PREDICTION OF THE ATOMIC CAUCHY STRESS TENSOR FOR A SIMULATED MOLECULAR SYSTEM USING DEEP LEARNING

SUMMER PROJECT REPORT

Agney K Rajeev

ID : SRP240772

Department of Biological Sciences

Indian Institute of Science Education and Research - Kolkata

May 14^{th}, 2024 - July 19^{th}, 2024

ABSTRACT

This study describes the development and analysis of models for computing the stress tensor in a water box simulation. The stress tensor computation is inherently difficult, with a time complexity of $O(n^4)$, where n is the number of atoms. Our dataset includes 5001 instances, with each frame of simulation comprising of 4275 atoms and each having 8 input and 6 output features. We use MultiLayer Perceptrons (MLP) and Graph Neural Networks (GNN) to model the stress tensor based on atomic positions, velocities, and one-hot atom type encodings. Despite its ability to recognise non-linear patterns, the MLP overfits and does not generalise well to the test set. In contrast, GNN models, which are intended to capture local atomic interactions via message passing, perform better with more balanced training and test results. Our findings show that GNNs are an interesting approach to stress tensor computing in molecular dynamics simulations, although more enhancements are required to attain optimal performance metrics.

Acknowledgements

I would like to express my gratitude to Dr. Neelanjana Sengupta for providing me with the opportunity to work on this exciting project. I deeply value the time and resources she dedicated to support me and my project. I would like to thank Pallab Dutta for his advice and supervision. The guidance he provided through our numerous conversations was absolutely invaluable. I would also like to thank IISER Kolkata for the invitation to their Summer Research Program providing the systemic support required for this project to be possible. Last but not least, I'd like to express my gratitude to all my lab mates and fellow interns all of whom have assisted me in completing this project directly or indirectly.

Contents

1 Introduction

The atomic Cauchy stress tensor is a quantity which provides comprehensive insights into the distribution of internal stresses within a molecular system. It is essential for the calculation of other mechanical properties of a system such as self-diffusivity, shear viscosity etc[1]. Traditional approaches for estimating the stress tensor often call for extensive processing resources and have limited scalability, particularly for large-scale simulations.

In recent times, combining computational methods and machine learning has created new opportunities for predicting and analyzing molecular systems[2][3]. Simulations are frequently used to study and predict the features and behaviors of molecular systems, and they have long been a foundation of biomolecular research. Techniques like molecular dynamics (MD) simulations[4] provide detailed insights into atomic and molecule structures and interactions. However, these techniques can be computationally expensive and time-consuming, particularly for large and complicated systems. On the other hand, machine learning, particularly deep learning, has achieved great success in a variety of disciplines by effectively capturing complex patterns and correlations in massive datasets. Deep learning models excel in learning data representations that can be applied to a wide range of tasks, including image identification[5] and natural language processing[6]. The combination of these two strategies has resulted in unique approaches to forecasting transport properties. Machine learning models can be trained on massive volumes of simulation data to identify underlying trends and anticipate new, unknown systems. This approach has substantial advantages in terms of speed and scalability, allowing researchers to investigate a wider range of systems and situations.

This study intends to use deep learning to create a predictive model that can accurately estimate the atomic Cauchy stress tensor from molecular dynamics simulations. By combining advanced neural network architectures with molecular simulation data, we want to build a strong and efficient framework that minimizes processing costs while maintaining accuracy.

The proposed approach comprises training a deep learning model on a large dataset derived from molecular dynamics simulations. This dataset will comprise a variety of molecular configurations and their stress tensor values, offering an abundance of information for the model to learn from. Once trained, the model will be able to predict the stress tensor for novel molecular configurations, allowing for quick and scalable stress analysis.

By correctly predicting the atomic Cauchy stress tensor using deep learning, this study hopes to contribute by providing a strong tool for speedy and accurate stress analysis. The

findings of this study have the potential to speed up the computation of additional transport parameters and improve our understanding of molecular-scale interactions and behaviors.

This report is organised as follows: section 2 provides the details regarding the molecular system under study and its properties used to form the dataset, section 3 details the deep learning model architectures used in the study. The hardware, software and performance metrics for the setup are presented in section 4 and numerical results in section 5. We conclude this paper and discuss its future scope in section 6.

2 The System

This section provides the details regarding the molecular system used for the simulations and how the Cauchy stress tensor is computed. The layout of the dataset generated from the simulation data is also mentioned.

2.1 Water Box

For this study, we consider a system of 1425 water molecules (4275 atoms) distributed across a periodic cubic box of length $35.22734\ A^o$. The system was simulated for 500 picoseconds with a time step of 0.1 picoseconds resulting in a trajectory of 5001 frames. The positions and velocities of each atom as well as their stress tensor components were then computed using the $MDAnalysis$ package. The first frame of the simulation is provided in fig.1

2.2 The Cauchy Stress Tensor

The stress at every point inside a material can be defined using the Cauchy stress tensor ($\boldsymbol{\tau}$). $\boldsymbol{\tau}$ is a second order tensor with nine components τ_{ij} where $i, j \in x, y, z$. It relates a surface perpendicular to a given vector $\boldsymbol{e}$ to the traction vector ($\boldsymbol{T}$) across the surface as follows.

$$\boldsymbol{T} = \boldsymbol{\tau} \ . \ \boldsymbol{e} \tag{1}$$

$$T_j = \sum_i \tau_{ij} \ e_i \tag{2}$$

Note that $\boldsymbol{\tau}$ is a symmetric tensor ($\tau_{ij} = \tau_{ji}$) and hence only has six independent components. For a molecular system, these stress components for the i^{th} atom can be calculated form the masses(m) and velocities($\boldsymbol{v_i}$) of each atom and the distances($\boldsymbol{r_{ij}}$) and net forces($\boldsymbol{f_{ij}}$) between each pair of atoms using the following equation[1]

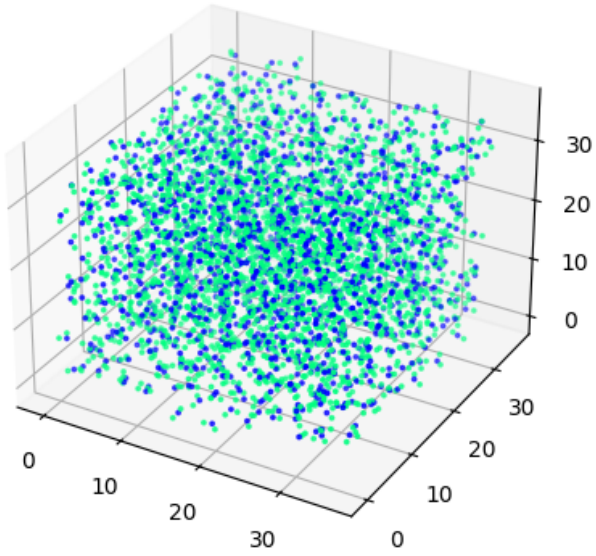

Figure 1: First frame of the water box simulation. Blue represents Oxygen atoms and Green represents Hydrogen atoms.

$$(\tau_i)_{\alpha\beta} = m(v_i)_\alpha(v_i)_\beta + \sum_j (r_{ij})_\alpha (f_{ij})_\beta \tag{3}$$

From eqn.3, we can see that the computation of the stress tensor requires a calculation of the interacting forces between each pair of atoms which is an $O(n^2)$ computation along with a summation across all atoms implying the computation of one stress component for a single atom to be of $O(n^3)$ and that for the entire system to be $O(n^4)$ where n is the number of atoms in the system. This time complexity could be a significant detriment to stress computation by restricting the scalability of the system.

2.3 Dataset

We create our dataset from the trajectory simulated for the water box. The positions and velocities of each atom are treated as input features along with a one-hot identifier for each of hydrogen and oxygen. One frame of simulation is taken as one training/testing instance for the model. Since the Cauchy stress tensor is symmetric, there are 6 independent stress components. These are taken as the output features. Thus our dataset consists of 5001

inputs with $4275 \times 8 = 34200$ input features and $4275 \times 6 = 25650$ output features. The distribution curves for the input and output features are provided in fig.2 and fig.3.

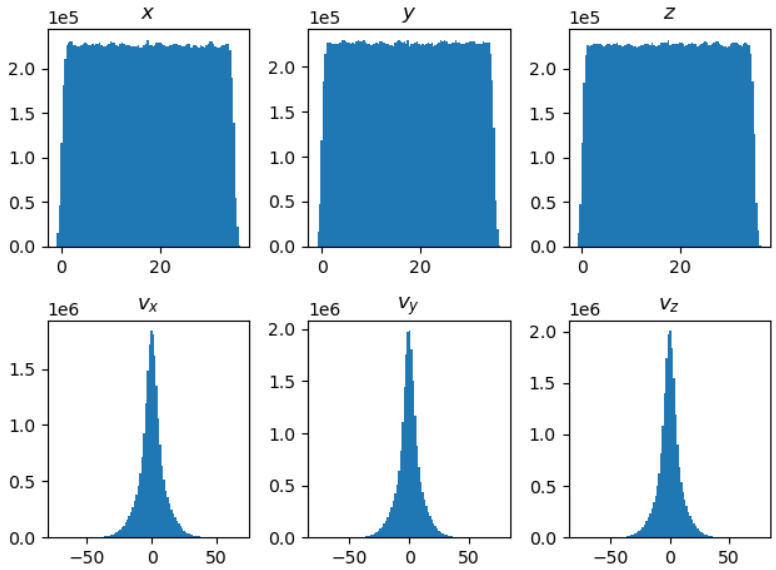

Figure 2: Number distribution of position(x, y, z) and velocity(v_x, v_y, v_z) components of the dataset

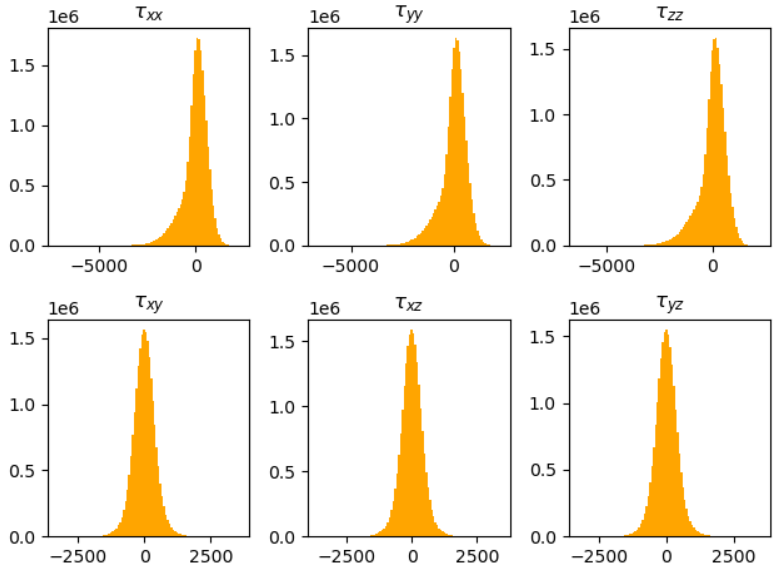

Figure 3: Number distribution of the independent stress components of the Dataset

We observe that the positions of the atoms follow a uniform distribution whereas the velocities follow a normal distribution. This is expected as the atoms satisfy the Boltzmann condition for statistical systems. We normalize the positions with respect to the box length

of the simulation and standardize the velocities w.r.t their variances inorder to balance the features.

In the output features, the off-diagonal terms of τ (shear terms) follow a normal distribution whereas the diagonal terms of τ (pressure terms) follow a skewed normal distribution which is skewed towards the negative axis. All stress terms where normalized w.r.t their variances.

The addition of more features such as mass, charge, permittivity etc. where considered, however, since only two types of atoms (hydrogen and oxygen are present in the water box), these would effectively provide only as much information as the one-hot encodings.

3 Models

In this section we discuss the architecture and hyperparameters of the various models implemented on the dataset.

We employ two types of models : MultiLayer Perceptron(MLP) and Graph Neural Networks(GNN)[7]

3.1 MultiLayer Perceptron

A MultiLayer Perceptron or MLP is an extension of Rosenblatt's Perceptron[8] which is understood to be the precursor of deep learning models. It consists of multiple fully connected layers arranged linearly and has the ability to recognize non-linear patterns in datasets. The primary layer is known as the input layer, the final layer is the output layer and all layers in between are the hidden layers.

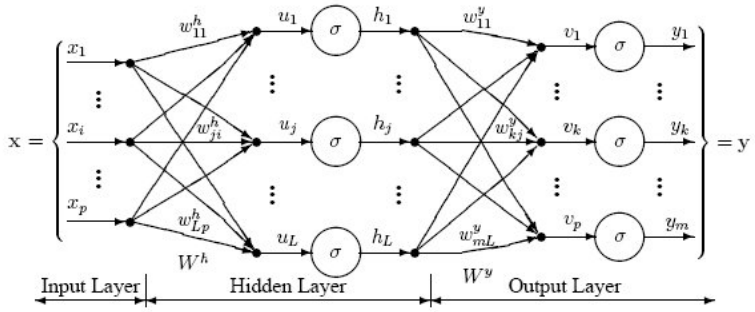

Figure 4: Schematic of a MultiLayer Perceptron

We use a 5 layer perception with 4 fully-connected networks on our dataset. The input layer consists of a linear arrangement of the input features of all atoms of a frame (34200) and the output layer consists of a similar arrangement of output features(25650). This helps the

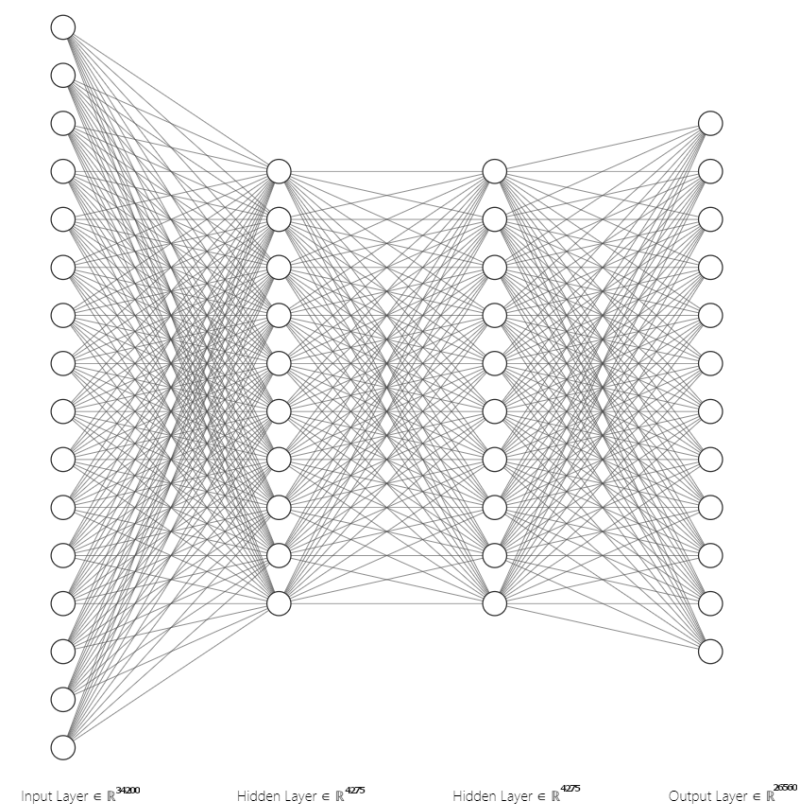

Figure 5: The architecture of the MLP used (figure not to scale)

model recognize cross-atom interactions as opposed to a model which takes each individual atom as input. The hidden layers are of size 4275 and ReLU activations are applied between each layer.

3.2 Graph Neural Networks

Graph Neural Networks or GNNs are deep learning models designed to run on data which can be represented as a graph data structure.

A graph is a data structure consisting of a set of nodes of which certain nodes are connected with each other via a set of edges. The nodes which are connected to a particular node via an edge are called the neighbours of that node. A GNN can take the graph as the input along with a set of features for the nodes(node features) and output a similar graph with a different set of node features as output. Such a task is known as node regression. The GNN accomplishes this by a process called *Message Passing* where the node features of all neighbours of a node are aggregated to produce the node features of a particular node. This allows the model to account for interactions between nodes.

In our dataset we can assume each frame as an individual graph were each atom is a node and spatially close atoms to be connected by edges. If we provide the positions, velocities and one-hot encodings of the atoms as the input node features, we can expect a GNN to provide the stress components as the output features.

The advantages of such an approach are (1) Since the stress tensor of an atom is a local property. By connecting the spatially close atoms using edges, there can be an aggregation of the local properties of the water box via message passing. (2) The GNN accounts for the permutation invariance of the graph where changing the order of the neighbours of an atom do not change the output. (3) By not connecting every node feature to every other node feature Ala MLP, the size of the model can be reduced allowing for larger hidden features and general scalability.

We design our GNN around the Graph Attention Network which uses an attention layer[9] for the message generation. We define a GAT block as a GAT layer[10] along with a skip connection using fully connected layers for each atom to preserve the contribution of individual atoms to their own stress as well as to prevent vanishing gradients. An ELU activation[11] is then used for non-linearity. The block diagram for the architecture for one GAT block used is provided in fig.6.

We create two types of Graphs and networks.

- A graph in which all atoms within a $15A^o$ radius of an atom are connected and a corresponding GNN consisting of a single GAT block with hidden dimension 64 followed by a fully connected layer.

- A graph in which all atoms within a $5A^o$ radius of an atom are connected and a corresponding GNN consisting of three GAT blocks with hidden dimension 64 followed by a fully connected layer.

Both approaches have an effective radius of interaction of $15A^o$ which is the standard radius for simulations.

4 Experimental Setup

Here we provide the hardware and software packages used for the experiment as well as the performance metrics and other auxiliary functions used in training and evaluation.

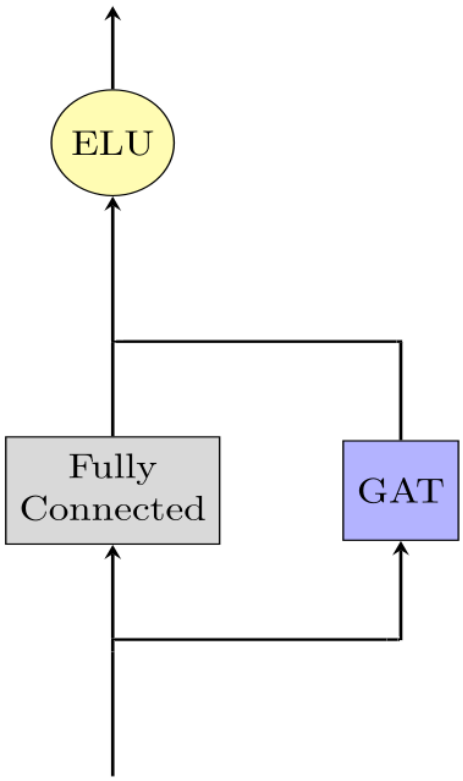

Figure 6: Block diagram of a GAT block

4.1 Hardware

Neural Network training was executed on a Linux server equipped with AMD Ryzen 9 7950X 16-Core Processor @ 3.00GHz, 16 GB RAM, and 1 NVIDIA GeForce RTX 3070 with 8GB VRAM

4.2 Software

The *PyTorch* library was used for designing the training and evaluation pipeline for the model. The *PyG* ecosystem of PyTorch was used for generating and loading graph data for the model as well as for creating the Graph Attention Layers. The *scikit-learn* library provided the evaluation metrics for benchmarking the models

4.3 Evaluation Metrics

The mean squared error loss function (MSELoss) was used as the loss function for the training as it is a standard fo regression tasks. The Adam optimizer was used for the

gradient descent with weight decay used wherever appropriate. Dropout of 20% was used after the first activation function for all models to minimize overfitting. The performance metrics used for evaluation as the R^2 score(R2), Mean Squared error(MSE) and the Mean Absolute Error(MAE).

5 Results and Discussion

The loss plots for the MLP, GNN with $5A^o$ cutoff and GNN with $15A^o$ cutoff are presented in figures 7, 8 and 9.

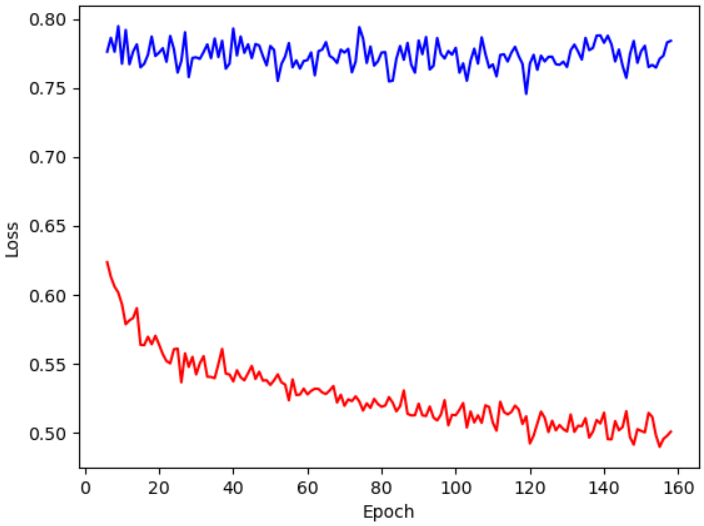

Figure 7: Epoch vs Loss plot for MLP (Training Loss in Red and Test loss in Blue)

The performance metrics for each model on the training and test dataset are provided in table1

Table 1: Performance of each model w.r.t each metric on the training as well as test dataset

	R2		MSE		MAE	
	Train	Test	Train	Test	Train	Test
MLP	0.49	0.19	0.521	0.665	0.517	0.774
GNN(5A°)	0.25	0.25	0.734	0.736	0.648	0.646
GNN(15A°)	0.25	0.24	0.735	0.736	0.647	0.646

We can observe that the MLP performs well quite well on the training set but fails to carry any significant generalization to the test set leading to large amounts of overfitting.

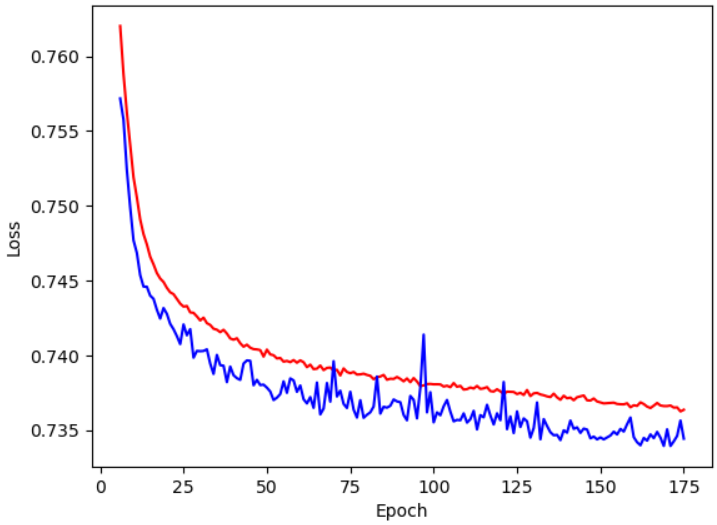

Figure 8: Epoch vs Loss plot for GNN with $5A^o$ cutoff (Training Loss in Red and Test loss in Blue)

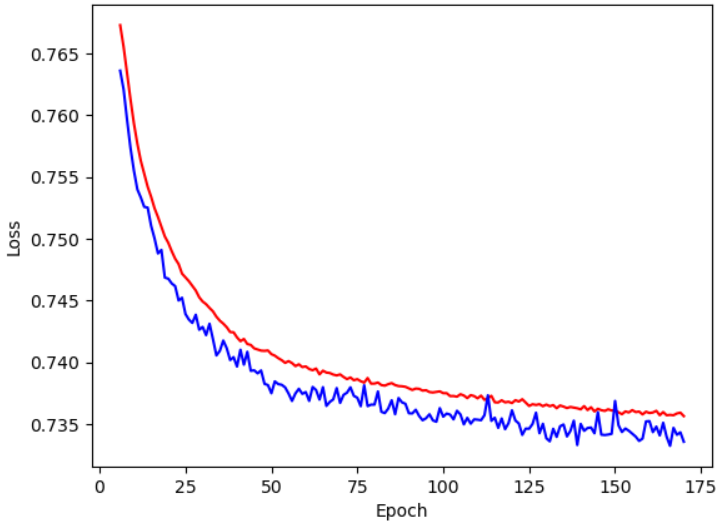

Figure 9: Epoch vs Loss plot for GNN with $15A^o$ cutoff (Training Loss in Red and Test loss in Blue)

Even after considerable use of dropout layers and weight decay, the test remained constant without improvement. This must imply that the MLP is unable to learn any pattern in the data and simply resorts to memorizing the input. This could be due to the lack of permutation invariance and as such simply changing the order of two atoms resulting in different predictions. There is also a problem of lack of scalability for the hidden dimension as the input sizes (34200) are huge for a training instance.

Both GNNs produce very similar results as well as similar convergence patterns. This is not surprising as the effective radius of both models are $15A^o$. However, the graph with a cutoff of $5A^o$ contains just 200,000 edges wheres as with a cutoff of $15A^o$ has 5,900,000. Working with a smaller cutoff yields faster results even with larger number of message passing layers. The GNN shows higher R2 value and lower MSE and MAE than the MLP for the test set. They also produce a similar convergence trend for both training and test sets. However, the R2 value of 0.24-0.25 is still not ideal for a task such as this.

6 Conclusions and Future Work

In this report, we analyze the potency of deep learning models in the computation of atomic Cauchy stress tensor for a water box system. We introduced two different architectures functioning in different ways as well as variations of the GNN architecture to aid in the construction of a suitable model. We benchmarked each model on the basis of 3 performance metric and discussed the findings of the experiment.

It is almost obvious that the GNN architecture is best suited for the problem at hand from the apriori discussion on the advantages of GNN in section 3.2 as well as thee results provided in section 5. However, the GNN class of architectures houses a multitude of models functioning under varying principles. The Graph Attention Layer used in this study is one such promising architecture in the GNN class.

One notable subclass of the GNN class are the PointCloud models[12]. Point cloud models function by taking a large number of points as nodes and dynamically computing the edges of a graph using a k-nearest neighbour algorithm with each message passing iteration. Although popular for segmentation and node classification tasks. These models could be modified for node regression tasks as well.

Another problem of interest is the computation of spatial Cauchy stress tensor for a simulation which aggregates the local stresses inside divisions of the simulation box called bins and assigns one stress tensor for one bin. Since bins have a fixed shape and form as opposed to atoms and graphs, a larger pool of State of the art architectures (CNNs, Transformers etc.) can be experimented.

Overall, this study opens the possibility of using deep learning models for faster computation of local properties of a molecular system. It also encourages construction and research on new foundation models which are optimized for regression tasks in large molecular systems.

References

[1] Maginn, E. J., Messerly, R. A., Carlson, D. J., Roe, D. R., & Elliot, J. R. (2018). Best Practices for Computing Transport Properties 1. Self-Diffusivity and Viscosity from Equilibrium Molecular Dynamics [Article v1.0]. Living Journal of Computational Molecular Science, 1(1), 6324. https://doi.org/10.33011/livecoms.1.1.6324

[2] Šlepavičius, J., Patti, A., McDonagh, J. L., & Avendaño, C. (2023). Application of machine-learning algorithms to predict the transport properties of Mie fluids. Journal of Chemical Physics Online/the Journal of Chemical Physics/Journal of Chemical Physics, 159(2). https://doi.org/10.1063/5.0151123

[3] Fraces, C. G., Papaioannou, A., & Tchelepi, H. (2020, January 15). Physics Informed deep learning for transport in porous media. Buckley Leverett Problem. arXiv.org. https://arxiv.org/abs/2001.05172

[4] Hollingsworth, S. A., & Dror, R. O. (2018). Molecular dynamics simulation for all. Neuron, 99(6), 1129–1143. https://doi.org/10.1016/j.neuron.2018.08.011

[5] Liu, L., Wang, Y., & Chi, W. (2024). Image recognition technology based on machine learning. IEEE Access, 1. https://doi.org/10.1109/access.2020.3021590

[6] Khurana, D., Koli, A., Khatter, K., & Singh, S. (2022). Natural language processing: state of the art, current trends and challenges. Multimedia Tools and Applications, 82(3), 3713–3744. https://doi.org/10.1007/s11042-022-13428-4

[7] GNNBook@2023. (n.d.). https://graph-neural-networks.github.io/

[8] Freund, Y., Schapire, R.E. Large Margin Classification Using the Perceptron Algorithm. Machine Learning 37, 277–296 (1999). https://doi.org/10.1023/A:1007662407062

[9] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017b, June 12). Attention is all you need. arXiv.org. https://arxiv.org/abs/1706.03762

[10] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2017b, October 30). Graph attention networks. arXiv.org. https://arxiv.org/abs/1710.10903

[11] Clevert, D., Unterthiner, T., & Hochreiter, S. (2015, November 23). Fast and accurate deep network learning by exponential Linear units (ELUS). arXiv.org. https://arxiv.org/abs/1511.07289v5

[12] Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2016, December 2). PointNet: Deep learning on point sets for 3D classification and segmentation. arXiv.org. https://arxiv.org/abs/1612.00593