in Engineering **37**, 229 (1994), https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620370205 .

[118] D. Burgess, D. Sulsky, and J. Brackbill, *Mass matrix formulation of the flip particle-in-cell method,* Journal of Computational Physics **103**, 1 (1992).

[119] D. Sulsky and A. Kaul, *Implicit dynamics in the material-point method,* Computer Methods in Applied Mechanics and Engineering **193**, 1137 (2004), meshfree Methods: Recent Advances and New Applications.

[120] W. M. Coombs, T. J. Charlton, M. Cortis, and C. E. Augarde, *Overcoming volumetric locking in material point methods,* Computer Methods in Applied Mechanics and Engineering **333**, 1 (2018).

[121] J. Brackbill and H. Ruppel, *Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions,* Journal of Computational Physics **65**, 314 (1986).

[122] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin, *The affine particle-in-cell method,* ACM Trans. Graph. **34** (2015), 10.1145/2766996.

[123] W. K. Liu, S. Jun, and Y. F. Zhang, *Reproducing kernel particle methods,* International Journal for Numerical Methods in Fluids **20**, 1081 (1995), https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.1650200824 .

[124] A. Stomakhin, R. Howes, C. A. Schroeder, and J. M. Teran, *Energetically consistent invertible elasticity.* in *Symposium on Computer Animation*, Vol. 1 (2012).

[125] G. Cybenko, *Approximation by superpositions of a sigmoidal function,* Mathematics of control, signals and systems **2**, 303 (1989).

[126] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges,* CoRR **abs/2104.13478** (2021), 2104.13478 .

[127] E. Noether, *Invariant variation problems,* Transport Theory and Statistical Physics **1**, 186 (1971).

[128] K. R. Allen, T. L. Guevara, Y. Rubanova, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, and T. Pfaff, *Graph network simulators can learn discontinuous, rigid contact dynamics,* in *Proceedings of The 6th Conference on Robot Learning*, Proceedings of Machine Learning Research, Vol. 205, edited by K. Liu, D. Kulic, and J. Ichnowski (PMLR, 2023) pp. 1157–1167.

[129] M. Tupek, J. Koester, and M. Mosby, *A momentum preserving frictional contact algorithm based on affine particle-in-cell grid transfers,* (2021), arXiv:2108.02259 [cs.CE] .

[130] T. Belytschko, W. K. Liu, B. Moran, and K. Elkhodary, *Nonlinear finite elements for continua and structures* (John wiley & sons, 2014).

[131] S. Villar, D. W. Hogg, K. Storey-Fisher, W. Yao, and B. Blum-Smith, *Scalars are universal: Equivariant machine learning, structured like classical physics,* Advances in Neural Information Processing Systems **34**, 28848 (2021).

[132] A. Sanchez-Gonzalez, V. Bapst, K. Cranmer, and P. Battaglia, *Hamiltonian graph networks with ode integrators,* arXiv preprint arXiv:1909.12790 (2019).

[133] M. D. Cranmer, S. Greydanus, S. Hoyer, P. W. Battaglia, D. N. Spergel, and S. Ho, *Lagrangian neural networks,* CoRR **abs/2003.04630** (2020), 2003.04630 .

[134] P. M. Weichsel, *The kronecker product of graphs,* Proceedings of the American mathematical society **13**, 47 (1962).

[135] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, *Kronecker graphs: an approach to modeling networks.* Journal of Machine Learning Research **11** (2010).

[136] J. Han, Y. Rong, T. Xu, and W. Huang, *Geometrically equivariant graph neural networks: A survey,* arXiv preprint arXiv:2202.07230 (2022).

[137] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization,* arXiv preprint arXiv:1412.6980 (2014).

[138] G. Klár, T. Gast, A. Pradhana, C. Fu, C. Schroeder, C. Jiang, and J. Teran, *Drucker-prager elastoplasticity for sand animation,* ACM Trans. Graph. **35** (2016), 10.1145/2897824.2925906.

[139] E. H. Lee, *Elastic-Plastic Deformation at Finite Strains,* Journal of Applied Mechanics **36**, 1 (1969).

[140] L. Blatny, H. Löwe, S. Wang, and J. Gaume, *Computational micromechanics of porous brittle solids,* Computers and Geotechnics **140**, 104284 (2021).

[141] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*, Vol. 37 (Springer Science & Business Media, 2010).

[142] R. Fedkiw, *Convergence, consistency and stability, von neumann analysis, and the lax-richtmyer equivalence theorem,* https://web.stanford.edu/class/cme306/ (2009).

[143] B. Chartres and R. Stepleman, *A general theory of convergence for numerical methods,* SIAM Journal on Numerical Analysis **9**, 476 (1972).

[144] B. Ribeiro, *Go-no-go report: Alternative to the Finite Element Method with graph Neural Networks*, Tech. Rep. (Delft University of Technology, 2023).

[145] J. Hanc, S. Tuleja, and M. Hancova, *Symmetries and conservation laws: Consequences of Noether's theorem,* American Journal of Physics **72**, 428 (2004), https://pubs.aip.org/aapt/ajp/article-pdf/72/4/428/7531083/428_1_online.pdf .

[146] S. Cresseri, C. Jommi, *et al.*, *Snow as an elastic viscoplastic bonded continuum: a modelling approach,* Italian Geotechnical J **4**, 43 (2005).

[147] B. Salm, *On the rheological behavior of snow under high stresses,* Contributions from the institute of Low Temperature Science **23**, 1 (1971).

[148] J. Desrues, F. Darve, E. Flavigny, J. Navarre, and A. Taillefer, *An incremental formulation of constitutive equations for deposited snow,* Journal of Glaciology **25**, 289 (1980).

[149] C. R. Meyer, K. M. Keegan, I. Baker, and R. L. Hawley, *A model for french-press experiments of dry snow compaction,* The Cryosphere **14**, 1449 (2020).

[150] A. LeBaron and D. Miller, *An energy-based microstructural constitutive model for fracture in snow,* in *Proceedings, International Snow Science Workshop, Banff, Alberta, Canada*, Vol. 29 (2014) pp. 134–138.

[151] B. W. Kabore, B. Peters, M. Michael, and F. Nicot, *A discrete element framework for modeling the mechanical behaviour of snow—part i: Mechanical behaviour and numerical model,* Granular Matter **23**, 1 (2021).

[152] K. R. Allen, Y. Rubanova, T. Lopez-Guevara, W. Whitney, A. Sanchez-Gonzalez, P. Battaglia, and T. Pfaff, *Learning rigid dynamics with face interaction graph networks,* arXiv preprint arXiv:2212.03574 (2022).

A

NEURAL NETWORK FUNDAMENTALS

A brief detail is given here on the basics of Nerual Networks. The reader is referenced to Murphy [72] for additional reading. The most basic form of Neural Networks is linear regression. Linear regression is a type of curve fitting used when the outputs are considered a linear combination of inputs. Thus, the expected value is represented according to Equation A.1 ([72]pg.365)

$$\mathbb{E}[y|\mathbf{x}] = \mathbf{w}^{\mathbf{T}}\mathbf{x} \tag{A.1}$$

Where $\mathbf{x}$ are inputs (features), $\mathbf{w}$ represent weights and y is the output.

The weights $\mathbf{w}$ are trained values which as the model is trained, allowed to see data, these values are systematically adjusted such that the error between the trained model and observed data points is minimized. This difference is a type of loss function. A model representing linear regression is taken as Equation A.2 ([72]pg.365):

$$p(y|\mathbf{x},\boldsymbol{\theta}) = \mathcal{N}(y|w_0 + \mathbf{w}^{\mathbf{T}}\mathbf{x}, \sigma^2) \tag{A.2}$$

Where $\boldsymbol{\theta} = (w_0, \mathbf{w}, \sigma^2)$ are all model parameters.

Now in connecting this linear regression model to the broader neural networks realm is the notion of flexibility. As mentioned above, this model relies on the assumption that the output(s) are linear combinations of the inputs. As this is not true for all systems, three edits are made to allow for flexibility. First, a feature transformation replacing $\mathbf{x}$ with the *basis function* $\phi(\mathbf{x})$, resulting in Equation A.3 ([72]pg.419):

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathbf{W}\phi(\mathbf{x}) + \mathbf{b} \tag{A.3}$$

The second step, an intuitive addition to amplify flexibility is bestowing the *feature extractor* ϕ with its own parameters $\boldsymbol{\theta}_2$:

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathbf{W}\phi(\mathbf{x};\boldsymbol{\theta}_2) + b \tag{A.4}$$

Where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ and $\boldsymbol{\theta}_1 = (\mathbf{W}, \mathbf{b})$.

The third step is repeating this process such that the surrogate model builds in complexity. Doing this L times yields Equation A.5 ([72]pg.419):

$$f(\mathbf{x};\boldsymbol{\theta}) = f_L(f_{L-1}(...(f_1(\mathbf{x}))...)) \tag{A.5}$$

With the network defined, an optimization problem is formed to tune its parameters $\boldsymbol{\theta} \in \Theta$ such that they reach a local minimum $\boldsymbol{\theta}^*$. This is done on a bounded scalar output loss function $\mathscr{L}$:

$$\boldsymbol{\theta}^* \in \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}}\, \mathscr{L}(\boldsymbol{\theta}) \tag{A.6}$$

In this work, the Adam optimizer [137] is then used in solving this optimization problem.

B

Damage Based Constiutive Model for Snow

Snow constitutive modeling is known as an open research question in literature [7]. The intuitive approach to model snow dynamics is discretizing the material domain into a Lagrangian frame, such as in DEM. In this frame the points represent individual snow grains, and the constitutive model depicts the interaction of said grains [6]. With the prominent paper of Stomakhin et al. [7], this fully Lagrangian perspective changed to the hybrid Eulerian-Lagrangian frame in MPM. Here, the constitutive model uses particle deformations to update the grid nodal velocities [138].

Lee's [139] multiplicative decomposition of a continuum's deformation gradient into elastic and plastic components is abundantly used in snow constitutive modeling to model finite strains [7, 8, 140]. The deformation gradient $\mathbf{F}$ is a mapping ϕ from the undeformed $\mathbf{X}$ to deformed $\mathbf{x}$ configuration via Equation B.1, and is split into elastic and plastic components according to Equation B.2. The method by how $\mathbf{F}^E$ and $\mathbf{F}^P$ are defined varies per model [7, 8, 140].

$$\mathbf{F} = \frac{\partial \phi}{\partial \mathbf{X}} \tag{B.1}$$

$$\mathbf{F} = \mathbf{F}^E \mathbf{F}^P \tag{B.2}$$

The stored deformation energy density ψ is defined by a variant of the St. Venant-Kirchhoff, including the Hencky strain $\ln \mathbf{\Sigma}$ from the singular value decomposition of $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. The gradient of this deformation energy density ψ (Equation B.3) defines the force $\mathbf{f}_i$ on a node (Equation B.4).

$$\psi(\mathbf{F}) = \mu \operatorname{tr}((\ln \boldsymbol{\Sigma})^2) + \frac{1}{2}\lambda(\operatorname{tr}(\ln \boldsymbol{\Sigma}))^2 \tag{B.3}$$

$$\mu = \frac{E}{2(1+\nu)}, \lambda = \frac{E}{(1+\nu)(1-2\nu)}$$

$$\mathbf{f}_i = -\frac{\partial \psi}{\partial \mathbf{x}_i} \tag{B.4}$$

The method by which $\mathbf{F}$ is decomposed in literature varies based on the type of yield surface used. A comparison of prominent MPM implementations is shown in Table B.1. The work of Stomakhin et al. [7] resembles the MPM sample code produced by Hu et al. [66], the current *ground truth* model.

Table B.1: Comparison of constitutive snow models in literature

	Stomakhin et al. [7]	**Gaume et al.** [8]	**Blatny et al.** [140]
Yield surface	Threshold	Cohesive Cam Clay	Drucker-Prager
Harden/ soften	Harden	Soften	Soften
Return mapping	No	Yes	Yes
Associative	-	Associative	Non-associative
Strain tensor	Hencky	Hencky	Hencky
Elastic Energy	St. Venant-Kirchhoff	St. Venant-Kirchhoff	St. Venant-Kirchhoff

The MPM key component of Stomakhin et al. [7], implemented in the sample MPM code of Hu et al. [66], is the definition of a threshold of acceptable values for the elastic component of the deformation gradient $\mathbf{F}^E$. Specifically, the values of $\boldsymbol{\Sigma}$ in $\mathbf{F} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$, must be within the critical stretch threshold (Figure B.1), and are kept inside via a clamping operation, Equation B.5. This constitutive model is not fully physically representative, as it lacks return mapping and an acceptable yield surface, but does yield physically realistic results, which is the intention of the authors [7].

$$1-\theta_c \leqslant \boldsymbol{\Sigma}_{ii} \leqslant 1+\theta_s \tag{B.5}$$

Additionally, this model utilizes the affine transfers between nodes and particles pioneered by Jiang et al. [122] to mitigate momentum dissipation effects normally affiliated with the hybrid Eulerian-Lagrangian Particle in Cell (PIC) method. The affine matrix is defined on the Eulerian grid via Equation B.6. This affine momentum is combined with the particle force contribution, Equation B.7, to act as the dynamic quantity used to update the nodal momentum.

$$\mathbf{C}_p^{n+1} = \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} \tag{B.6}$$

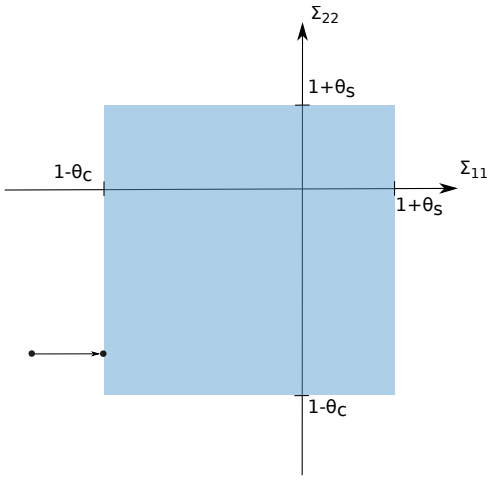


Figure B.1: Threshold mapping diagram

$$\mathbf{Q}_p = \Delta t V_p^0 M_p^{-1} \frac{\partial \psi}{\partial \mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{n\,T} + m_p \mathbf{C}_p \tag{B.7}$$

$$M_p = \frac{1}{4}\Delta x^2$$

Hardening of the Lamé parameters is implemented using the magnitude of the plastic deformation gradient, Equation B.8. The definition of $\mathbf{F}^P$ in Hu et al. [66] is taken as the ratio of the determinants for $\mathbf{F}$ and $\mathbf{F}^E$, where $\mathbf{F}^E$ is defined as the clamped version of $\mathbf{F}$[1].

$$\mu = \mu_0 e^{\xi(1-J_P)},\ \lambda = \lambda_0 e^{\xi(1-J_P)} \tag{B.8}$$

$$\xi \text{ - hardening coefficient}$$

$$J_P^{t+1} = J_P^t \frac{\det \mathbf{F}}{\det \mathbf{F}^E}$$

The updated hardened Lamé parameters are implemented in computing the energy density ψ, and further there after its gradient in determining the particle force (Equation B.4) in updating a particle to its next time step. See Algorithm B.1 for a sequential description of the MPM implementation.

[1] There is a clamping operation applied to J_P^{t+1}, such that the quantities value must lie between 0.60 and 20.0

Algorithm B.1 MPM88 Algorithm

Require: m_p, V_p, $\mathbf{x}_p$, $\mathbf{v}_p$, $\mathbf{C}_p$, $\mathbf{F}_p$
  for t in time **do**
    for p in Particles **do** ▷ Particle-to-Grid
      $\mu \leftarrow \mu_0 \, e^{\xi(1-J_p)}$
      $\lambda \leftarrow \lambda_0 \, e^{\xi(1-J_p)}$
      Polar decomposition: $\mathbf{R}$, $\mathbf{S} \leftarrow \mathbf{F}_p$
      $D_{\text{inv}} \leftarrow 4\,dx^2_{\text{inv}}$
      Particle Force: $PF \leftarrow 2\mu(\mathbf{F}_p - \mathbf{R})\mathbf{F}^T + \lambda(J-1)J$
      $\sigma \leftarrow -dt\,V\,D_{inv}\,PF$
      affine = $\sigma + m_p\mathbf{C}_p$
    end for
    Gather $\mathbf{v}_p$ & $\mathbf{v}_p \cdot m_p$ on nodes via a quadratic B-spline kernel.
    Apply velocity boundary conditions
    for p in Particles **do** ▷ Grid-to-Particle
      Use kernel to map $\mathbf{v}_p$ & $\mathbf{C}_p$ from nodes to particle p.
      $\mathbf{x}_p \leftarrow dt\,\mathbf{v}_p$
      $\mathbf{F}_p^{uncl} \leftarrow (\mathbf{I} + dt\,\mathbf{C}_p)\mathbf{F}_p$
      $J_p^{uncl} \leftarrow \det(\mathbf{F}_p^{uncl})$
      Singular Value Decomposition: $\mathbf{U\Sigma V} \leftarrow F$
      Clamp: $\mathbf{\Sigma}^{\boldsymbol{clmp}} \leftarrow 1-\theta_c \leqslant \mathbf{\Sigma} \leqslant 1+\theta_s$
      $\mathbf{F}_p^{clmp} = \mathbf{U\Sigma}^{clmp}\mathbf{V}$
      $J_p \leftarrow J_p \frac{J_p^{uncl}}{\det(\mathbf{F}_p^{clmp})}$
      Clamp: $J_p \leftarrow 0.6 \leqslant J_p \leqslant 20.0$
    end for
  end for

C

CONSISTENCY, STABILITY AND CONVERGENCE DEFINITIONS

This appendix section gives formal definitions of consistency, stability and convergence properties used in this work. These arise from the existing literature: *Numerical Mathematics* text of Quarteroni et al. [141], *Nonlinear finite elements for continua and structures* text of Belytschko et al. [130], a course lecture of R. Fedkiw [142] and the work of Chartres & Stepleman [143]. A range of works are used, as universally standard definitions for these terms are not apparent in literature [143]. For the problem, find the solution x such that

$$F(x,d) = 0 \tag{C.1}$$

for the set of solution dependent data d, and the function relation F between d and x. The numerical solution to this problem is given by:

$$F_n(x_n, d_n) = 0 \quad \text{for} \quad n \geqslant 1 \tag{C.2}$$

Here, n is a parameter dependent on the case, and it is expected that the numerical solution converges to the exact solution $x_n \to x$ as $n \to \infty$. This requires that as $n \to \infty$, $d_n \to d$ and F_n approximates F. This leads to the definition of *consistency*:

$$F_n(x,d) = F_n(x,d) - F(x,d) \to 0 \quad \text{for} \quad n \to \infty \tag{C.3}$$

This evaluates that the chosen numerical approximate function F_n is able to more closely capture the underlying true solution x at d, as $n \to \infty$. Next, the numerical solution is *stable* if for small perturbations at the input level yield small perturbations in the output. The numerical method is said to be stable if it yields stable numerical solutions.

$$||x_n^A - x_n^B|| \leqslant C\varepsilon \quad \forall n > 0 \quad \text{for all } x_0^A \quad \text{such that } ||x_0^A - x_0^B|| \leqslant \varepsilon \tag{C.4}$$

Where C is a positive constant. Last, convergence is defined. For an admissible datum d in Equation C.1, the corresponding true solution $x(d)$ and the numerical solution $x_n(d+\delta d_n)$ for the datum $d+\delta d_n$:

$$\begin{aligned} &\forall \varepsilon > 0\, \exists n_0(\varepsilon),\, \exists \delta(n_0,\varepsilon) > 0 : \\ &\forall n > n_0(\varepsilon), \forall |||\delta d_n|| < \delta(n_0,\varepsilon) \qquad \Rightarrow ||x(d) - x_n(d+\delta d)|| \leqslant \varepsilon \end{aligned} \tag{C.5}$$

Convergence is distinctly different from consistency in that consistency only entails the chosen numerical method shows the ability to approximate the underlying function, while convergence demands the underlying function is approximated across an entire input domain. An example of this distinction is highlighted by the Courant Number, in that a particular finite difference solution can be consistent with the true function, however it will not converge if the Courant Number is not met [111].

D

GRAPH NETWORK-BASED SIMULATORS

Throw the lens of inductive bias presented by Battaglia et al. [84] is the GNS model framed [85] to motivate its aptitude in capturing the problem domain of particle point mass simulations. This section does not cover basics of GNN's as ample literature exists on the subject (See [84, 96]), nor does it cover the detailed implementation of the GNS model (see Sanchez-Gonzalez et al. [85]). Instead, a conceptual understanding of the model pipeline is given with motivation of architecture choices based on existing literature.

RELATIONAL REASONING AND INDUCTIVE BIAS

A driving principle in the Battaglia et al. [84] work is making *infinite use of finite means.* This is motivated via *combinatorial generalization*, generalizing to unseen realms using fixed known building blocks. Relational reasoning is proposed as the range of avenues in which the following curated building blocks may be composed:

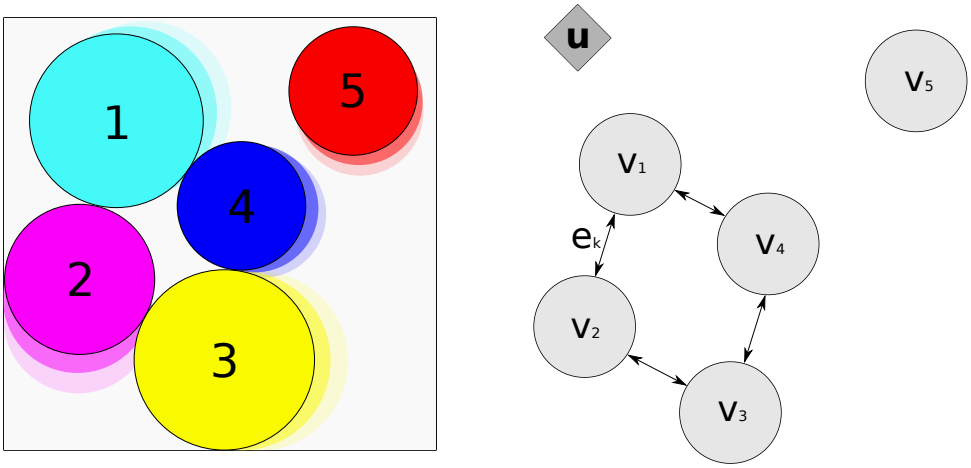
- entity - a select element with attributed quantities.
- relation - the means by which different entities interact,
- rule - a function which maps known entities and relations to other relations and entities.

The arrangement of the 3 building blocks stated yields a *structured representation*, and the manipulation of entities and relations via rules to other entities and relations is *structured computation.*

Inductive bias is the second key item in this lens, that a particular solution may be biased in a certain direction regardless of the underlying data. Bayesian models complete this via choice of a prior, and ML models do this via architecture choices. The reader is refereed to the text of Battaglia et al. [84] for an in-depth description.

GNN'S APTITUDE

Section 2.2 alludes to GNNs and their aptitude to capture physical domains. This is now done in a substantiated manner through relational reasoning and inductive bias. Figure D.1 shows elastic balls moving in a 2D frame, with interactions captured via a contact model, as expected with a conventional DEM perspective. This example showcases both the relational reasoning and inductive bias which graphs leverage.



(a) Physical domain of interest

(b) Graph representation

Figure D.1: Simplified elastic balls example

The transition from Figure D.1a to Figure D.1b is the expression of the problem domain via a graph, made of nodes v_i, edges e_k and a global **u**. This graph representation exemplifies the relational reasoning that each of the graph feature capture; entities, relations and rules. Conveniently, each node (entity) contains features of the ball (mass, radius, color, etc.). The edges (relations) in Figure D.1b are drawn between balls which are assumed to interact. It is emphasized the nodes which **are not** connected, ie node 5 is isolated. In a contact model the nodes in touch alone will have influence on each other. The balance between nodes which are and are not connected induces a bias over the domain, in contrast to assuming all nodes have equal influence on the other, as a naive neural fully connected implementation would. Lastly, a global **u** is given to capture invariant domain properties (gravity, system energy, center of mass, etc.).

Inductive bias utilizes *a priori* information to prioritize one solution over another. The center of mass of the system is computed from the summation of mass and position values for each of n balls, however the **order** in which this is done has no influence on the outcome. If a conventional neural network is used to make this computation, where each input neuron is a node's mass and location, and each ordering of the nodes is unique, then there are $n!$ permutations. Each permutation is an ordering of the nodes in the input layer. GNN's architecture induces permutation invariance [96], reducing this $n!$ permutation dependency to 1 by using the *a priori* information that the center of

Figure D.2: GNS Flowchart

mass is invariant to summation permutation.

GNS Model Algorithm Overview

Figure D.2 gives a gentle introduction to the GNS algorithm. The model architecture is built on the `tf_estimator.Estimator` class of `TensorFlow` 1. A simulation initiates with the metadata dataset normalization statistics, and the hyperparameters for the GNS model definition. A simulator is defined via Algorithm D.2 and Algorithm D.1. Noise is generated according to Algorithm D.4. This noisy sequence is used to construct a graph in Algorithm D.3. The `simulator` module wraps these together to compute predictions according to these definitions. The loss is computed according to the acceleration loss in Table 3.2 and the network is tuned using an Adam optimizer. Last, this is returned to an `EstimatorSpec` for `TensorFlow`.

INTERACTIONNETWORK MESSAGE PASSING BLOCK

The GNS model utilizes the `InteractionNetwork`[1] message passing architecture, originally proposed by Battaglia et al. [81]. Figure D.3 gives a conceptual visual for the architecture, while Algorithm D.1 details the updates made by the model. This specific model solely uses local information to make local updates, utilizing information of both the local nodes and local edges.

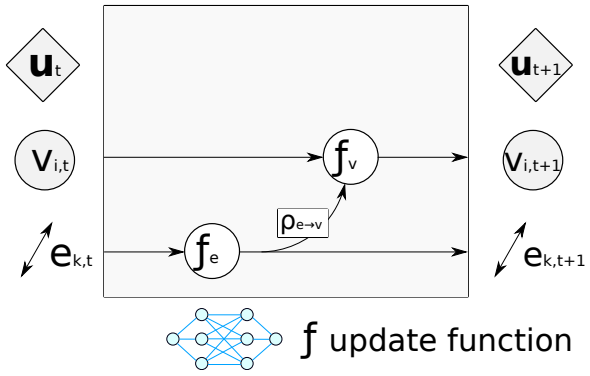


Figure D.3: `InteractionNetwork` Block

Algorithm D.1 `InteractionNetwork` message passing breakdown

Require: $E, V, \mathbf{u}$
  for $k \in \{1...N^e\}$ **do**
    $\mathbf{e}_{k,t+1} \leftarrow f_e(\mathbf{e}_{k,t}, \mathbf{v}_{r_{k,t}}, \mathbf{v}_{s_{k,t}}, \mathbf{u})$ ▷ Compute updated edge attributes
  end for
  for $i \in \{1...N^n\}$ **do**
    let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho_{e\to v}(E'_i)$ ▷ Aggregate edge attributes per node
    $\mathbf{v}_{i,t+1} \leftarrow f_v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$ ▷ Compute updated node attributes
  end for

ENCODER, PROCESSOR DECODER STRUCTURE

Section 2.2 introduces the Encoder, Processor, Decoder strategy utilized in literature. The specific GNS construction is detailed here to highlight model specifics. Importantly, recent work of Alon & Yahav [98] uncovered the *over-squashing* phenomena present when pooling too much information onto a node prior to it's update. Figure D.4 exemplifies this when summing neighboring nodal information (colors) onto a single node to be updated, all the information may be squashed into a single teal color. However, it is desired that the presence of all 4 colors is positioned on the node, not their mixture.

To alleviate this undesired effect, the space which the graph is constructed in is projected

[1] The `InteractionNetwork` architecture does not make use of globals, and thus no global updates are shown in Figure D.3.

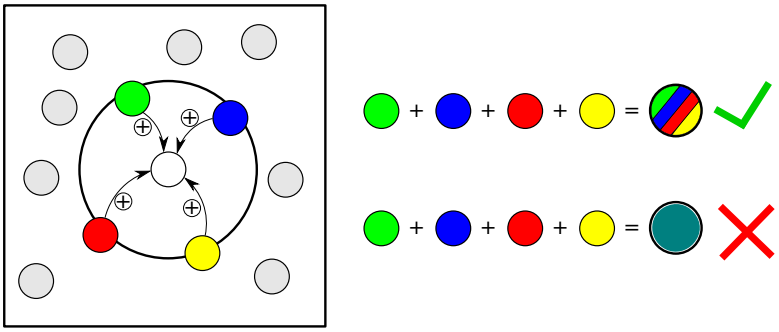


Figure D.4: Example of *over-squashing*

from 2D to a higher dimension space. This intends to let node and edge embeddings express more information in the latent space. Figure D.5 showcases the Encoder projection phase, which uses a trained neural network to project the low dimension vector to a high dimensional one, where more colors are seen, representing the latent space's ability to express more information. The processor stage completes m message passing steps according to the `InteractionNetwork` architecture, leading to the final output stage which is projected back to the regular 2D space. A step-by-step description is given by Algorithm D.2.

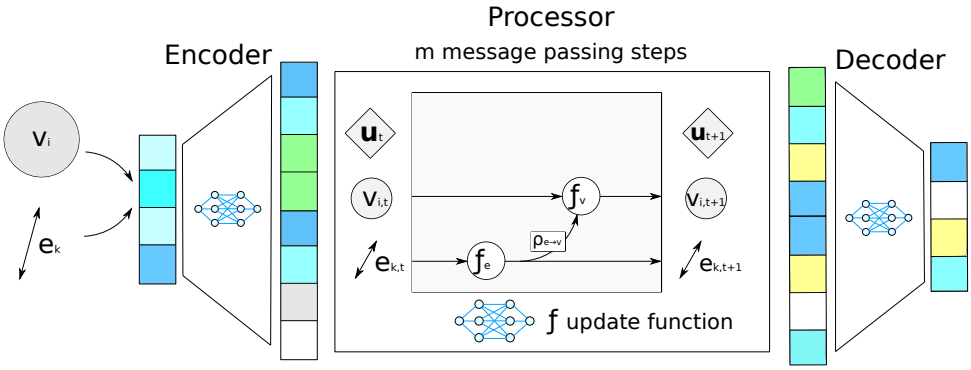


Figure D.5: Encoder-Processor-Decoder Architecture. Adapted from [144]

GRAPH CONSTRUCTION

In the same fashion as meshfree methods, the GNS model computes the graph connectivity prior to each encoding phase. That is, the model assumes no fixed connectivity *a priori* to the simulation, as mesh based methods conventionally do. The GNS setup builds the graph by iterating through every particle, and connecting all surrounding particle within a connectivity radius r to the central node, as seen for 3 example nodes in Figure D.6. A connectivity circle is drawn for three particles, edges are created, and the repetition of this for every particle yields a single graph $\mathcal{G}$ for input.

Algorithm D.2 `Encoder Processor Decoder`

Require: Graph $\mathcal{G}$ ▷ `Encoder`

for $i \in \{1...N^v\}$ **do** ▷ Project the nodes to the latent space

$V_i \leftarrow \phi_v^{\text{enc}}(v_i)$

end for

for $k \in \{1...N^e\}$ **do** ▷ Project the edges to the latent space

$E_k \leftarrow \phi_e^{\text{enc}}(e_k)$

end for

for m message passing steps **do** ▷ `Processor`

$E^{m+1}, V^{m+1} \leftarrow$ `InteractionNetwork`(E^m, V^m)

$E^{m+1} = E^m + E^{m+1}$ ▷ Append edge residuals

$V^{m+1} = V^m + V^{m+1}$ ▷ Append node residuals

end for

for $i \in \{1...N^v\}$ **do** ▷ `Decoder`

$v_i \leftarrow \phi_v^{\text{dec}}(V^m)$

end for

$\tilde{\tilde{\mathbf{x}}}_i \leftarrow v_i$ ▷ Extract normalized acceleration

$\ddot{\mathbf{x}}_i = (\tilde{\tilde{\mathbf{x}}}_i \cdot \mathbf{a}_{\text{std}}) + \mathbf{a}_{\text{mean}}$ ▷ Unnormalize acceleration

Algorithm D.3 `learned_simulator._encoder_preprocessor`

Require: $\mathbf{x}_i^{(t-h)\to t}$ for all particles

▷ **Define node features**

$\mathbf{x}^t \leftarrow \mathbf{x}_i^{(t-h)\to t}$ ▷ Define current position

$\dot{\mathbf{x}}^{(t-h+1)\to t} \leftarrow \mathbf{x}_i^{(t-h)\to t}$ ▷ Velocity via finite differences

$v_{s_k}, v_{r_k}, k \leftarrow \mathbf{x}^t$ ▷ Define sender nodes, receiver nodes and total edge count k via r

$\tilde{\mathbf{x}}_i^{(t-h+1)\to t} = (\dot{\mathbf{x}}^{(t-h+1)\to t} - \mathbf{v}_{\text{mean}})/\mathbf{v}_{\text{std}})$ ▷ Normalize velocity via scalar statistics

$\tilde{x}_i^{(t-h+1)\to t} \leftarrow \tilde{\mathbf{x}}_i^{(t-h+1)\to t}$ ▷ Merge 1^{st} and 2^{nd} velocity dimensions

$v_i \leftarrow \tilde{x}_i^{(t-h+1)\to t}$ ▷ Append flattened velocity sequence as a node feature

$\mathbf{l}_i \leftarrow \mathbf{x}^t$, boundary ▷ Compute the distance a node is from all 4 boundaries

$\tilde{\mathbf{d}}_i \leftarrow \mathbf{d}_i, r$ ▷ Normalize to r and clamp at ± 1

$v_i \leftarrow \tilde{\mathbf{d}}_i$ ▷ Append clipped distance to boundary to the node

$v_i \leftarrow n_i$ ▷ Append particle type embedding to the node

▷ **Define edge features**

$r_k = (\mathbf{v}_{s_k} - v_{r_k})/r$ ▷ Define edge length and orientation normalized to r

$e_k \leftarrow \mathbf{d}_k, |\mathbf{d}_k|$ ▷ Append edge orientation and its magnitude as an edge feature

▷ **Define graph**

$\mathcal{G} \leftarrow v_i, e_k$ ▷ Assign node and edge feature as a `GraphTuple` object.

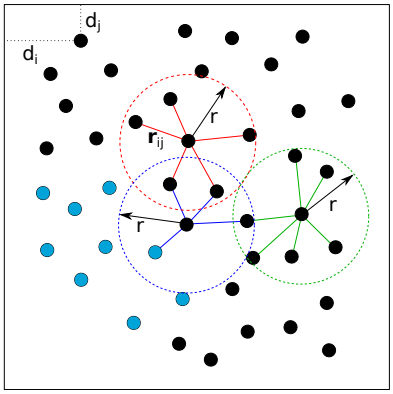


Figure D.6: Visualization of the GNS graph construction via a connectivity radius r

Error metrics

See Table 3.2 in the main text for a description of the error metrics and loss.

Features and labels

Table D.1 details the features and labels the GNS model uses to predict the single output, nodal acceleration $\ddot{\mathbf{x}}_i$. These values are normalized to either normalization statistics of the training data, or to the connectivity radius r hyperparameter. Algorithm D.3 details the means by which these input features are defined from a sequence of position values.

Table D.1: GNS model graph input features

Feature/ Label	Description	Location	Description	Normalized to
Feature	Velocity $\dot{\mathbf{x}}_i$	Node	Computed from position value finite differences going back h steps	Normalization statistics
Feature	Particle type n_i	Node	Classifying label	None
Feature	Clipped distance to boundary	Node	The distance to the outer box boundary, clipped to 1 if greater than r	r
Feature	x and y edge length	Edge	The x and y edge length decomposition	r
Feature	Edge length euclidean norm	Edge	Edge length magnitude	r
Label	Acceleration $\ddot{\mathbf{x}}_i$	Node	Directly predicted by the GNS model as the decoder stage output	Normalization statistics

Of particular interest are the normalization statistics used. As explained in the GNS publication [85], these are used to normalize velocity and acceleration values to zero mean and unit variance. To elaborate on this here, is how this normalization is done. These

training statistics are computed from finite difference of the position data to yield velocity and acceleration of every particle at every time step. This is reduced to 8 single scalar values, x and y direction velocity mean, velocity standard deviation, acceleration mean and acceleration standard deviation[2]. Prior to the Encoder phase, the velocity statistics are used to normalize the nodal velocity, and post the Decoder phase are these statistics used to invert the normalized predicted value.

RANDOM WALK NOISE

A random walk noise sequence is defined for the velocity values. Algorithm D.4 shows the used Gaussian random walk sequence provide a batch of position values $\mathbf{x}_i^t$ over an input sequence length h. The random walk is summed twice. This allows the random walk to be applied as a velocity random walk, instead of a positional random walk. A key component is the definition of σ_0. In the open sourced code, this is done setting σ_0 = 0.00067 / $\sqrt{h-1}$. The division by h is to yield the same noise at the end of the sequence. This normalization is an application of Bienaymé's identity.

Algorithm D.4 `Random walk velocity noise`

Require: $\mathbf{x}_i^t, \mathrm{d}t$ ▷ Batch of node positions
  for $i \in \{1...N\}$ **do** ▷ Compute velocity
    $\dot{\mathbf{x}}_i^t \leftarrow (\mathbf{x}_i^{t+1} - \mathbf{x}_i^t)/\mathrm{d}t$
  end for
  for $i \in \{1...N\}$ **do** ▷ Compute noise
    $\tilde{\dot{\mathbf{x}}}_i^t \leftarrow \mathcal{N}(0, \sigma = \sigma_0/\sqrt{h-1})$
  end for
  for t in h **do** ▷ Apply a velocity random walk
    $\tilde{\dot{\mathbf{x}}}_i^t \leftarrow \tilde{\dot{\mathbf{x}}}_i^t + \tilde{\dot{\mathbf{x}}}_i^{t-1}$
  end for
  for t in h **do** ▷ Integrate the random walk
    $\tilde{\mathbf{x}}_i^t \leftarrow \tilde{\dot{\mathbf{x}}}_i^t \cdot \mathrm{d}t$
  end for
  for t in h **do** ▷ Apply a displacement random walk
    $\tilde{\mathbf{x}}_i^t \leftarrow \tilde{\mathbf{x}}_i^t + \tilde{\mathbf{x}}_i^{t-1}$
  end for
  Return $\tilde{\mathbf{x}}_i^t$

NORMALIZATION STATISTICS

The open-source code of Sanchez-Gonzalez et al [85] implements a model without implementing the time step. The model assumes a uniform time step exists in the entire dataset. Based on this assumption, the code is written assuming a normalized time step $\mathrm{d}t$ of 1. The corresponding paper motivates this decision with, "omitting constant Δt for simplicity".

[2] This raises the unsolved question as to what degree using these normalization statistics directly from the training dataset hinders generalization of the GNS model beyond training trajectories.

Sanchez-Gonzalez et al [85] additionally released datasets to train the proposed GNS model. Within these datasets are scalar normalization statistics for the mean and standard deviation of velocity and acceleration. The publication does not state how they are computed, which means an approximate must be determined. Via trial and error, Algorithm D.5 is determined which yields the normalization statistics in the published datasets via the position data. This algorithm computes mean and standard deviation for velocity and acceleration through finite differences assuming the time step is 1.

Algorithm D.5 `Normalization Statistics`

Require: x_i^{dtj} for i in N particles in d dimensions of t time steps and j training trajectories.
  for j in training trajectory **do**
    Trajectory = Trajectories[j] ▷ Trajectory shape $[t, N, d]$
    for d in dimensions **do**
      v = numpy.gradient(Trajectory[:, :, d], axis = 0) ▷ Use numpy gradient to compute the pseudo-velocity, with the time step = 1
      $\mathbf{v}_{\text{mean}}$[j, d] = numpy.mean(**v**) ▷ Compute the mean velocity
      $\mathbf{v}_{\text{std}}$[j, d] = numpy.std(**v**) ▷ Compute the velocity standard deviation
      a = numpy.gradient(velocity, axis = 0) ▷ Use numpy gradient to compute the pseudo-acceleration, with the time step = 1
      $\mathbf{a}_{\text{mean}}$[j, d] = numpy.mean(**a**) ▷ Compute the mean acceleration
      $\mathbf{a}_{\text{std}}$[j, d] = numpy.std(**a**) ▷ Compute the acceleration standard deviation
    end for
  end for
  ▷ Compress the mean and standard deviation values.
  $\mathbf{v}_{\text{mean}}$ = numpy.mean($\mathbf{v}_{\text{mean}}$, axis=0)
  $\mathbf{v}_{\text{std}}$ = numpy.std($\mathbf{v}_{\text{std}}$, axis=0)
  $\mathbf{a}_{\text{mean}}$ = numpy.mean($\mathbf{a}_{\text{mean}}$, axis=0)
  $\mathbf{a}_{\text{std}}$ = numpy.std($\mathbf{a}_{\text{std}}$, axis=0)

Computational Complexity

Chami et al. [78] reports the training complexity of this GNN formulation, and is given in Equation D.1. Keeping the count of node features, edge features and labels constant, the model is linear in complexity with both node count and edge count.

$$\mathcal{O}(|E|\mathscr{D} + |V|\mathscr{M}) \tag{D.1}$$

E- number of node neighbors

$\mathscr{D} = \sum_l d_l$ - total dimension of all layers

V- number of nodes

$$\mathscr{M} = \sum_l d_l d_{l+1} \text{ - accounting for matrix multiplication}$$

E

GEOMETRIC DEEP LEARNING LENS

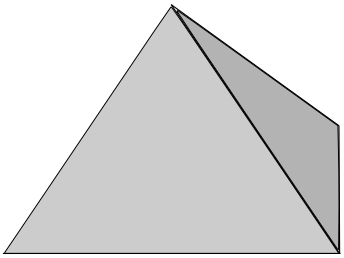
The lens of Geometric Deep Learning (GDL) unifies existing prominent ML models through invariants identified in all models. The recent work of Bronstein et al. [126] lays out a formal analysis of this program, which key elements are summarized here. This framing is vital to understand the GNS discussion provided in Chapter 5.

GDL FUNDAMENTALS

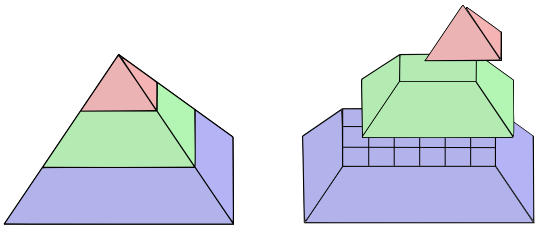
*A multitude of machine learning (ML) architectures (CNN, GNN, RNN, etc.) are implemented and reported on without a complete understanding for **why** they perform well. This text identifies core attributes which all these architectures share, and unifies them under a single Blueprint.*

The essence of deep learning is: (*i*) Representation of the interested domain with features (*ii*) learning via local gradient descent (implemented via backpropagation). Convention is used that the ML system operates on signals (functions) which exist in a domain Ω. With large amounts of data in the ML field comes with it the curse of dimensionality. From a Bayesian perspective, **geometric priors** may be leveraged to reduce this high dimensional space by exploiting geometric regularity of the underlying domain (Figure E.1 & Figure E.2). This is motivated via the Erlangen Program, the study of geometry through invariants. That is, focusing on the quantities of interest which remain unchanged via a group of transformations. For example, Euclidean transformations preserve lengths and angles, while affine transformations preserve parallelism.

Symmetry and scale separation are the 2 geometric priors exploited for defining a blueprint to unify modern ML architectures. Symmetry is defined as a transformation which leaves a property unchanged, invariant (Figure E.2). Examples are molecule classification, which is invariant of the molecule's rotational orientation, or in summing forces in particle systems, which is invariant to permutation. A set of symmetries is referred as a group $\mathfrak{G}$,

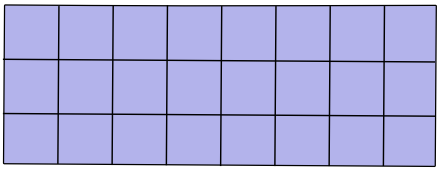


(a) Visual representation of the interested domain

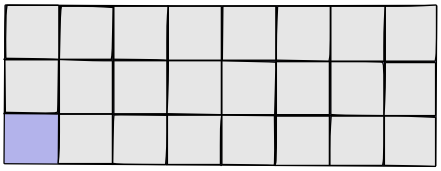


(b) Geometric prior of scale separation

Figure E.1: Visualization of scale separation



(a) Visual of a single domain scale



(b) Implementation of symmetry geometric prior

Figure E.2: Visualization of symmetry

which obeys the properties of composition, associativity, identity, inverse and closure (but not commutativity) see [126](pg. 13). Group action is $\mathfrak{G}$ acting on a point $\mathbf{u}$ in Ω, rendering it to another location in Ω.

> Noether's Theorem connects symmetry principles and conservation laws. A prominent example is invariance to time yields the conservation of energy, and invariance to spatial translation yields the conservation of linear momentum [145].

Of significant interest is linking the underlying domain, to the functions defined on the signals (ie learned interpolations). **The symmetry of Ω, which the signals $\mathscr{X}(\Omega)$ are defined on, induces structure to the function f defined on these signals**. The goal is thus to reduce the space of potential interpolates $\mathscr{F}(\mathscr{X}(\Omega))$ to those which satisfy one of the following symmetry priors: (*i*) invariant if the output $\mathscr{Y}$ is unaffected by the group action on the input. (*ii*) equivariant if the output is affected in the same manner which the group action acts on the inputs. An example of $\mathfrak{G}$-invariance is system kinetic energy, as the quantity is unaffected by the orientation or location of motion, only on the magnitude. An example of $\mathfrak{G}$-equivariance is the input direction of applied forces on a particle, as the resultant force is dependent on the input's orientation.

Scale separation is a geometric prior which leverages our domain being divided into levels of structure (Figure E.1b). Each level has its own respective structure and symmetry which it obeys. The choice of levels is dependent on the problem. An example of global

Table E.1: Implementation of the GDL Blueprint for the GNS model.

Blueprint Element	**Description**	**GNS Implementation**
Linear $\mathfrak{G}$-equivariant layer	Equivariant feature	$\mathbf{x}_i \rightarrow \dot{\mathbf{x}}_i$
Nonlinearity σ	Activation function	ReLU
Local pooling (coarsening)	Localized information sharing	Learned message passing
$\mathfrak{G}$-invariant layer (global pooling)	A global readout state	None

level structure for the particle domain is $\mathfrak{G}$-invariant for the system energy, while local summation of forces is a $\mathfrak{G}$-equivariant structure. An equivariant map is considered localized if the mapping depends solely on values about a small radius r, known as the *receptive field*. This is analogous to the *support domain* in conventional meshfree methods [12].

Figure E.3 shows the blueprint presented by Bronstein et al. The formal definition is given in the text (pg. 29), however for conceptual purposes, Figure E.3 exemplifies the localized equivariance and global invariance used in graphs, facilitated via coarsening pooling operations.

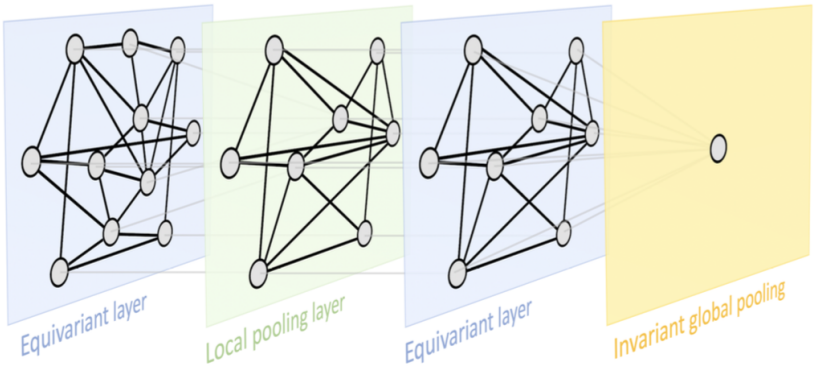


Figure E.3: Blueprint of geometric deep learning for an example graph. Typical implementation is a permutation equivariant layer for computing nodal features, followed by local pooling for graph coarsening and lastly a permutation-invariant global readout layer [126].

Table E.2: Benchmark function equivalents for the GDL structure

GDL item	Benchmark component
Domain Ω	MPM material points
Signals $\mathscr{X}(\Omega)$	ML input features
Function f	Learned GNS model

GNS MODEL

Table E.1 defines how the GNS model fits the GDL Blueprint[1]. The input nodal values of absolute position are transformed to a pseudo-velocity[2], which via aggregation are equivariant under Euclidean transformations [126](pg. 84). Localized information coarsening, and simultaneously propagation of information locally, is done through a learned pooling function. In the published GNS model no global readout is used[3]. Table E.2 shows how the key computational components of this project fits the GDL domain.

[1] See Bronstein et al. [126](pg. 29) for a formal definition of the Blueprint elements.

[2] 'pseudo' is used here as velocity is computed as finite differences of position, divided by 1, ie assuming Δt is unity.

[3] The InteractionNetwork architecture is used, which has no global readout. The GNS paper does comment in the appendix that using a global readout does not have significant improvement on performance.

F

Snow Blocks Implementation

The snow blocks dataset is created using the demo code of Hu et al [66]. This is a direct implementation of the default code setup, with minimal changes[1]. Figure F.1 gives a visual of the setup, Table F.2 details the GNS implementation and Table F.3 details the additional MLS-MPM parameters used by Hu et al. The reader is referred to Appendix B for a description of the constitutive model and the MLS-MPM algorithm.

A key choice made is the block positioning. All 3 have their respective starting points in the y location fixed at $y1_0$, $y2_0$ and $y3_0$ (see Table F.3). However, the x coordinates are sampled at random using a uniform random distribution between and including the bounds[2] [0.2, 0.6]. The purpose of this is that every trajectory is a unique implementation to create dataset variability. Additionally, Hu et al uses `TaiChi`'s[3] random number generator to randomly position the 200 material points making up each square.

To uniquely identify each block separately in the GNS model, particles belonging to each respective block are given different particle type labels. See Appendix D for a description of this node feature. Two formulations of the dataset are given, each using the same original position data, with only different particle labels assigned (see Table F.1).

[1] Changes made are the starting location of each block, the total particles each block contains and a termination final time added.
[2] Figure F.1 shows these x coordinate bounds in red.
[3] `TaiChi` is the base program which the MLS-MPM code is written on by Hu et al [66].

Table F.1: Snow blocks dataset particle labels (see Figure F.1)

Dataset	**Particle type label order (top, middle, bottom)**
snow blocks v1	7, 0, 5
snow blocks v2	5, 0, 7

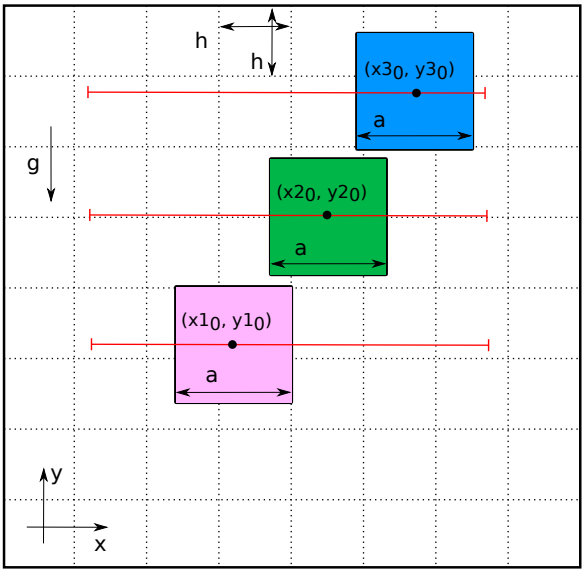


Figure F.1: Snow blocks visual setup

Table F.2: Snow blocks setup

GNS Hyperparameters		**Dataset Parameters**	
GNN Architecture	`Interaction Network`	Total time	0.30
Loss function	L_2 norm acceleration	gravity g	200
Learning optimizer	Adam	time step	1e-4
Learning rate	exponential decay from 10^{-4} to 10^{-6}	Square side length a	0.16
Training steps	10^5 and 10^6	Mesh size h	0.0125
Message passing steps	10	Num. particles	600
NN Architecture	2 hidden layers 128 layer neurons	Train/test/valid sets	300/20/15
Noise std	6.7e-4	Trajectory length	1000
Input sequence length h	5	Dataset time step	1e-3
Connectivity radius r	0.023		
Seed	33		

Table F.3: Additional snow blocks MLS-MPM parameters

Parameter	**Value**
Particle mass	1.0
Young's modulus	1000
Poisson's ratio ν	0.20
Hardening	10.0
Critical compression θ_c	0.025
Critical stretch θ_c	0.0075
$x1_0 = x2_0 = x3_0$:	[0.20, 0.60]
$y1_0$, $y2_0$, $y3_0$:	0.45, 0.65, 0.85

G

Lagrangian Discretization Limitations

Attention has been raised that the assumptions made in MPM's discretization of the material into Lagrangian points needs to be addressed to understand the limitations it imposes on the computational model. This is done in two parts: (*i*) The discretization assumptions made in literature (*ii*) The implications of this assumption.

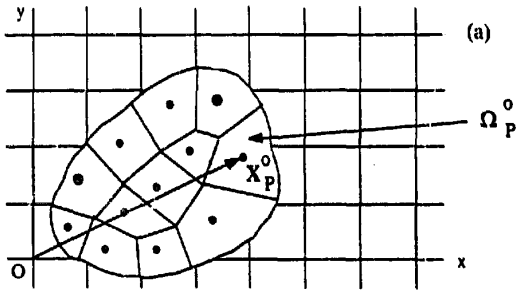


Figure G.1: MPM discretization diagram [13]. Continuum material points overlaying an Eulerian grid.

Figure G.1 depicts the localized continuum that each material point represents in the MPM configuration. The original publication of MPM contrasts the method with FEM, implying that it is an alternative option for the method. Table G.1 details key paper implementations of MPM, showing that with a single exception [70], MPM uses a continuum domain assumption.

Table G.1: Summary of particle physical representations from key papers.

Implementation	Particle physical representation	Keywords	Material point attributes
MPM [13]	Continuum	Original publication of MPM	mass, position, velocity, strain & stress
MPM [70]	Granular grain	Implementation for granular material	mass, position, velocity, strain & stress
MPM [7]	Continuum	Implementation for snow	mass, position, velocity & deformation gradient
MLS-MPM [66]	Continuum	Ground truth model in this project	volume, mass, position, velocity & deformation gradient

The snow science community takes two main categories for classifying mechanical constitutive models [146]:

- (*i*) phenomenological - mechanical features are described using global variables which assume a homogenized continuum.
- (*ii*) micromechanical - mechanical features are described from microscopic granular interactions.

The acceptance of a phenomenological or a micromechanical based approach to the constitutive model is a long-standing open research question in the community, dating back to the '60s [4, 5]. There are both prominent phenomenological continuum approaches [5, 147–149], and micro-level granular approaches [6, 150, 151] existing in literature. A guiding principle identified is: the continuum approach is taken when larger scale simulations are made, which can not computationally afford modeling grain-to-grain interactions. Importantly, the implications of taking a continuum approach over a granular approach appears to be an open-research question itself.

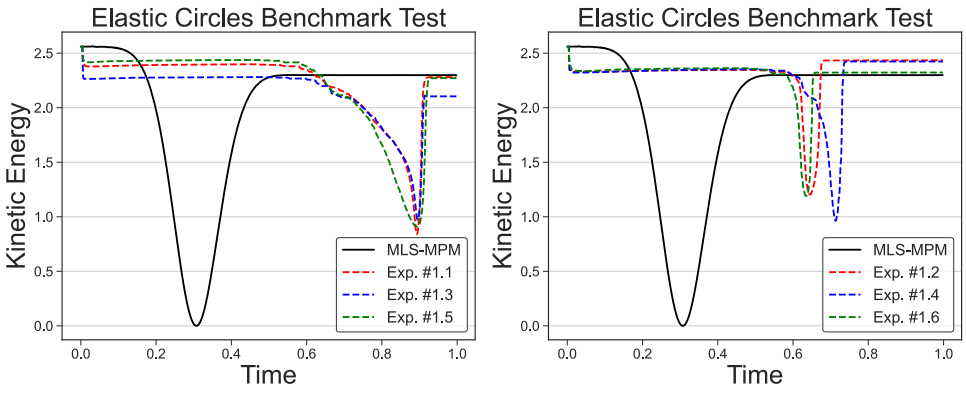
CONCLUSION

The ground truth model in this project (MLS-MPM) uses a continuum approximation of snow, which the implications of this assumption is an open-research question. It is concluded that the approach is not directly physically representative of the granular material.

H

Elastic Bodies Additional Results

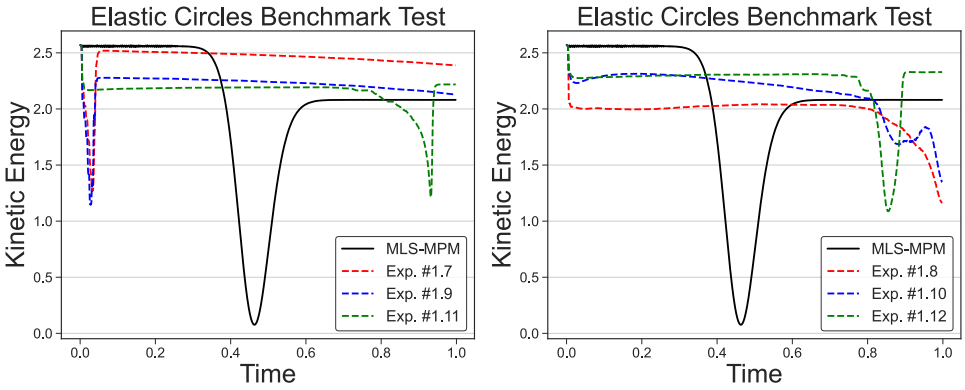
Included here are the kinetic energy plots corresponding to the benchmark setup described in Section 3.1. The legend in these figures corresponds to Table 3.7 and Table 3.8.



(a) 10^5 training steps

(b) 10^6 training steps

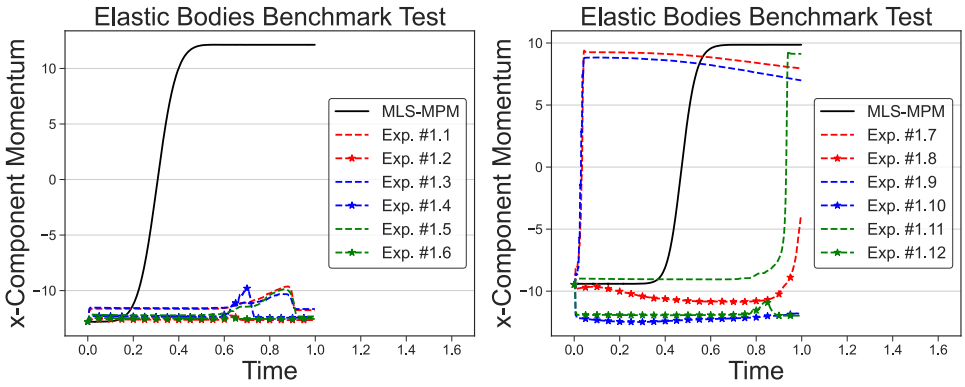
Figure H.1: GNS model kinetic energy predictions for dataset v1



(a) 10^5 training steps (b) 10^6 training steps

Figure H.2: GNS model kinetic energy predictions for dataset v2

Figure H.3 shows the x-component of momentum for the top left body during respective labeled numerical experiments. Figure H.3a shows that although collision dynamics completely fail, there is an apparent linear momentum conservation. This is covered in the discussion of Section 3.1. Figure H.3b shows the instabilities in different GNS predictions. These arise from the rotational dependencies of the model to its chosen ML features. This is covered in Section 3.1.

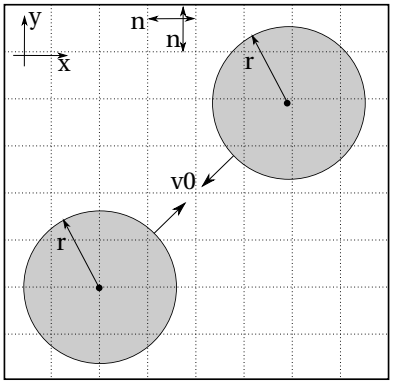


(a) Dataset v1 (b) Dataset v2

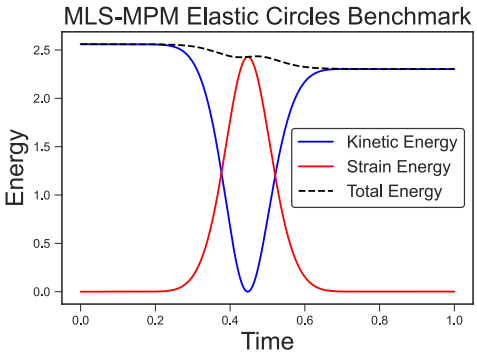
Figure H.3: Momentum of the top left body during simulations

I

ELASTIC BODIES ROTATIONAL ENERGY



(a) Elastic bodies benchmark test initial conditions.



(b) Benchmark test energy evaluation

Figure I.1: Elastic bodies benchmark setup.

The elastic bodies benchmark test (Figure I.1a) proposed by Sulsky et al. [13] is used in this thesis to determine if the GNS model [85] can capture underlying domain physics of interest. Section 3.1 highlighted the rotational motion in the MLS-MPM and GNS predictions. Here, these will be quantified in a spatial-temporal reference frame to illustrate their presence. This is done by computing the residual velocity, the difference in the body's mean velocity, and that of each particle.

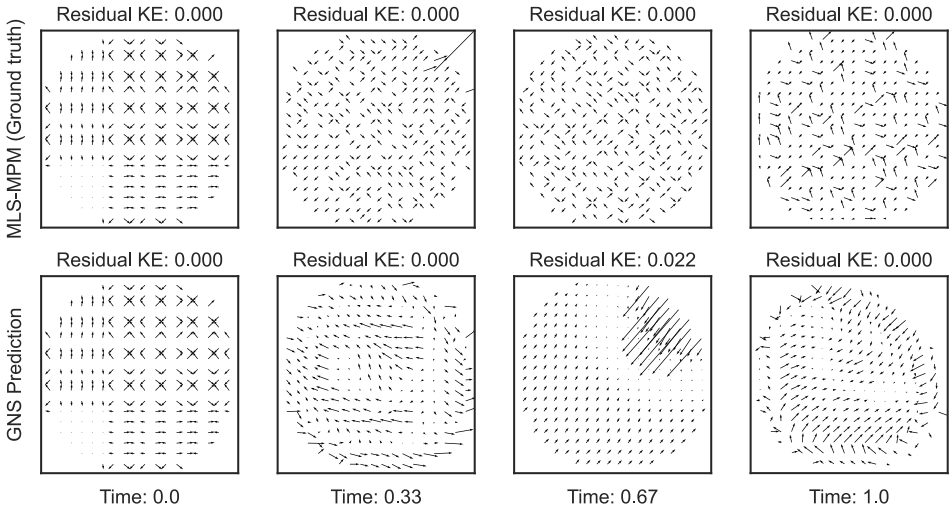


Figure I.2: Experiment #1.1 lower left body residual kinetic energy

The body velocity $\mathcal{V}$ of a single body of N particles, each of velocity $\mathbf{v}_i$ is taken as:

$$\mathcal{V} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{v}_i$$

The residual velocities $\boldsymbol{v}_i$ of each particle is thus defined as:

$$\boldsymbol{v}_i = \mathbf{v}_i - \mathcal{V}$$

The kinetic energy $\mathcal{K}$ of the residual velocity for k dimensions and N object particles each of equal mass m is assumed as:

$$\mathcal{K} = \frac{1}{2}m\sum_{k=1}^{2}\sum_{i=1}^{N}v_{ik}^2 \tag{I.1}$$

Figure I.2 and Figure I.3 show the residual velocities $\boldsymbol{v}_i$ plotted as a vector field. Vector magnitudes are defined relatively for visualization only. The title of each graph includes the magnitude of the residual kinetic energy with unit energy, computed according to Equation I.1.

Figure I.2 illustrates that the elastic bodies dataset v1 MLS-MPM simulation and GNS prediction contain no rotational or vibration residual energies. There is a clear residual kinetic energy magnitude in the GNS prediction at 0.67 unit time. This is the collision deformation.

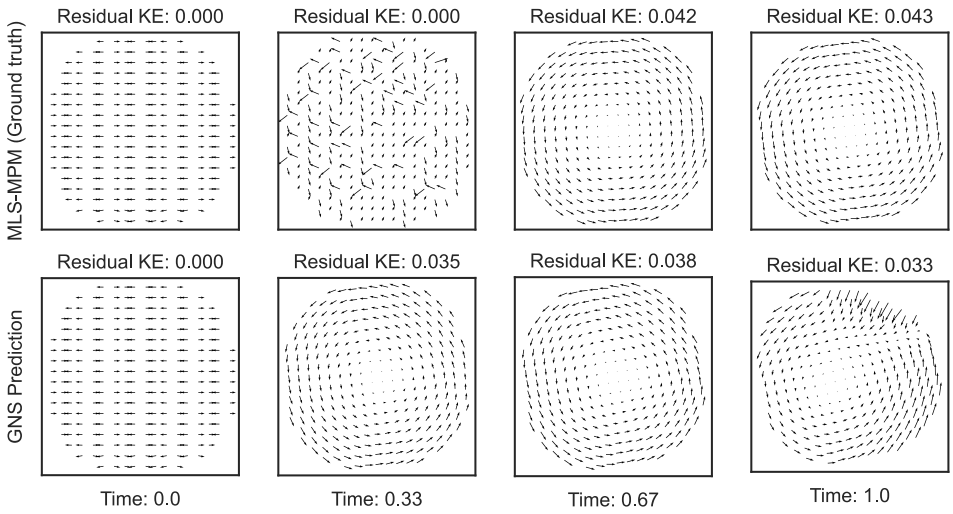


Figure I.3: Experiment #1.8 lower left body residual kinetic energy

Figure I.3 shows a clear rotational energies in the dataset v2 formulation. The MLS-MPM ground truth simulation contains this rotational only after impact. Contrarily, the GNS prediction shown clear rotational motion prior to the impact at 0.33 unit time.

J

Elastic Bodies Benchmark Variations

Variations between the MLS-MPM implementation of the elastic bodies benchmark, and that of MPM reported by Sulsky et al [13] are identified in this work. An analytical solution to a simplified point mass representation is done, and a discussion is drawn around the results.

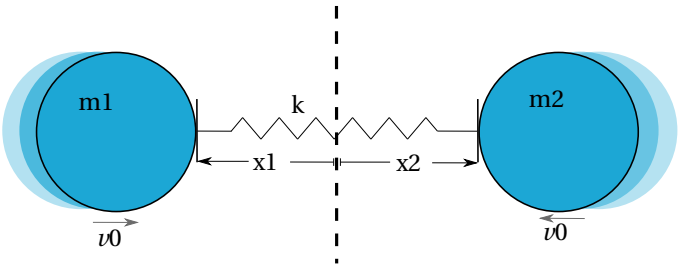


Figure J.1: Elastic bodies simplified diagram

Deriving the equation of motion via the Lagrange equation

Lagrange equation states the dynamic equilibrium of a system via its generalized coordinates q_s as the following:

$$\frac{d}{dt}\frac{\partial \mathcal{T}}{\partial \dot{q}_s} - \frac{\partial \mathcal{T}}{\partial q_s} + \frac{\partial \mathcal{V}}{\partial q_s} - Q_s^{\text{ncons}} = 0$$

Which is a function of the system's kinetic energy $\mathcal{T}$, its potential energy $\mathcal{V}$ and non-conservative forces Q_s^{ncons}

The following assumptions are taken:

- let $m_1 = m_2 = m$
- $x_1 = x_2 = x$

As the system contains conservative forces only (an elastic impact), $\mathcal{Q}_s^{\text{ncons}}$ is eliminated.

The kinetic and potential energy definitions follow:

$$\mathcal{T} = \frac{1}{2} m_1 \dot{x}_1^2 + \frac{1}{2} m_2 \dot{x}_2^2 = m\dot{x}^2$$

$$\mathcal{V} = \frac{1}{2} k x_1^2 + \frac{1}{2} k x_2^2 = k x^2$$

And the Lagrange equation terms follow as:

$$\frac{d}{dt}\frac{\partial \mathcal{T}}{\partial \dot{q}_s} = 2m\ddot{x}$$

$$\frac{\partial \mathcal{T}}{\partial q_s} = 0$$

$$\frac{\partial \mathcal{V}}{\partial q_s} = 2kx$$

Arranging the resultant terms into the Lagrange equations yields for the system equation of motion:

$$m\ddot{x} + kx = 0$$

SOLVING THE EQUATION OF MOTION ORDINARY DIFFERENTIAL EQUATION VIA THE LAPLACIAN METHOD

$$0 = \mathcal{L}[m\ddot{x} + kx]$$

$$= m\mathcal{L}[\ddot{x}] + k\mathcal{L}[x]$$

$$= m[s\mathcal{L}[\dot{x}] - \dot{x}(0)] + kX(s)$$

$$= m[s(sX(s) - x(0)) - \dot{x}(0)] + kX(s)$$

$$= ms^2X(s) - msx(0) - m\dot{x}(0) + kX(s)$$

$$= X(s)(ms^2 + k) - msx(0) - m\dot{x}(0)$$

$$X(s) = \frac{msx(0) + m\dot{x}(0)}{ms^2 + k}$$

$$= \frac{sx(0) + \dot{x}(0)}{s^2 + k/m}$$

$$= \frac{sx(0)}{s^2 + k/m} + \frac{\dot{x}(0)}{s^2 + k/m}$$

$$\text{let } \sqrt{k/m} = \omega$$

$$= x(0)\frac{s}{s^2 + \omega^2} + \frac{\dot{x}(0)}{\omega}\frac{\omega}{s^2 + \omega^2}$$

Apply the inverse Laplace transform $\mathscr{L}^{-1}$. Helpful to lookup a Laplace transform table here.

$$x(t) = x(0)\cos(\omega t) + \frac{\dot{x}(0)}{\omega}\sin(\omega t)$$

And for investigating the system kinetic energy, here is the term's first derivative with respect to time.

$$\dot{x}(t) = -x(0)\omega\sin(\omega t) + \dot{x}(0)\cos(\omega t)$$

KINETIC ENERGY

With the solved system velocity with time, comes the kinetic energy with time. This will be done via a piecewise arrangment of: (i) prior to collision, (ii) during collision, and (iii) post collision.

The kinetic energy prior to collision is taken as.

$$\mathscr{T} = \frac{1}{2}m_1\dot{x}_1(0)^2 + \frac{1}{2}m_2\dot{x}_2(0)^2 = m\dot{x}(0)^2$$

The kinetic energy during the collision is taken as.

$$\mathscr{T}(t) = m\dot{x}(t)^2$$

The kinetic energy after the collision is taken as.

$$\mathscr{T}(t) = m\dot{x}(0)^2$$

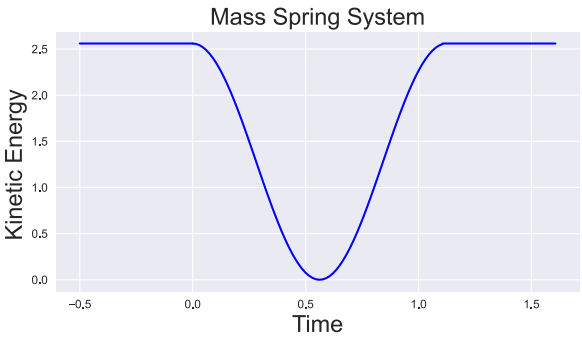


Figure J.2: Analytical solution to the mass-spring system: k = 1000, m = 128, $v_0 = \sqrt{2} \cdot 0.1$

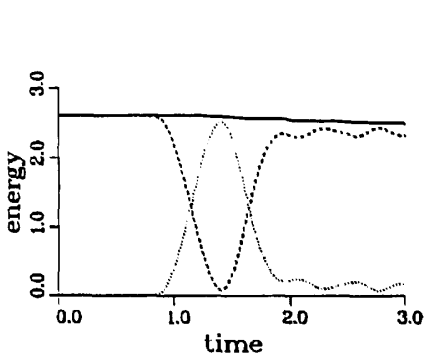
COMPARING RESULTS

The use of the elastic bodies benchmark test is motivated for the main purpose of evaluating energy conservation in the system, as discussed in Chapter 3. After implementing this benchmark in MLS-MPM compared to MPM, a key difference is seen between Figure J.3a and Figure J.4a, the total time of the collision. The simplified point mass model shows a total time of collision around 1 second (Figure J.2), and this is also seen in Figure J.3a, but is not found in Figure J.4a. Thus, a key difference seen in the MLS-MPM implementation is the total collision time.

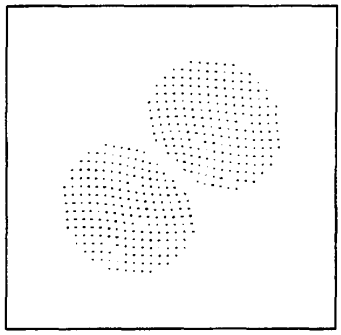
Next, qualitative measures of the degree which the bodies are deformed and the space between contact is seen between Figure J.3b and Figure J.4b. The distance between the two bodies is believed to be due to the basis function used on the nodes. MPM uses a linear interpolation function with a width of one node, while MLS-MPM uses B-splines with a width of one and a half nodes. Thus the larger support domain of this kernel causes a greater separation at the collision.

This distance is graphically shown in Figure J.5 where the center of mass for each circle is shown as a function of time. The minimum gap between the two bodies is computed at a distance of 0.483, and this distance is seen in Figure J.5. Assuming a rigid body contact, where each body has a radius of 0.19, the gap between the two bodies at collision does not drop below a distance of 0.107.

In conclusion, the key parameter of the energy conservation is largely captured by both the MPM and MLS-MPM models. The key difference between the two is the total time of the collision, which the MLS-MPM model is different from that found in MPM and the analytical solution to the simplified problem.

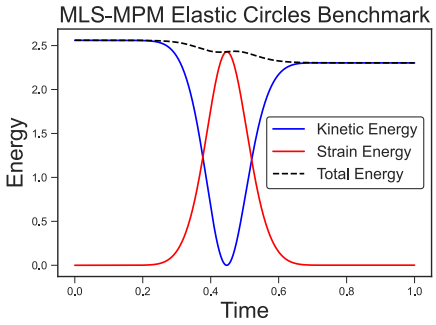


(a) Energy diagram

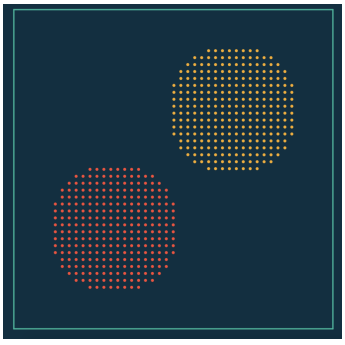


(b) Circle deformation

Figure J.3: Results of Sulsky et al [13]



(a) Energy diagram



(b) Circle deformation

Figure J.4: MLS-MPM Results

J

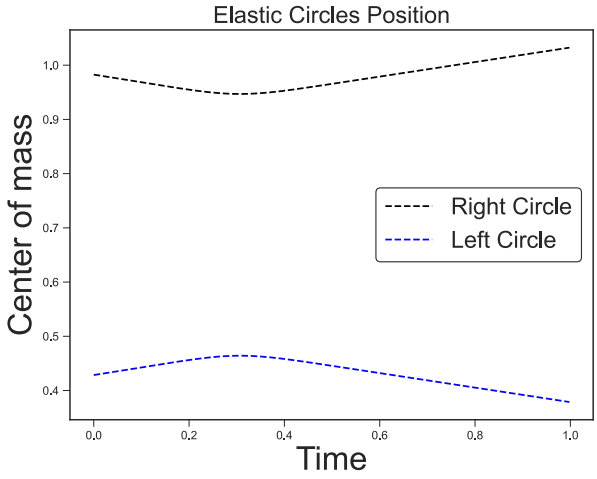


Figure J.5: Center of mass distance from origin for the left and right bodies.

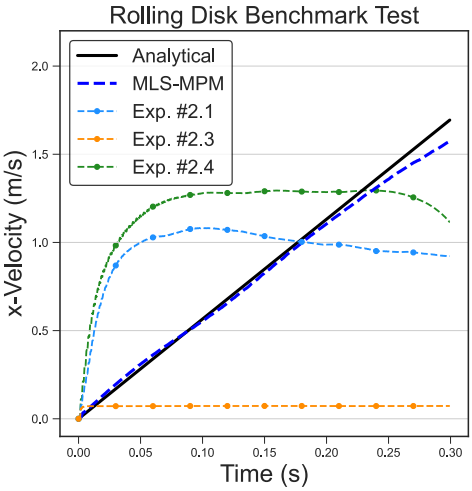
K

Rolling Disk Additional Results

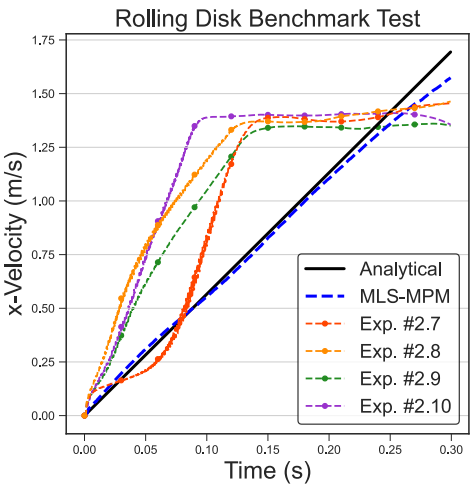
Chapter 3 and Chapter 4 showed the implementation of the rolling disk benchmark test, and the utility of its results in analyzing a GNN mode. For completeness, velocity plots corresponding to these sections are included here. Unlike the acceleration plots shown in these chapters, the velocity plots here do not visibly have noise.

Chapter 3 and Chapter 4 also highlighted the noise which exists in the derived acceleration data. A reminder, the standard GNS model uses twice central finite differences of position to arrive at the target acceleration. Figure K.4 shows the noise in this target data, and is quantified via the variance in computed acceleration value.

Figure K.5 views the vector field for Experiment #2.7 shown in Figure 3.18. At each time step given, the residual kinetic energy is calculated according to Equation I.1. This quantity shows there is a difference in rotational energy between the MlS-MPM method and the GNS prediction.

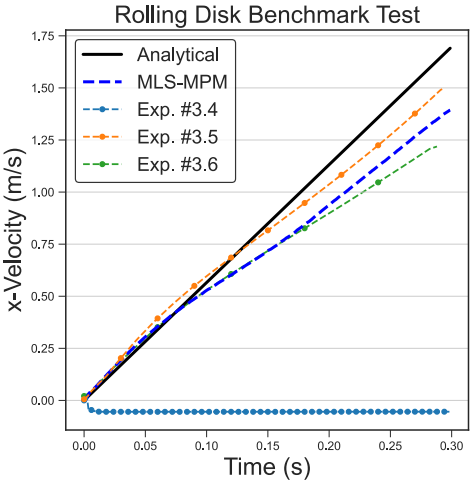


(a) 10^5 training steps

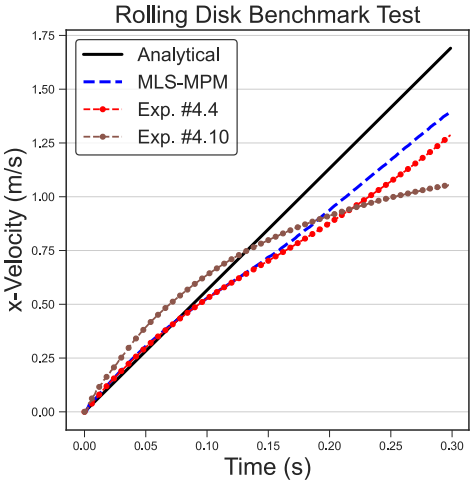


(b) 10^6 training steps

Figure K.1: Rolling Disk Velocity Plots for Set #2

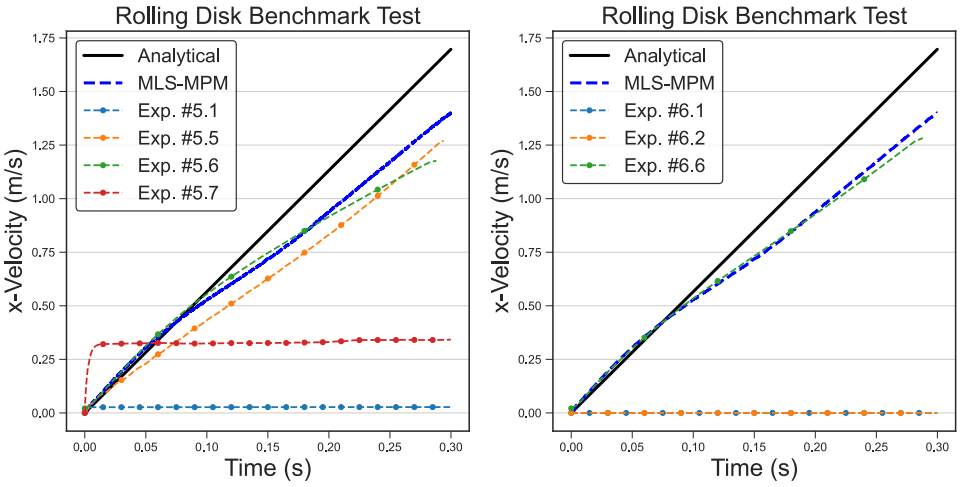


(a) Set #3 Velocity



(b) Set #4 Velocity

Figure K.2: Rolling Disk Velocity Plots



(a) Set #5 Velocity

(b) Set #6 Velocity

Figure K.3: Rolling Disk Velocity Plots

Time step (s)	**Acceleration Standard Deviation** (m/s^2)
3e-05	13.160
6e-05	3.9157
3e-04	1.3468
6e-04	1.1469
3e-03	0.7617
6e-03	0.7576

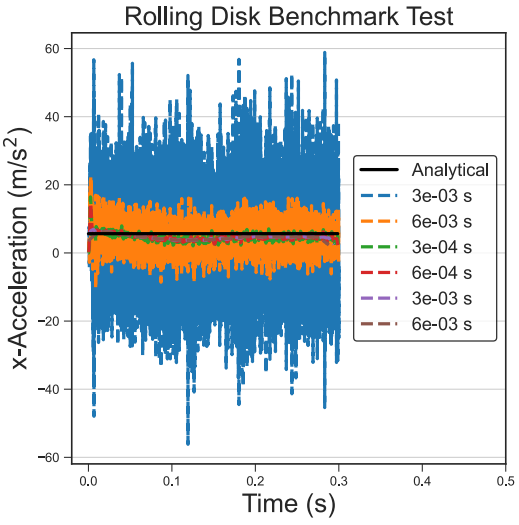


Figure K.4: MLS-MPM acceleration noise as a function of the time step

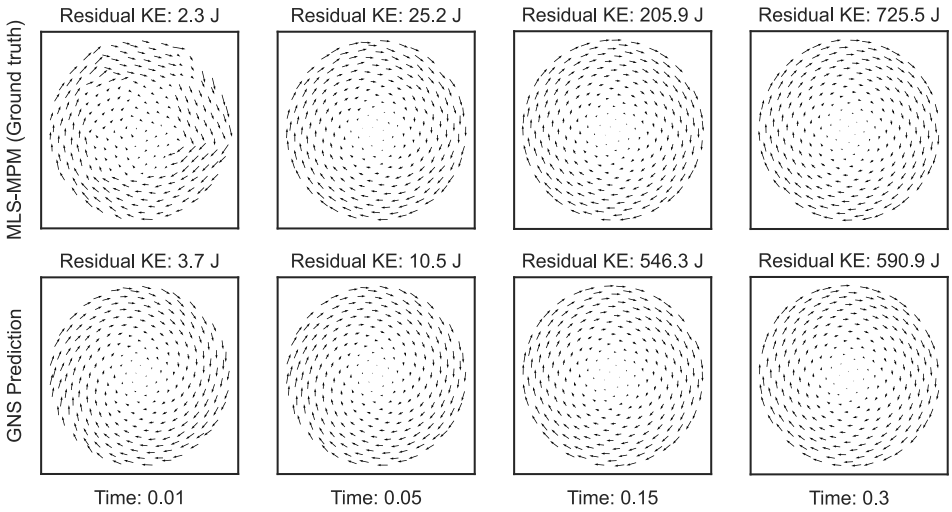


Figure K.5: Experiment #2.7 velocity vector field and residual kinetic energy

L

ROLLING DISK DISCRETIZATION

Algorithm L.1 is used to discretize the circle domain for the rolling disk benchmark test. This is necessary as no exact discretization method is specified by Bardenhagen et al for the benchmark test [70]. One may infer that a uniform grid is used, as in other prominent papers (see [13, 17]). In this uncertainty, an algorithm which uniformly distributes particles within a circle is chosen. The purpose is to meet the uniform mass distribution expected over a disk area and a smooth radial surface on the outside.

Section 3.2 introduced the limitations encountered with the MLS-MPM method when applying boundary conditions. Specifically, the offset which the rolling disk moves along the lower boundary. The interaction of this disk with the boundary is found to be very sensitive to the initial vertical placement of the disk. For the value of α=3 chosen in Algorithm L.1, a value of b_0=5·10^{-3} m is required such that the particles do not interact with the boundary condition applied at the bottom row. The connection between α and b_0 results from α determining how many particles lie on the boundary of the surface describing the disk surface. Thus, directly how many particles interact with the boundary condition. This interaction is an open research direction. It is not pursued here as the focus of this work is on GNN models.

Algorithm L.1 Uniform particle discretization

Require: center $\mathbf{x}_0$ + [0, b_0], radius R, ratio edge particles α=3, total particle count n

$\phi = (1+\sqrt{5})/2$ ▷ Golden ratio

$\theta_{\text{step}} = 2\pi\phi$ ▷ Step angle

$r = [0,1]$ ▷ Random float value from a uniform distribution

$\theta_{\text{offset}} = 2\pi r$ ▷ Generate a random offset angle

$b =$round$(\alpha\sqrt{n})$

for i in n **do**

  if $i > n - b$ **then**

    $r \leftarrow R$ ▷ Assign radius at the surface

  else

    $r \leftarrow R\sqrt{i-0.5}/\sqrt{n-(b+1)/2}$ ▷ Define a uniform radius

  end if

  $\theta \leftarrow i\theta_{\text{step}} + \theta_{\text{offset}}$ ▷ Append random offset to the angle

  $\mathbf{x} \leftarrow \mathbf{x}_0 + r\begin{bmatrix}\cos\theta \\ \sin\theta\end{bmatrix}$ ▷ Compute Cartesian coordinates

end for

M

GNS Fixed Model Hyperparameters

The GNS publication came with it extensive empirical investigation of varying the model hyperparameters and the resulting MSE value [85]. The focus of this thesis is not to repeat this hyperparameter investigation, but to leverage the formulated benchmark tests to understand to what degree the GNS model can meet the benchmark tests. Table M.1 lists model parameters kept constant in Chapter 3.

Table M.1: Motivation for GNS fixed model hyperparameters

Hyperparameter	Reason
`message passing architecture`	The model uses `Interaction Network`. The choice of message passing architecture is a research question alone [87, 96, 109].
`message passing steps`	As this is inherently linked to the `connectivity radius`, it is worth investigating. However, only if needed due to computational expense and already being investigated by the GNS paper.
`history size`	Conflicting reasoning here. There are conflicting results between the GNS paper and MeshGraphNets [103], which makes investigating it of interest. On the other hand the GNS paper already investigates it and has conclusions.
`sampling strategy`	This is a research question itself, with options presented by Rozemberczki et al. [101].
`neural network architecture`	Investigated in GNS paper and shows to have little effect on results beyond 128 neurons per layer @ 2 layers.
`Adam optimizer`	Assumed an adequate optimizer for the problem.
`learning rate`	Exponential learning rate decay from 10^{-4} to 10^{-6} is assumed sufficient for the problem.
`loss function`	The L_2 norm is used commonly for learning complex physics [90, 103, 152]. Notable variants do exist [86, 89].

N

Example Training and Validation Curves

For completeness, example training and validation curves are computed for the GNS model. These show that the training and validation lines reduce in unison over the course of training. No evidence of over- or under- fitting is seen. This is expected, as the training and validation sets are pulled from the same underlying distribution (ie the same benchmark setup).

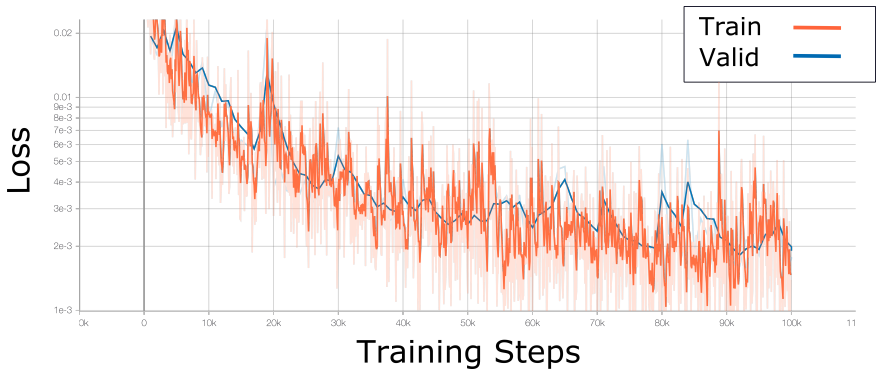


Figure N.1: Rolling disk experiment # 2.1

Figure N.2: Rolling disk experiment # 2.7