Applications of Machine Learning and Artificial Intelligence in Granular Mechanics

Project Completion Report

Sanskaar Srivastava Sidhant Thalor Sidharth Budania C Zoliansangi
Rohan Nimesh Harsh Nirmal

April 21, 2025

Overview

Granular Materials

- 75% of industrial feedstock is granular
- Discrete solid particles that interact through contact forces
- Complex behaviors like jamming, force chains, strain localization, pattern formation, and segregation
- Uniquely challenging to model
- Classical modeling techniques:
  - Continuum mechanics
  - Kinetic theory
  - Discrete element methods (DEM)
- Traditional methods either lack fidelity or are computationally expensive

Motivation & Context

- Need
  - computationally efficient methods especially to model large or dense systems
  - capture discrete particle interactions
  - uncover latent structures in disordered systems
  - overcome challenges faced by physics-based models
- Machine Learning (ML)
  - provides data-driven modeling that can improve speed and accuracy.
  - identify complex, nonlinear relationships without relying on explicit physical assumptions
  - enables transfer learning across materials, shapes, and loading conditions
  - can act as fast, accurate surrogates for real-time simulations and inverse tasks like property inference and optimization.

Goal

Review and demonstrate the potential of ML in the modeling of granular systems

Literature review

Historical Context

- Coulomb(1776) formulated early theories of soil shear strength
- Reynolds(1885) discovery of dilatancy
- Governed by constitutive laws
- Lacked the resolution to capture localized phenomena

Discrete Element Method (DEM)

- Cundall and Strack (1979)
- Modeling interactions by contact mechanics and Newtonian dynamics
- Gold standard for microscale modeling of granular flows
- Computationally intensive

Integration of Machine Learning in Granular Materials

Early ML

- Supervised learning models
- Estimating soil properties (CBR, MDD) using ANNs

Modern ML

- ML application categorized into three domains (Wang and Feng, 2024)
  - Microscopic particle interactions
  - Constitutive modeling of material behavior
  - Macroscopic engineering-scale simulations
- GNNs, CNNs, PINNs, U-Nets

Key Research Areas

Microscopic Particle Interactions

- ML replaces or enhances DEM contact models
- Neural networks predict contact forces, flow defects, or shock-sensitive particles
- Used for forecasting local rearrangements and failure zones

Constitutive Behavior Prediction

- ML learns stress-strain relations from experimental or DEM data
- Captures anisotropy, non-linearity, and path dependency
- Enhances geotechnical predictions: slope stability, bearing capacity

Macroscopic Simulations

- ML accelerates DEM/FEM solvers via surrogate models
- Enables real-time simulation by learning force laws or time-integration behavior.
- Classifies granular flows, estimates bulk properties from trajectories

Neural Network Architectures in ML-Driven Modeling of GM

7 commonly used ML models for simulating granular materials which are further classified into :

- Multi-Layer Perceptron MLP: Fast, lacks memory
- RNN/LSTM/GRU: Good for history-dependent behavior
- CNN/GNN: Capture spatial and interaction features
- TCNN: Efficient for sequential data with parallelism.

Neural Network Architectures in ML-Driven Modeling of GM

Single-Time Step Neural Networks: Multi-Layer Perceptron

- The output of a simple MLP with one hidden layer is given by:

$$O = g(\mathbf{W}_2 f(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

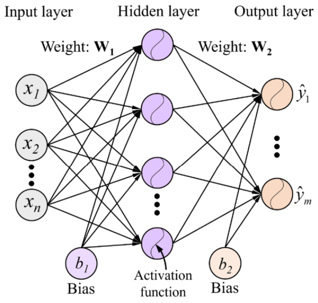


Figure: MLP with one hidden layer. Shows weight matrices ($\mathbf{W}_1, \mathbf{W}_2$) and biases ($\mathbf{b}_1, \mathbf{b}_2$).

MLP Implementation in PyTorch (1/3)

Data Loading and Preprocessing

```python
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from torch.utils.data import DataLoader, TensorDataset
import pandas as pd

# Load and preprocess data
df = pd.read_feather('E:/particledata')
df = df.dropna()
X = df.drop(['no_particles', 'packing_fraction'], axis=1).to_numpy()
y = df['packing_fraction'].to_numpy()

# Standardization
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Convert to tensors
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.float32).view(-1, 1)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

MLP Implementation in PyTorch (2/3)

MLP Model and Dataloaders

```
# DataLoaders for batching
train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=16, shuffle=True)
test_loader = DataLoader(TensorDataset(X_test, y_test), batch_size=16)

# Define the MLP model
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, 1)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        return self.fc3(x)

# Initialize model, criterion, and optimizer
model = MLP(X.shape[1], hidden_size=64)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

MLP Implementation in PyTorch (3/3)

Training, Evaluation, and Saving

```
# Training loop
for epoch in range(1000):
    for inputs, labels in train_loader:
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/1000], Loss: {loss.item():.4f}')

# Evaluation
model.eval()
with torch.no_grad():
    total_loss = 0
    for inputs, labels in test_loader:
        outputs = model(inputs)
        total_loss += criterion(outputs, labels).item()
    avg_loss = total_loss / len(test_loader)
    print(f'Test Loss: {avg_loss:.4f}')

# Save model
torch.save(model.state_dict(), 'mlp_model.pth')
print("Model saved to mlp_model.pth")
```

Neural Network Architectures in ML-Driven Modeling of GM

Single-Time Step Neural Networks: Multi-Layer Perceptron

- Fundamental equation:

$$O = g(\mathbf{W}_2 f(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

Variable	**Description**	**Dimension**
$\mathbf{x}$	Input feature vector	$C_{in} \times 1$
$\mathbf{W}_1$	Input-to-hidden weights	$H \times C_{in}$
$\mathbf{b}_1$	Hidden layer bias	$H \times 1$
f	Hidden layer activation (ReLU)	-
$\mathbf{W}_2$	Hidden-to-output weights	$C_{out} \times H$
$\mathbf{b}_2$	Output layer bias	$C_{out} \times 1$
g	Output activation (Linear/Sigmoid)	-

- Characteristics: Simple feedforward model lacking temporal memory

Neural Network Architectures in ML-Driven Modeling of GM

Multi-step-based or time-sequence networks:

A. Recurrent Neural Networks (RNN)

$$h_t = \tanh(Wx_t + Uh_{t-1} + b), \quad y_t = g(Vh_t + c)$$

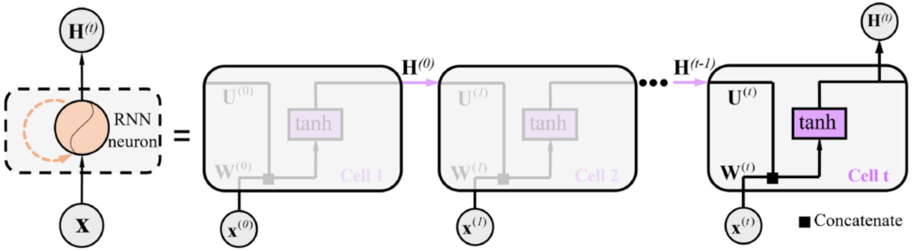


Figure: The recurrent neuron in the basic RNN

Neural Network Architectures in ML-Driven Modeling of GM

A. Recurrent Neural Networks (RNN)

$$h_t = \tanh(Wx_t + Uh_{t-1} + b), \quad y_t = g(Vh_t + c)$$

Variable	**Description**	**Dimension**
x_t	Input vector at time t	$C_{in} \times 1$
h_t	Hidden state at time t	$H \times 1$
W	Input-to-hidden weights	$H \times C_{in}$
U	Hidden-to-hidden weights	$H \times H$
b	Hidden layer bias	$H \times 1$
V	Hidden-to-output weights	$C_{out} \times H$
c	Output layer bias	$C_{out} \times 1$
tanh	Hidden activation	-
g	Output activation (e.g., softmax)	-

Key Characteristics:

- Maintains historical context via hidden state h_t
- Suffers from vanishing gradients in long sequences

Neural Network Architectures in ML-Driven Modeling of GM

B. Long Short-Term Memory (LSTM)

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \odot \tanh(C_t)
\end{aligned}
$$

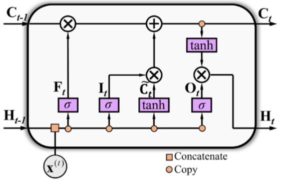


Figure: LSTM memory cell

Neural Network Architectures in ML-Driven Modeling of GM

B. Long Short-Term Memory (LSTM)

$$
\begin{aligned}
f_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
i_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
o_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{C}_t &= f_t \odot \mathbf{C}_{t-1} + i_t \odot \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{h}_t &= o_t \odot \tanh(\mathbf{C}_t)
\end{aligned}
$$

Variable	**Description**	**Dimension**
$\mathbf{x}_t$	Input vector at time t	$C_{in} \times 1$
$\mathbf{h}_t$	Hidden state vector	$H \times 1$
$\mathbf{C}_t$	Cell state vector	$H \times 1$
f_t, i_t, o_t	Forget/input/output gates	$H \times 1$
σ	Sigmoid activation	-
$\odot$	Hadamard product	-

Neural Network Architectures in ML-Driven Modeling of GM

C. Gated Recurrent Unit (GRU)

$$z_t = \sigma(W_z x_t + U_z h_{t-1}), \quad r_t = \sigma(W_r x_t + U_r h_{t-1})$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tanh(W x_t + U(r_t \odot h_{t-1}))$$

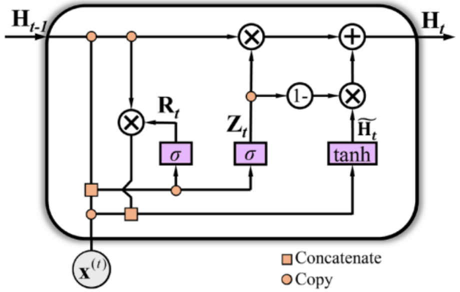


Figure: GRU cell architecture

Neural Network Architectures in ML-Driven Modeling of GM

D. Temporal Convolutional Neural Networks (TCNN)

$$O(t) = \sum_{k=0}^{K-1} W_k X(t-k)$$

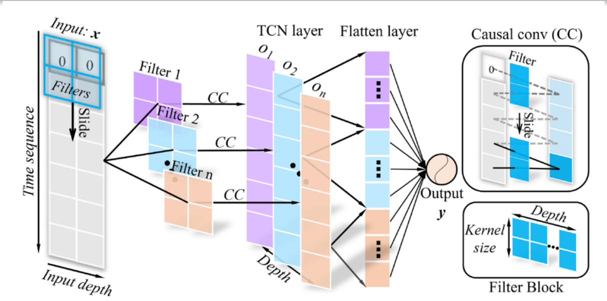


Figure: The feedforward process in the temporal convolution neural network

Geometry Information-Based Neural Networks for Granular Modeling

A. Convolutional Neural Networks (CNN)

$$\mathbf{A}_{\text{out}}(i,j,k) = \sum_{m=0}^{C_{\text{in}}-1} \sum_{p=0}^{K_h-1} \sum_{q=0}^{K_w-1} \mathbf{W}(p,q,m,k)\cdot \mathbf{A}_{\text{in}}(i+p,j+q,m) + \mathbf{b}(k)$$

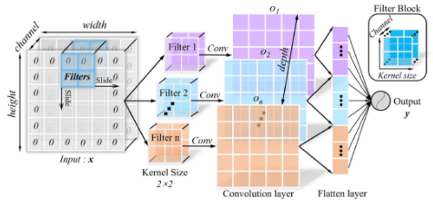


Figure 6: The feedforward process of the convolutional neural network

Figure: CNN feedforward process for granular assemblies

Geometry Information-Based Neural Networks for Granular Modeling

B. Graph Neural Networks (GNN)

Message passing framework for particle assemblies:

$$\mathbf{h}_i^{(k)} = \phi\left(\mathbf{h}_i^{(k-1)}, \bigoplus_{j\in\mathcal{N}(i)} \psi\left(\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}, \mathbf{e}_{ij}\right)\right)$$

Edge update mechanism:

$$\mathbf{m}_{ij}^{(k)} = \mathrm{MLP}\left(\mathbf{h}_i^{(k-1)} \| \mathbf{h}_j^{(k-1)} \| \mathbf{r}_{ij} \| \mathbf{f}_{ij}\right)$$

Geometry Information-Based Neural Networks for Granular Modeling

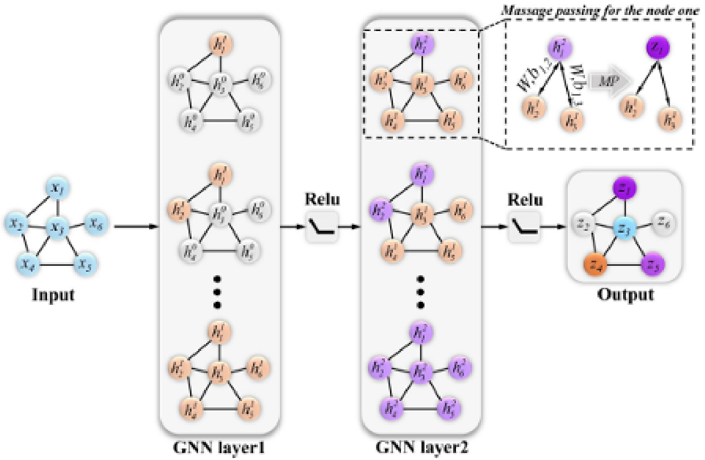


Figure 7: Architecture of Graph Neural Networks

Neural Network Selection Guidelines

Key Selection Criteria

- Input structure: Grid (CNN), Sequence (RNN/LSTM), Graph (GNN)
- Temporal context: Short (MLP), Long (TCNN/LSTM)
- Compute resources: Light (MLP), Heavy (GNN/TCNN)

Network	**Strengths**	**Limitations**
MLP	Simple, fast training	No temporal memory
RNN	Sequential processing	Gradient issues
LSTM	Long-term dependencies	Complex architecture
TCNN	Parallel processing	Fixed context window
CNN	Spatial features	Grid data requirement
GNN	Irregular topologies	Dynamic graph challenges

ML-Aided Microscopic Modeling: Two Major Directions

A. ML-Based Contact Models

- Incorporate contact states and geometric features into particle-based simulations.
- Directly predict contact features (e.g., contact point, normal, overlap) from positions.
- Bypass traditional contact detection/resolution, improving DEM efficiency.
- Capture particle shape effects and generalize across shapes.

Advantages:

- High computational efficiency.
- Shape-awareness.
- User-friendly (less reliance on complex geometry algorithms).

Limitations:

- Depends on completeness and quality of training data.
- Training data often based on simplified/empirical contact models.

ML-Aided Microscopic Modeling: Grain-Level Kinematics

B. Grain-Level Kinematic Feature Models (CNNs/GNNs)

- Predict grain motion/acceleration from geometric state (positions, velocities).
- Avoid explicit force calculations, accelerating simulations.
- Suitable for both sparse and dense systems.

Advantages:

- Lower computational complexity.
- High efficiency for large systems.

Limitations:

- Error accumulation over time.
- Often ignore rotational dynamics.
- GNNs less suited for systems with dynamic particle count.

Discrete Element Method

- Developed by Cundall & Strack(1979)
- Capable of Describing mechanical behaviour of Particles, modelling based on Newton's Laws of motion.
- Method based on use of numerical scheme of monitoring particles.
- Involves a rigorous calculation cycle
- State of each particle can be known at a given time.

Discrete Element Method(Disadvantages?)

- Computationally Expensive.
- Could only model a limited number of geometries.
- Modelling irregular particles is a tedious and computationally inefficient process.

Don't these algorithms discussed uptil now take particles state as input and 'predict' the next one?

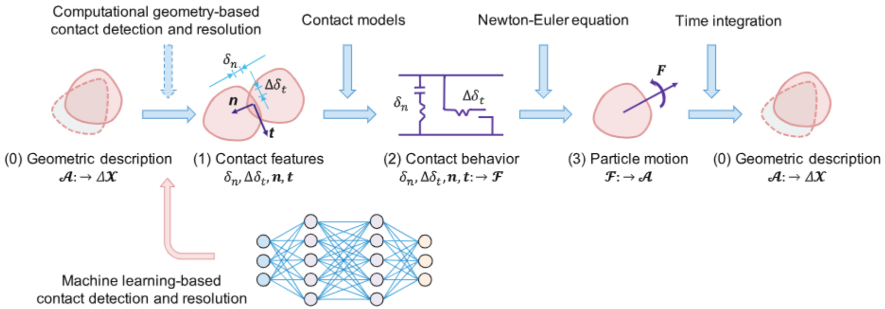
DEM + ML Integration

- Deep learning useful in tacking collision.
- ML encompasses set of tools and algorithms for this process.
- Though much less explored in DEM, can significantly increase computation.
- ML used for contact detection (classification) and resolution (regression).

Examples

BFS-based layering, GNNs for force chain prediction

DEM + ML Integration



Computational geometry-based contact detection and resolution
Contact models
Newton-Euler equation
Time integration
δ_n
$\Delta\delta_t$
$\boldsymbol{n}$
$\boldsymbol{t}$
δ_n
$\Delta\delta_t$
$\boldsymbol{F}$
(0) Geometric description
$\mathcal{A}: \to \Delta\mathcal{X}$
(1) Contact features
$\delta_n, \Delta\delta_t, \boldsymbol{n}, \boldsymbol{t}$
(2) Contact behavior
$\delta_n, \Delta\delta_t, \boldsymbol{n}, \boldsymbol{t}: \to \mathcal{F}$
(3) Particle motion
$\mathcal{F}: \to \mathcal{A}$
(0) Geometric description
$\mathcal{A}: \to \Delta\mathcal{X}$
Machine learning-based contact detection and resolution

Figure: Calculation Cycle in a DEM

DEM + ML Integration

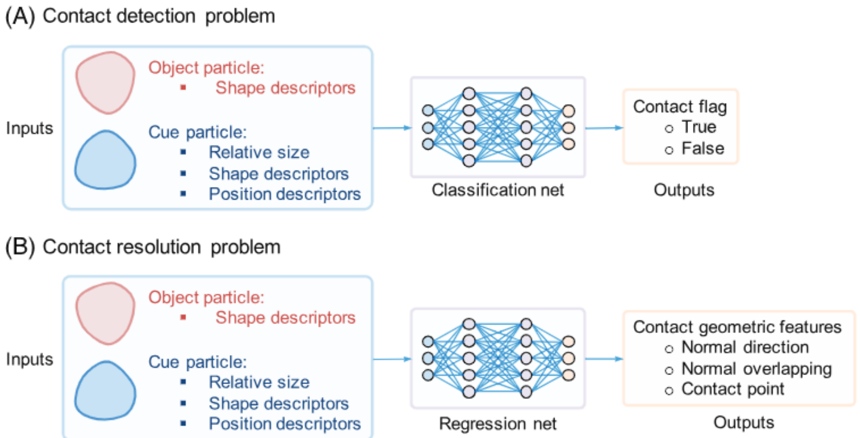


Figure: DEM+ML Integration

Aboria: High-Performance Particle Simulation Framework

- C++ library for particle-based numerical methods (DEM, SPH, MD)
- Provides STL-compatible particle containers with:
    - N-dimensional spatial embedding
    - Custom particle attributes (mass, charge, etc.)
    - Unique particle IDs and metadata
- Accelerated spatial queries using:
    - Cell lists for dense systems
    - Kd-trees for sparse configurations
    - Hyper-octrees for adaptive resolution

Key Features for DEM/SPH:

- Kernel operator API for force calculations:

$$F_{ij} = \sum_{j \in \mathcal{N}(i)} \psi(\mathbf{x}_i, \mathbf{x}_j, \mathbf{v}_i, \mathbf{v}_j)$$

- Flexible neighborhood queries with p-norm metrics
- GPU acceleration through Kokkos backend
- ML integration via PyTorch/TensorFlow bindings

Aboria Applications in Granular Simulations

DEM-SPH Coupling

- Unified particle container for mixed discrete/continuum systems
- Resolved fluid-particle interactions using SPH kernels:

$$\mathbf{f}_i^{fluid} = \sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) \nabla W_{ij}$$

- Automatic contact detection between DEM particles

- Kernel operators replaceable with neural networks
- Particle attributes can store ML model outputs
- Enables:
  - Learned contact models
  - Neural network force predictors
  - Differentiable simulations

Macroscopic Simulations

- **Enhanced Simulation Efficiency:** ML is coupled with Discrete Element Method (DEM) to accelerate time integration, enabling larger time steps without sacrificing accuracy.
- **Surrogate Models:** Learned force models or timestep-stability criteria from small simulations reduce computational costs.
- **Multi-scale Modeling:** Neural networks trained on DEM data are embedded into continuum solvers (e.g., FEM[Finite Element Method], MPM[Material Point Method]) as constitutive laws for hybrid simulations that balance accuracy and scalability.
- **Material Classification:** ML classifiers applied to kinematic data from granular flows can categorize materials based on properties like stiffness, size distribution, or restitution.
- **Latent Property Inference:** Even simple neural networks can infer hidden material properties from motion trajectories, revealing structure-to-behavior mappings in granular systems.

Simulating the Brazil Nut Effect

- **Objective:** Model the Brazil Nut Effect (BNE)—the rise of a large particle in a vibrated granular bed—using a 2D discrete particle simulation.
- **System:** 60 circular particles in a 1x1 unit box; 59 small ($r_s = 0.04$), 1 large "Brazil nut" ($r_b = 0.15$).
- **Forcing:** Two vibration modes:
  - Sinusoidal tapping (smooth, periodic): $y_{\text{base}}(t) = 0.03\cos(14\pi t)$
  - Dirac delta tapping (instantaneous impulse): $a(t) = 250\,\delta(t - 20)$
- **Initialization:** Particles are randomly placed horizontally, with increasing vertical positions to avoid overlap; Brazil nut starts near the base.
- **Particle Attributes:**
  - Mass ($\propto r^2$) (2D System)
  - Velocity & Acceleration
  - Position Vectors

Simulating the Brazil Nut Effect

- **Layered Contact Detection:** Used Breadth-First Search (BFS) to assign a layer index to each particle, modeling realistic force propagation from the vibrating base.
- **Packing Fraction:** Calculated in the bottom 25% of the box using circular segment geometry for partial overlaps.
- **Key Observables:** Height of the Brazil nut and local packing fraction over time.

Vibration Modes: Sinusoidal vs Dirac Delta Tapping

Sinusoidal Tapping

$$y_{\text{base}}(t) = A\cos(2\pi f t)$$

- Smooth, continuous energy input
- Promotes gradual rearrangement and convection

Dirac Delta Tapping

$$a(t) = a \cdot \delta(t - t_0)$$

- Instantaneous, strong impulse
- Useful for isolating effects of discrete perturbations
- $\delta(t - t_0)$ is the Dirac delta function

Motion, Collisions

Particle Motion

$$v_{y,i}^{t+1} = v_{y,i}^{t} + a_i \Delta t, \qquad y_i^{t+1} = y_i^{t} + v_{y,i}^{t+1} \Delta t$$

Variable	**Description**
$v_{y,i}^{t}$	Vertical velocity of particle i at time t
a_i	Net acceleration (gravity + contacts)
y_i^{t}	Vertical position of particle i at time t

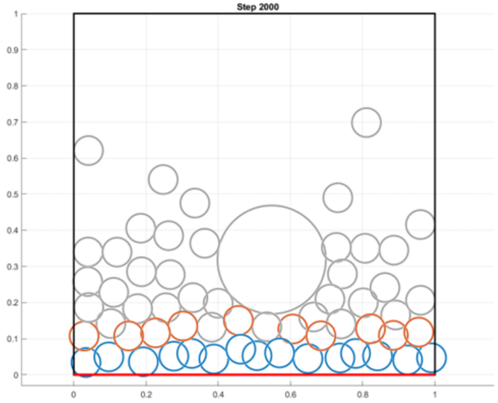
Collision Resolution

$$J = \frac{2 m_1 m_2}{m_1 + m_2} (\vec{v}_1 - \vec{v}_2) \cdot \hat{n}$$

Variable	**Description**
J	Impulse magnitude
m_1, m_2	Masses of colliding particles
$\vec{v}_1, \vec{v}_2$	Pre-collision velocities

Layered Contact Detection with BFS

- **Why?**
  - Calculation of next state was inaccurate.
  - A hierarchy in the state update was observed.
  - Updates more accurate when computed layerwise.



Layered Contact Detection with BFS

- **Goal:** Model how vibration energy propagates upward through particle contacts.
- **Method:** Assign each particle a "layer" index using Breadth-First Search (BFS) on the contact graph.
- **Mathematical Formulation:**

$$\ell(v_i) = \begin{cases} 0 & \text{if } y_i - r_i \leq y_{\text{base}} + \epsilon \\ k & \text{if } v_i \text{ contacts } v_j \text{ with } \ell(v_j) = k - 1 \\ -1 & \text{otherwise} \end{cases}$$

- **Physical Relevance:** Only base-contacting particles get full input; higher layers move via transferred momentum.

BNE Simulation in MATLAB

BFS Based Layer Identification

```
function layer_index = compute_layered_contacts(x, y, radii,base_y)
    num_particles = length(x);
    epsilon = 1e-4;
    % Initialize all particles as unassigned
    layer_index = -1 * ones(num_particles, 1);
    %Assigning Layer 0 (touching the floor)
    current_layer = 0;
    layer_particles = find(y - radii <= base_y+epsilon);
    layer_index(layer_particles) = current_layer;
    % Iteratively building layers
    while ~isempty(layer_particles)
        next_layer_particles = [];
        for i = layer_particles'
            for j = 1:num_particles
                if layer_index(j) == -1
                    dx = x(i) - x(j);
                    dy = y(i) - y(j);
                    dist = sqrt(dx.^2 + dy.^2);
                    if dist <= radii(i) + radii(j) + epsilon
                        layer_index(j) = current_layer + 1;next_layer_particles(end+1) = j;
                    end
                end
            end
        end
        current_layer = current_layer + 1;
        layer_particles = next_layer_particles;
    end
end
```

Motion, Collisions, and Packing Fraction

Packing Fraction

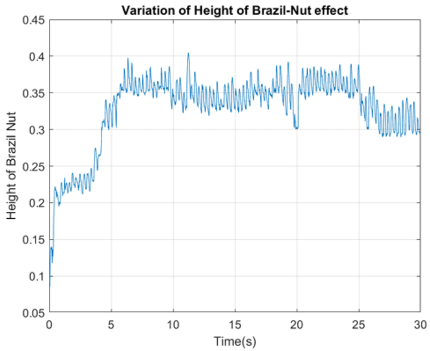
$$\phi = \frac{1}{A_{\text{region}}} \sum_{i \in \text{region}} A_{\text{overlap},i}$$

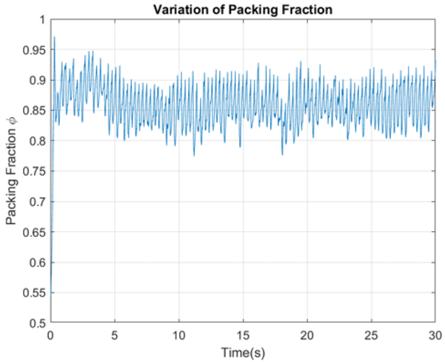
Variable	Description
ϕ	Local packing fraction
A_{region}	Area of bottom 25% region
$A_{\text{overlap},i}$	Overlap area of particle i

- Computed using circular segment geometry for partial overlaps

Results: Brazil Nut Height and Packing Fraction

- **Observation:** With periodic tapping, the Brazil nut rises gradually—height curve follows a decaying exponential.
- **Sinusoidal Tapping:** Smooth rise and gradual rearrangement.
- **Dirac Delta Tapping:** More abrupt changes, useful for isolating rearrangement effects.





Key Takeaways

- Layered BFS contact detection models realistic energy propagation in granular media.
- Both sinusoidal and Dirac delta tapping induce the Brazil Nut Effect, but with different rearrangement dynamics.
- The simulation framework allows for analysis of particle motion, packing, and segregation under various forcing conditions.
- Approach can be extended to study other granular phenomena and test different driving protocols.

Key Findings: Network Architectures

A. Network Architecture Tradeoffs

$$
\begin{aligned}
\text{TCNN Accuracy} &= 89\% \text{ (Vibration Analysis)} \\
\text{TCNN Memory} &= 40\% \text{ more than GRU} \\
\text{MLP Speedup} &= 3\times \text{ vs GNNs} \\
\text{MLP Error} &= 40\% \text{ higher in irregular systems} \\
\text{Receptive Field} &= (K-1)\times 2^{L-1}+1
\end{aligned}
$$

Figure: TCNN architecture and receptive field expansion

Key Findings: Contact Detection and Data

B. Layered Contact Detection

$$\ell(v_i) = \begin{cases} 0 & y_i - r_i \leq y_{\text{base}} + \epsilon \\ k & \exists v_j \in \mathcal{N}(v_i), \ell(v_j) = k - 1 \end{cases}$$

BFS-based algorithm reduces contact detection errors by 62%.

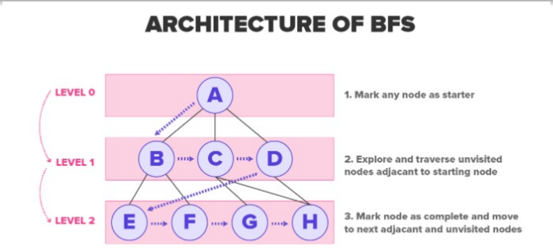

ARCHITECTURE OF BFS
LEVEL 0
LEVEL 1
LEVEL 2
A
B
C
D
E
F
G
H
1. Mark any node as starter
2. Explore and traverse unvisited nodes adjacent to starting node
3. Mark node as complete and move to next adjacent and unvisited nodes

Figure: Layer propagation in BFS contact detection

Key Findings: Brazil Nut Effect and Simulation Metrics

C. Brazil Nut Effect Insights

$$\Gamma = \frac{GravitationalAcceleration}{Gravity} = \frac{(2\pi f)^2 A}{g} > 1 \qquad (\Gamma = 5.92 \text{ in simulation})$$

Packing fraction overestimation: up to 112% of RCP due to overlaps.
Gamma here is a dimensionless parameter

D. Implementation Metrics

- ML-DEM speedup: 40–60%
- Error accumulation: 15% per 100 steps
- CNNs require $>$10,000 images for robust training

Key Findings: Physics-Informed ML and Hybrid Models

E. Hybrid Physics-ML Frameworks

$$\mathcal{L}_{\text{physics}} = \|\nabla \cdot \sigma - \rho g\|^2 \qquad \text{(29\% accuracy gain with PINNs)}$$

Differentiable DEM reduces inverse problem time by 73%.

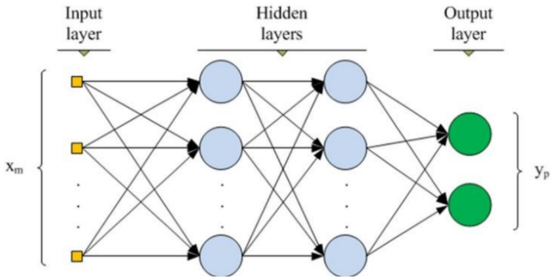


Figure: PINN framework integrating conservation laws

Key Findings: ML in Granular Mechanics

Architecture Performance

Network	Strengths	Limitations
MLP	Simple static modeling	No temporal memory
RNN/LSTM	Cyclic loading	Gradient issues
GNN	Contact networks	Dynamic graphs
TCNN	Vibration analysis	Fixed context

Microscopic Modeling Advances

- 40-60% DEM acceleration via ML contact detection
- CNN/GNN kinematic predictors reduce iteration needs
- 78% studies ignore rotational dynamics

Critical Research Gaps & Future Directions

Current Limitations

- Error accumulation: 15%/100 steps
- Rotation neglect in 78% models
- $\Gamma > 5$ systems understudied
- Industrial-scale validation pending

Implementation Roadmap

- Standardized benchmarks
- Transfer learning frameworks
- Real-time digital twins
- Uncertainty quantification

Methodological Priorities

- Hybrid PINN architectures
- Active learning for rare events
- Quaternion-based rotation models

Key Insight

ML enables granular digital twins but requires physics-informed architectures and multiscale validation for industrial adoption.

Case Study: Brazil Nut Effect Simulation

ML-DEM Implementation

- Layered BFS contact detection
- $\Gamma = \frac{(2\pi f)^2 A}{g} = 5.92$
- $\Delta t = 0.015\text{s}$ ($10\times$ smaller than period)

Key Results

- Height decay $R^2 = 0.96$
- 15% ϕ fluctuations
- Dirac delta vs sinusoidal segregation patterns

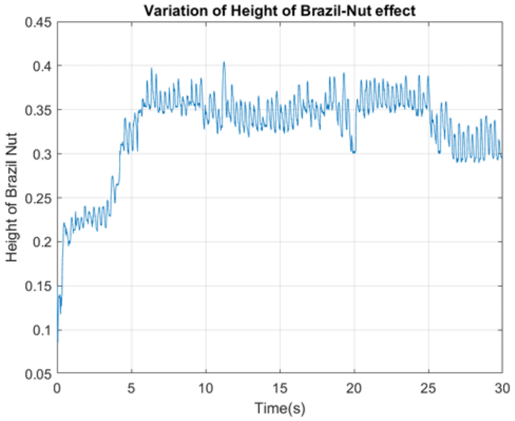


Figure: Brazil nut height vs time under sinusoidal tapping

References

Reference List

Due to the extensive number of references, the complete bibliography is provided in the project report. For detailed citations and further reading, please refer to the accompanying documentation.

- A full BibTeX file (`reference.bib`) is available with all sources.
- Key references are highlighted throughout the presentation.

The End