Efficient n-body simulations using physics informed graph neural networks

Víctor Ramos Osuna
Departmento de Sistemas Informáticos
E.T.S.I Sistemas Informáticos – Universidad Politécnica de Madrid
C. de Alan Turing, s/n, 28031 Madrid, Spain
victor.ramos.osuna@upm.es

Alberto Díaz-Álvarez
Departmento de Sistemas Informáticos
E.T.S.I Sistemas Informáticos – Universidad Politécnica de Madrid
C. de Alan Turing, s/n, 28031 Madrid, Spain
alberto.diaz@upm.es

Raúl Lara Cabrera
Departmento de Sistemas Informáticos
E.T.S.I Sistemas Informáticos – Universidad Politécnica de Madrid
C. de Alan Turing, s/n, 28031 Madrid, Spain
raul.lara@upm.es

Abstract

This paper presents a novel approach for accelerating N-bodies simulations by integrating a physics-informed graph neural networks (GNNs) with traditional numerical methods. Our method implements a leapfrog-based simulation engine to generate datasets from diverse astrophysical scenarios which are then transformed into graph representations. A custom-designed GNN is trained to predict particle accelerations with high precision. Experiments, conducted on 60 training and 6 testing simulations spanning from 3 to 500 bodies over 1000 time steps, demonstrate that the proposed model achieves extremely low prediction errors—loss values while maintaining robust long-term stability, with accumulated errors in position, velocity, and acceleration remaining insignificant. Furthermore, our method yields a modest speedup of approximately 17% over conventional simulation techniques. These results indicate that the integration of deep learning with traditional physical simulation methods offers a promising pathway to significantly enhance computational efficiency without compromising accuracy.

1 Introduction

A N-bodies system is a problem in computational physics that studies the evolution of N particles under mutual gravitational interactions, formulated based on Newton's equations. Its analytical solution is only possible for $N = 2$, so for larger values various numerical methods are employed. These methods, despite being capable of high precision, are very costly due to the need to compute every interaction between each pair of particles at every iteration Aarseth and Aarseth (2003). This has led to the search for alternatives that can accelerate or emulate simulations without significantly compromising accuracy.

XVI XVI Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (maeb 2025).

Over the past two decades, the rise of deep learning (DL) techniques has opened up new possibilities for modelling these complex systems. In particular, graph neural networks (GNNs) have proven to be an especially suitable approach for working with graph-structured data, as they are capable of capturing both local and global relationships among the nodes. This is evidenced by various classification and regression tasks in physical problems, where they have been applied by leveraging their ability to model complex interactions in dynamic systems Battaglia et al. (2018). Since gravity is fundamentally a long-range force, for computational efficiency we adopt a local interaction approximation –a strategy similar to that employed in hierarchical methods like Barnes-Hut Barnes and Hut (1986)–. This assumption allows the GNN to focus on local interactions while still approximating the global dynamics of the system, albeit at the risk of oversimplifying the true nature of gravitational forces.

In this work, we propose an approach that combines the generation of simulation data from an N-bodies system with a regression model based on GNNs. The methodology presented consists of three main modules:

1. **Physical simulation and data generation**: A simulation engine employing a leapfrog integration scheme Iserles (1986) iteratively updates the positions, velocities, and accelerations of particles. It generates diverse scenarios (e.g. random distributions, 3D discs, multi-disc systems) to create a dataset containing both particle attributes and their target accelerations.
2. **Transformation to graph representation**: The simulation data is converted into graphs, with each particle represented as a node (including attributes such as position) and edges established based on proximity.
3. **GNNs-based regression model**: A graph neural network based on the *EdgeConv* architecture Wang et al. (2019) propagates and transforms node information to predict particle accelerations. The model includes encoders for nodes and edges, mapping initial features into a higher-dimensional space, and an output layer tailored for the regression task.

The **primary objective** of this work is to **validate the ability of the GNN model to predict accelerations in N-bodies systems with sufficient precision**, demonstrating that the integration of realistic physical simulation with DL can substantially improve the simulation time of complex dynamics while maintaining a high degree of fidelity to reality.

The remainder of the paper is organised as follows. Section Related works reviews recent work on the application of DL in computational physics. This is followed by Sections Methodology and Experiments and results, which detail the structure, execution, and results of the experiments. Finally, the paper concludes with Section Conclusion, which presents the findings and outlines potential future directions for this work.

2 Related works

The simulation of N-bodies systems has been a central area in computational physics, particularly in astrophysics. The earliest work addressing these systems dates back to the second half of the 1980s, when direct methods were used to compute the gravitational interactions between every pair of particles. Although these algorithms allowed for the simulation of small systems with great precision, their high computational complexity –$O(N^2)$– increased rapidly with the number of particles, prompting the development of more efficient algorithms. Among these, the Barnes-Hut method Barnes and Hut (1986) stands out as a hierarchical approach to approximate gravitational forces, reducing the complexity to $O(NlogN)$–, while the fast multipole method (FMM) Greengard and Rokhlin (1987) –also hierarchical, but approximating gravitational effects by region or cluster rather than evaluating every individual interaction—achieves a complexity of $O(N)$–.

At the same time, various numerical integration schemes have been developed to simulate the temporal evolution of these systems. The leapfrog integrator is perhaps the most well-known. Introduced in 1967, it has been extensively studied over the following decades, leading to numerous variants of its basic operation Iserles (1986). It is also considered one of the most popular methods due to its ability to preserve physical properties, such as energy conservation, throughout the system's evolution. The implementation of these methods in simulators has enabled the modelling of both stellar systems and the formation and evolution of galaxies Hernquist (1990); Navarro et al. (1997), with the latter often utilising different density profiles to describe the mass distribution.

With the popularisation of machine learning (ML) in the 2010s, many traditional numerical methods have been combined with techniques from the field of artificial intelligence. For instance, Sánchez-González et al.Sanchez-Gonzalez et al. (2020) demonstrated that the use of GNNs enables the extraction of latent representations that encapsulate crucial information about the dynamics of systems composed of a large number of particles. These representations, by capturing the essential information from interactions at different scales, facilitate the prediction of trajectories and accelerations without the need to explicitly compute every interaction at each iteration. This strategy not only drastically reduces the computational burden but also makes it feasible to apply these techniques in scenarios where direct simulation is prohibitive, such as in the modelling of diffusion processes, turbulence, or granular dynamicsRaissi et al. (2019); Pfaff et al. (2020). Moreover, research in related fields—such as fluid dynamics and materials mechanics—has confirmed that learning latent representations through GNNs offers a scalable approach to tackling problems with very high dimensionality and complex nonlinear relationships Maurizi et al. (2022); Sergeev et al. (2024).

On the other hand, the intrinsic capability of GNNs to solve regression tasks in physical domains has been widely explored. Recent studies have harnessed these networks to predict continuous properties such as forces, energies, and accelerations, demonstrating their effectiveness in inferring relevant physical quantities Battaglia et al. (2016); Kipf and Welling (2016). One of the principal strengths of this neural network architecture in such tasks lies in its ability to integrate the topological structure and inherent connectivity of the data, which is essential for capturing the essence of interactions in N-bodies systems. This feature allows, for example, the precise inference of variations in force fields from the spatial distribution of particles, adapting to all kinds of configurations Gilmer et al. (2017). Additionally, it has been shown that the generalisation of these models facilitates their application in various contexts, ranging from the prediction of properties in simulations of composite materials Chen et al. (2023) to the modelling of complex biological systems Yang et al. (2022); Bongini et al. (2022).

Therefore, we can conclude that while traditional models remain essential for the development of N-bodies simulations, the literature supports the notion that integrating DL techniques—and in particular, GNNs—offers a means to alleviate the complexity of these methods, making them more scalable for simulations that require a very large number of interacting elements. Rather than replacing classical numerical methods (e.g., finite difference or finite element methods), GNNs act as efficient computational shortcuts that complement existing simulation techniques, providing data-driven approximations that reduce computational costs without compromising the underlying physical modeling.

3 Methodology

We will divide this section into the four main stages that have comprised this study: (i) generation of a dataset from a physical N-bodies simulation, (ii) transformation of the raw dataset into a graph structure, (iii) definition of a GNN architecture for a regression problem, that is, the prediction of each body's acceleration, and (iv)training and evaluation of the model.

All experiments were executed on a workstation equipped with an AMD Ryzen 5 5600X 6-Core Processor (6 cores, 12 threads) running Ubuntu GNU/Linux 24.04, with a frequency of approximately 4650 MHz, and 32 GB of system memory. The machine uses an NVIDIA GeForce RTX 2070 graphics card with 8 GB of dedicated memory, running under CUDA 12.4 with driver version 550.120.

3.1 Data generation from physical simulation

The foundation of the methodology is a simulation engine that integrates the evolution of N-bodies systems. It begins with a set of initial conditions defined in terms of the particles' positions, velocities, and masses. The simulation engine employs a leapfrog integration scheme, which guarantees stable numerical integration and preserves system invariants (e.g. total energy) over time.

3.1.1 Gravitational acceleration formula

For each particle i in a system of N particles, the acceleration is computed by summing the contributions from all particles j (excluding i, as is evident) as follows:

$$\vec{a_i} = G \sum_{j=1, j \neq i}^{N} m_j \frac{\vec{r_j} - \vec{r_i}}{\left(\|\vec{r_j} - \vec{r_i}\|^2 + \epsilon^2\right)^{3/2}} \tag{1}$$

where G is the gravitational constant, m_j is the mass of particle j, $\vec{r_i}$ and $\vec{r_j}$ are the positions of particles i and j respectively, and ϵ is the smoothing parameter used to avoid singularities. The pseudocode is presented in Algorithm 1.

Algorithm 1 N-Bodies simulation using leapfrog method

```
procedure NBodySimulation(R, V, M, dt, T, G, ε)
    A ← Accelerations(R, M, G, ε)
    for t ← 1 to T do
        for i ← 1 to N do
            V[i] ← V[i] + A[i] × dt/2
        end for
        for i ← 1 to N do
            R[i] ← R[i] + V[i] × dt
        end for
        A ← Accelerations(R, M, G, ε)
        for i ← 1 to N do
            V[i] ← V[i] + A[i] × dt/2
        end for
        Store state (R, V, A)
    end for
end procedure
```

In this algorithm, the function `Accelerations` is responsible for implementing the formula from Equation 1 for each particle.

3.1.2 Dataset generation

The dataset is generated from multiple simulations by varying the initial parameters (i.e. random distribution, $3D$ disc, multi-disc), organising the information into three sets:

- **Positions** $\{\vec{r_t}\}$ for each frame t.
- **Features**: These may include additional information (e.g. mass).
- **Regression labels** corresponding to the accelerations $\{\vec{a_t}\}$.

Additionally, the approach includes the possibility of incorporating temporal information by shifting the position sequence, which is achieved by concatenating previous frames as part of the feature vector for each time step.

3.2 Transformation to graph structure

The dataset is structured in a tabular format, but to capture the inherent relational structure of these N-bodies systems, we will transform it into a graph-based structure. To achieve this:

1. Each particle is represented as a **node**, with its initial feature being a vector that includes its position, its mass, and other data derived from its temporal evolution.
2. These nodes will be connected by **edges** based on a k-nearest neighbors (KNN) criterion, where each node is connected to its k closest neighbors in terms of Euclidean distance. The parameter k has been experimentally defined. Optionally, edge attributes may be computed, for example, the magnitude of the position difference: $e_{ij} = \|\vec{r_i} - \vec{r_j}\|$. However, in this work, no attributes have been incorporated into the edges.

Algorithm 2 describes the process by which each frame of the dataset is transformed into a graph.

Algorithm 2 Frame to graph transformation using KNN

procedure GRAPHTRANSFORMATION(R, X, Y, k)
  Initialize:
  $Nodes \leftarrow \emptyset$
  $Edges \leftarrow \emptyset$
  for $i \leftarrow 1$ **to** N **do**
    Add node i with feature $X[i]$ to $Nodes$
  end for
  for $i \leftarrow 1$ **to** N **do**
    Find the k nearest neighbors of node i based on Euclidean distance
    for each neighbor j of i **do**
      Add edge (i, j) to $Edges$
      Calculate $e \leftarrow \|R[i] - R[j]\|$ and assign as (i, j) attribute
    end for
  end for
  Build graph G from $Nodes$ and $Edges$, and assign Y tags
  return G
end procedure

3.3 Model architecture and inference

The model architecture is structured into three main modules: (i) the initial transformation of input features, (ii) the iterative propagation of information through message passing layers, and (iii) the generation of the final output for the regression task.

It has been designed to be modular, allowing experiments with different types of aggregation functions –such as mean or maximum–modifying the depth of the network, or incorporating attention mechanisms to weight the messages from neighbouring nodes differently.

All the implementation is based on PyTorch and PyTorch Geometric, leveraging vectorised operations and GPU acceleration, which is essential when training with large datasets and/or models.

Algorithm 3 describes the inference process through the proposed model.

Algorithm 3 Inference process

procedure FORWARDPASS(G, L)
  for node i in G **do**
    $h[i] \leftarrow \text{NodeEncoder}(X[i])$
  end for
  for $l \leftarrow 1$ **to** L **do**
    for node i in G **do**
      $m[i] \leftarrow 0$
      for neighbour j of i **do**
        $m[i] \leftarrow m[i] + \phi\big(h[i], h[j], E[i, j]\big)$
      end for
      $h[i] \leftarrow \text{LayerNorm}\left(\text{concat}\left(h[i], m[i]\right)\right)$
    end for
  end for
  for node i in G **do**
    $\hat{a}[i] \leftarrow \text{Linear}(h[i])$
  end for
  return $\{\hat{a}[i]\}_{i=1}^{N}$
end procedure

3.3.1 Initial features transformation

Before message propagation begins, each node in the graph is transformed via an encoder—based on an multilayer perceptron (MLP)—that projects the original features into a d-dimensional space,

thereby facilitating the capture of non-linear relationships. The MLP employs a Tanh activation function to preserve both positive and negative values in the transformed features.

In other words, if $\vec{x_i} \in \mathbb{R}^{d_{in}}$ is the input feature vector for node i, then the encoder produces:

$$h_i^{\vec{(0)}} = f_{\text{node}}(\vec{x_i}), \quad f_{\text{node}} : \mathbb{R}^{d_{in}} \to \mathbb{R}^d \tag{2}$$

This process may incorporate regularization through dropout to stabilize training, although it is not employed in the present work.

Similarly, although it is not used in this paper, it is possible to enable a similar encoder via a configuration parameter to map edge attributes into a space compatible with dimension d. That is, if $\vec{e_{ij}}$ is the attribute associated with the edge between nodes i and j, then:

$$\vec{z_{ij}} = f_{\text{edge}}(\vec{e_{ij}}), \quad f_{\text{edge}} : \mathbb{R}^{d_e} \to \mathbb{R}^d \tag{3}$$

As in the node encoder, a Tanh activation function can be applied in f_{edge} to maintain the full range of values. This allows the additional information regarding the relationships between nodes to be integrated coherently with the node representations.

3.3.2 Message passing and aggregation

The architecture is built upon message passing layers based on *EdgeConv* Wang et al. (2019), a variant of graph convolutions that eases the integration of topological information from the nodes.

Message function For each node i, a function ϕ is defined that combines the information from i with that of each of its neighbors j. This function can be implemented using an MLP and takes into account both the current representations $h_i^{\vec{(l)}}$ and $h_j^{\vec{(l)}}$, as well as the edge attributes $\vec{z_{ij}}$—when available. The combination is expressed as:

$$\vec{m_{ij}^{(l)}} = \phi\left(h_j^{\vec{(l)}}, h_i^{\vec{(l)}}, \vec{z_{ij}}\right) \tag{4}$$

The function ϕ is designed to be invariant to the order of neighbors and to capture complex interactions. In our implementation, the MLP employs a Tanh activation function to maintain negative values in the message representations, ensuring that the transformations preserve important directional information.

Aggregation and update The messages from all neighbours are aggregated using a summation function. The choice of this function is due to the properties it exhibits in the GNNs literature Gilmer et al. (2017); Xu et al. (2018).

$$\vec{m_i^{(l)}} = \sum_{j \in \mathcal{N}(i)} \vec{m_{ij}^{(l)}} \tag{5}$$

Where $\mathcal{N}(i)$ denotes the nodes in the neighborhood of node i. Next, the node is updated via a residual operation that concatenates the previous representation and the aggregated message, followed by a LayerNorm operation:

$$h_i^{(\vec{l+1})} = \text{LayerNorm}\left(\text{concat}\left(h_i^{\vec{(l)}}, \vec{m_i^{(l)}}\right)\right) \tag{6}$$

This residual update—$h_i^{\vec{(l)}}$—helps preserve information across layers, while LayerNorm ensures stable training dynamics by normalizing the concatenated features.

Regarding depth and receptive field The number of layers L is a critical hyperparameter: as L increases, each node is able to access information from more distant neighbours in the graph, which is essential for capturing long-range interactions in N-bodies systems. However, an excessive number of layers may lead to over-smoothing Li et al. (2020), meaning that the node representations become overly homogeneous and the model's ability to differentiate between nodes is reduced, compounded by an increase in computational load that negates the original efficiency benefits.

3.3.3 Output layer and mapping to the regression task

The final module of the architecture is responsible for converting the representations obtained after the message passing layers into the desired output.

Each final node i has a representation $\vec{h_i^{(L)}}$ which is transformed by a linear layer that maps the embedded space to a vector of dimension d_{out}. In our case, for the prediction of accelerations in a $3D$ space, $d_{\text{out}} = 3$:

$$\vec{\hat{a}_i} = W_{\text{out}} \vec{h_i^{(L)}} + \vec{b_{\text{out}}} \tag{7}$$

where W_{out} and $\vec{b_{\text{out}}}$ are the parameters of the output layer.

3.4 Training and evaluation

After defining the dataset—through graphs derived from the raw simulation data—and the GNN architecture, the training and evaluation process aims to optimize both the model's predictive capability and generalization for the problem. Algorithm 4 describes the training process of the model.

Algorithm 4 Training process

```
procedure TRAINGNN(D, E, B)
    for epoch ← 1 to E do
        Shuffle D
        Divide D into B batches
        for batch in B do
            loss ← 0
            for graph G in batch do
                â ← ForwardPass(G)
                loss ← loss + MSE(â, Y)
            end for
            Backpropagate loss
            Update weights (ADAM)
        end for
    end for
    Final test evaluation
end procedure
```

The dataset has been divided with an approximately 90%-10% split, where 90% of the data is used for training and 10% for testing. No validation set was used. The loss function employed is the mean squared error (MSE) between the model's predicted accelerations and the actual accelerations.

For the training process, the Adam optimizer is used to update the model parameters. The implemented training cycle consists of:

- **Batch extraction**: Iterating over the training dataset in batches, where each batch contains a set of graphs along with their corresponding labels.
- **Inference**: For each batch, the model performs a forward pass through the graph, generating acceleration predictions $\vec{\hat{a}_i}$ for each node.
- **Loss calculation**: The MSE is computed between the predictions and the actual accelerations.
- **Backpropagation**: The loss is backpropagated and the model weights are updated.

4 Experiments and results

The complete source code for all experiments and simulations presented in this paper is available at `https://github.com/KNODIS-Research-Group/efficient-n-body-simulations-using-physics-informed-gnn`. This repository contains all the scripts, data preprocessing routines, and model implementations required to reproduce the results described here.

For the training of the model, a dataset was created that includes 60 random simulations, distributed as 10 simulations each of 3, 25, 50, 100, 250, and 500 bodies, all executed over 1000 iterations. The model evaluation was performed using 6 simulations, each corresponding to one of the aforementioned numbers of bodies –3, 25, 50, 100, 250, and 500– also over 1000 steps.

The simulations were carried out by modelling spiral galaxies that include a black hole. The simulation parameters are described in Table 1.

Table 1: Simulation parameters and their respective values

Parameter	**Value**
Time step	0.0001
Gravitational constant	4.5×10^{-6}
Total galaxy mass	1.0
Radial scale	3.0
Vertical scale	0.3
Black hole mass proportion	0.01
Arms	2

These parameters have been adjusted based on the total mass of the galaxy, in an effort to simulate the initial state of a galaxy in a dimensionless system in a relatively realistic manner. However, absolute fidelity was not the aim, as the focus of this paper is on the simulation of N-bodies systems; thus, while these values are representative, they do not correspond to any actual configuration of a theoretical or known galaxy.

4.1 Stepwise analysis

The results obtained from the model were grouped by scene. The data show that the loss is extremely low, with values on the order of 10^{-14} to 10^{-13}. Figure 1 summarises the results for the loss and the average step times used during the simulation execution.

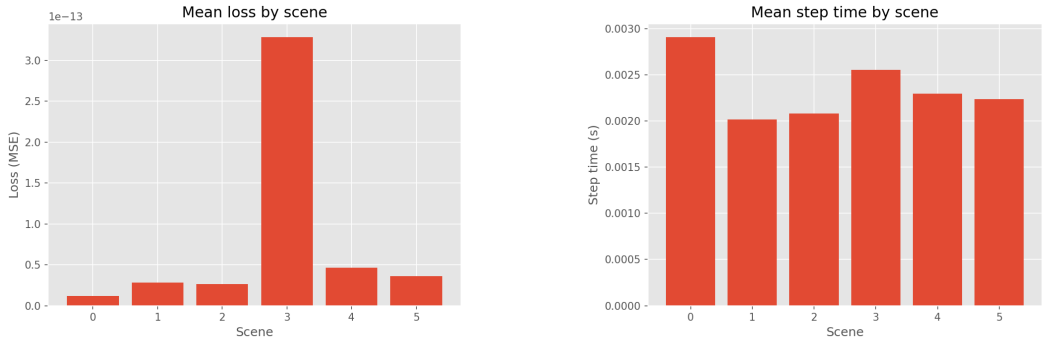


Figure 1: Results of the error analysis (left) and average scene execution time (right) for 1000 simulation steps performed using the model.

These results indicate that the model achieves more than acceptable accuracy and consistency in its predictions at each step.

4.2 Rollout evaluation

Errors for position, velocity, and acceleration of all particles have been averaged across scenes for each step. The acceleration error remains almost constant (with only minimal variation at the sixth decimal place), which indicates that the prediction of acceleration is very stable over time. In contrast, the errors in position and velocity start at zero and gradually increase, reaching values on the order of 10^{-21} and 10^{-17} respectively by the final step of the simulation, thereby demonstrating good control over error propagation in these variables. Figure 2 illustrates the evolution of these metrics over time.

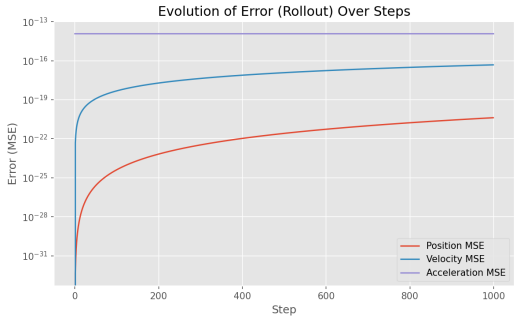
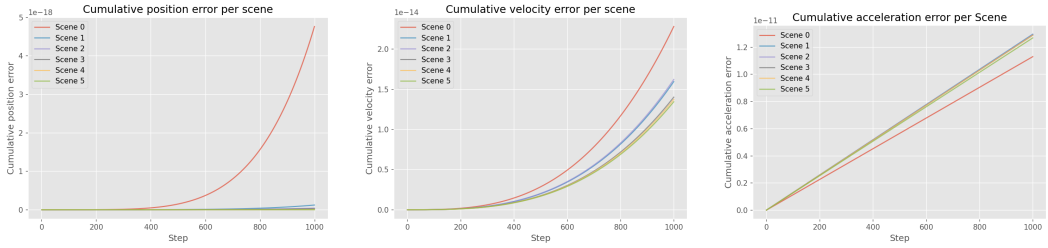


Figure 2: Evolution of the MSE in position, velocity, and acceleration over 1000 simulation steps. The value at each step is the average of the values for all particles across all scenarios for that step.

In the figure, one can observe the complete stability of the acceleration predictions and an almost imperceptible increase in the errors in position and velocity. Focusing on the accumulated errors (see Figure 3), which are obtained by summing the errors at each step, it reveals that:



(a) (b) (c) c

Figure 3: Accumulated errors in position (left), velocity (centre) and acceleration (right). The error accumulation is shown for each of the 6 scenes, from the one with the fewest particles (scene 0, 3 bodies) to that with the most particles (scene 5, 500).

1. The accumulated error in the position variable at the end of the simulation is on the order of 10^{-18}, with a final average of approximately 8.27×10^{-19}.
2. The accumulated error in velocity is on the order of 10^{-14}, with a final average of approximately 1.6×10^{-14}.
3. The accumulated error in acceleration reaches values on the order of 10^{-11}, with a final average of approximately 1.26×10^{-11}.

These low error values indicate that the error propagation over time is practically negligible.

4.3 Regarding computational efficiency

A speedup of approximately 1.17 times has been achieved, that is, the model runs roughly 17% faster than the base simulation. Although moderate, it represents a promising avenue for further research, particularly as it has been realised without sacrificing the accuracy of the predictions.

5 Conclusion

Following the analysis of the results obtained in this study, it appears that integrating GNNs techniques with physical N-bodies simulations can accelerate their execution while maintaining a high degree of accuracy, with very low losses and negligible accumulated errors.

It should be noted that the dataset was divided into a training-test split without a dedicated validation set. This decision was primarily driven by the focus on establishing baseline performance. However, the absence of a validation set is a limitation, as it restricts our ability to monitor overfitting. Future work will aim to incorporate a separate validation set or adopt cross-validation techniques to enhance performance evaluation and robustness.

Another relevant consideration is that the black hole is treated uniformly as any other particle, which may inadequately capture its dominant gravitational influence. A differentiated treatment—such as assigning a higher weight or designing a specialized connectivity scheme—should be explored in future work to better reflect its unique role in the system. Additionally, further improvements to the implementation may enhance the model's performance and make it even more competitive compared to traditional simulation techniques.

Finally, although this study focuses on small- to medium-scale systems (up to 500 bodies), real-world astrophysical simulations often involve thousands or even millions of particles. Addressing scalability is therefore a key avenue for future research. Possible approaches include using sparse graph representations, hierarchical modeling strategies, or hardware acceleration to extend the applicability of the method to larger and more complex systems.

Acknowledgments and Disclosure of Funding

The support of the Comunidad Autónoma de Madrid under ALENTAR-J-CM project (reference TEC-2024/COM-224) is gratefully acknowledged.

References

Aarseth, S. J. and Aarseth, S. J. (2003). *Gravitational N-body simulations: tools and algorithms*. Cambridge University Press.

Barnes, J. and Hut, P. (1986). A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449.

Battaglia, P., Pascanu, R., Lai, M., Jimenez Rezende, D., et al. (2016). Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29.

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

Bongini, P., Pancino, N., Scarselli, F., and Bianchini, M. (2022). Biognn: how graph neural networks can solve biological problems. In *Artificial Intelligence and Machine Learning for Healthcare: Vol. 1: Image and Data Analytics*, pages 211–231. Springer.

Chen, S., Wulamu, A., Zou, Q., Zheng, H., Wen, L., Guo, X., Chen, H., Zhang, T., and Zhang, Y. (2023). Md-gnn: A mechanism-data-driven graph neural network for molecular properties prediction and new material discovery. *Journal of Molecular Graphics and Modelling*, 123:108506.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

Greengard, L. and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348.

Hernquist, L. (1990). An analytical model for spherical galaxies and bulges. *Astrophysical Journal, Part 1 (ISSN 0004-637X), vol. 356, June 20, 1990, p. 359-364.*, 356:359–364.

Iserles, A. (1986). Generalized leapfrog methods. *IMA Journal of Numerical Analysis*, 6(4):381–392.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Li, G., Xiong, C., Thabet, A., and Ghanem, B. (2020). Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*.

Maurizi, M., Gao, C., and Berto, F. (2022). Predicting stress, strain and deformation fields in materials and structures with graph neural networks. *Scientific reports*, 12(1):21834.

Navarro, J. F., Frenk, C. S., and White, S. D. (1997). A universal density profile from hierarchical clustering. *The Astrophysical Journal*, 490(2):493.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. (2020). Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. (2020). Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR.

Sergeev, P., Kiselev, D., and Makarov, I. (2024). Graph neural networks for complex fluid simulations. In *2024 IEEE 28th International Conference on Intelligent Engineering Systems (INES)*, pages 000145–000152. IEEE.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Yang, X., Wang, W., Ma, J.-L., Qiu, Y.-L., Lu, K., Cao, D.-S., and Wu, C.-K. (2022). Bionet: a large-scale and heterogeneous biological network model for interaction prediction with graph convolution. *Briefings in bioinformatics*, 23(1):bbab491.