National College of Ireland

Emulating Simulation Models with Neural Networks

M.Sc. Research Project
Data Analytics

Aryan Singh
Student ID: 21205523

School of Computing
National College of Ireland

Supervisor: Prof. Dr. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aryan Singh
Student ID:	21205523
Programme:	Data Analytics
Year:	2023
Module:	M.Sc. Research Project
Supervisor:	Prof. Dr. Hicham Rifai
Submission Due Date:	14/08/2023
Project Title:	Emulating Simulation Models with Neural Networks
Word Count:	4995
Page Count:	16

Signature:	
Date:	18th September 2023

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Emulating Simulation Models with Neural Networks

Aryan Singh
21205523

Abstract

Deep Learning (DL) algorithms have innovated and repurposed a myriad of fields and even improved methodologies previously thought to be optimal. Modeling, especially Simulation Models (SM) is one such field. In this paper, a novel methodology is proposed to improve the performance and computation time of SMs through the application of DL models as surrogate models. An SM of traffic behavior in an inter-connected road network was created as the data source for this project. This model was run iteratively with varying parameters to generate data points. The I/O data of the SM was stored and curated to be used as the training data for the DL models. The relationships between the features in the data points generated by the SM were studied to model a Graph. Two NNs were trained on the raw simulation data and the transformed graph data, respectively. The NNs were evaluated on their accuracy and R-squared values against the output of the SM. The NNs provided high accuracy and were able to map the relationships between the features of the SM. The primary goal of this research was to identify and quantify the performance benefits of using an NN to emulate an SM. The processing speeds of both the models were compared and the Neural Networks proved to be exponentially faster than the Simulation model.

1 Introduction

1.1 Motivation and Background

Simulation Models (SMs) have been used by researchers and scientists for decades, with documented cases going back to World War 2 when researchers in the Manhattan Project used a Monte-Carlo algorithm to simulate a nuclear detonation(Gillier and Lenfle (2019)). These models allow researchers to imitate a system and its natural advancement through time much faster than real time. They allow researchers to peek into the behavior and different states of systems to predict possible outcomes and prepare accordingly. The complexity of an SM is directly related to the complexity of the system it is trying to simulate. Important systems with any degree of interest worthy of simulation are seldom simple. This results in the SM being rather complex and complicated, requiring large amounts of resources and human input to become reliable(Zhang et al. (2023)). This has led researchers to look for alternatives elsewhere.

The purpose of the research in this paper is not to speed up the simulation modeling process or even to look at prospective alternatives, it has already been done successfully in multiple studies(Thomas et al. (2017), Kwak et al. (2021)). The primary objective of this study is to investigate whether SMs that have already been created and operated could possibly be replaced by a DL model. The differentiating factor of this study will be that the DL model would not be trained on the data used for the Simulation Modeling, or any external data source for that matter. The study's primary focus will be training the DL model on the structure and output of the simulation model itself. This, in essence, will create a Deep Learning emulation of our SM. In simple words, the DL model would emulate the SM and act as a surrogate model. The motivation for this research is to find out whether this is possible and if it is, to evaluate the computational performance benefits of using the DL model. Since SMs transition through each state of the system they are trying to simulate and try to imitate the system as closely as possible, they require a lot of resources and time to provide results. The DL model on the other hand could learn the underlying relationships between the features and could derive the results directly without transitioning through the intermediate states. The research expects there to be a performance bump due to this reason.

1.2 Research Question

To what extent does the implementation of neural networks in emulating simulation models affect computational speed in comparison to conventional simulation techniques?

1.3 Research Objective

The objective of this research is to evaluate the computation time improvement of using a DL model instead of an SM. An SM of traffic behavior in an inter-connected road network is created and subsequently run through multiple iterations with varying parameters while logging its input and output parameters. These parameters are stored and help in the curation of a dataset. The dataset is then converted into a graph through vector embedding to better represent the relationships among the different features in the dataset. A feed-forward Neural Network(NN) is trained on the graph and then evaluated on its accuracy and R-squared value compared to the SM. The final objective of this paper is to establish an improvement of the computation time required by NN over SM to predict scenarios.

1.4 Document Structure

1. Introduction: The Introduction section eases into the research area and the topic, and provides the background and the motivation for the research topic while explaining the rationale behind it.

2. Related Work: The paper goes over the current state of the art in related technologies. It goes into the previous work that has been done on simulation models and deep learning and covers an in-depth literature review of the amalgamation of both of these technologies.

3. Methodology: The Methodology section covers the methodology used in the paper and an explanation of the different steps involved in the process.

4. Design Specification: This section explains the design choices and structure of the methodology used. It covers the frameworks used and the reasoning behind them. and the steps involved

5. Implementation: This section covers the implementation process and explanation of how the programming was carried out.

6. Evaluation: The evaluation section critically evaluates the performance of the methodology used in the paper, using relevant metrics and sound rationale to reach a verdict.

7. Conclusion and Future Work: The conclusion section discusses what was right or wrong in the application of the paper, and what things could have been different. It also discusses future prospects for this research.

2 Related Work

2.1 Current State-of-the-Art Simulation Modeling and DL Incorporation

Similar to a lot of fields, Simulation Modeling has also seen the adoption of ML and DL systems on a large scale(Tolk (2015), Yeo and Melnyk (2019), Hu et al. (2018)). Simulation of Urban Mobility (SUMO) or traffic simulation has been attempted and revised countless times. The authors in the study Bi et al. (2019) simulated intersectional traffic in tandem with a deep learning network, with the focus being on more of a physics-based modeling compared to a statistical-modeling. The authors employed a combination of a Convolutional Neural Network (CNN) and a Recurrent Neural Network(RNN) to learn the path trajectory of vehicles and set rules on which a visual-graphic simulation was run. The results critically compared the performance of this approach versus the traditional SUMO architecture. The primary evaluation metrics, ADE (average displacement error) and the Final Displacement Error (FDE) of the DL approach were significantly better than the SUMO approach, demonstrating a decrement of 87% and 89.3% in the errors respectively. Another study Hu et al. (2018), saw a similar bump in performance where the authors employed a Long Short-Term Memory (LSTM) network to predict the rainfall-runoff demonstrating that the LSTM network had a significantly higher accuracy and an r-squared value of 0.95, implying that it understood the underlying variance correctly with a degree of 95%.

2.2 Previous Work on Similar Surrogate Models

Although a lot of work has been done on incorporating DL networks with simulations but not much could be found on training a deep learning network on simulation data except in Wang et al. (2019). The authors of this study developed a one-step end-to-end NN trained on simulation data to reconstruct Computational Ghost Imaging (CGI) data. ‘Ghost imaging is often understood as imaging using light that has never physically interacted with the object to be imaged. Instead, one light field interacts with the object and a separate light field falls onto the imaging detector.’, Padgett and Boyd (2017). Data scarcity is one of the driving factors cited by the authors of Wang et al. (2019) to warrant the use of simulation data as opposed to a curated dataset. A visual comparison of the reconstructed image results shows the superiority of the Deep Learning Computational Ghost Imaging (DLCGI) model over the two simulation models i.e. CGI and Compressive Ghost Imaging(CSGI). The DLCGI is not all perfect and scores lesser in r-squared values compared to the simulation models. The authors of this paper and all other previously mentioned papers, however, failed to measure the differences in computational time between the SM and DL models. Another relevant paper Liang et al. (2022), however, did critically compare the computation time differences. Importantly, they did not use the data obtained from the SM as the input data for the DL network, which was externally sourced from NOAA-20. The methodology however did make use of the data generated by the Community Radiative Transfer Model (CRTM) simulation as labels for the input data. The authors employed a deep neural network algorithm to emulate the Community Cadiative Transfer Model (FCDN-CRTM), which simulates the brightness temperatures (BTs) of clear skies over ocean surfaces. The authors briefly noted that the computation time decreased from 900s for the simulation model to just 8s for the DL model, indicating a 112-fold speed increase.

2.3 Simple NNs and Graph-Vector Embedded Neural Networks on Tabular Data

While DL mode has shown exceptional premise on non-structured data such as images, videos, objects etc., there is strong and compelling work showing that they are still outperformed by simple tree-based structures such as Xg-Boost (XGB) and Random Forest (RF) (Shwartz-Ziv and Armon (2022), Grinsztajn et al. (2022), Fayaz et al. (2022)). This hasn't stopped developments in the application of DL frameworks on tabular data showing promising and exceptional results, arik2021tabnet, katzir2020net, joseph2021pytorch, popov2019neural, zhu2021converting).

The paper "Well-tuned Simple Nets Excel on Tabular Datasets" by Kadra et al. (2021) Using a combination of 13 regularization techniques, regularizing plain Multilayer Perceptron (MLP) networks. Results indicate that well-regularized MLPs outperform both state-of-the-art neural networks and traditional methods like XGB across 40 tabular datasets. In Villaizán-Vallelado et al. (2023), an Interaction Network (IN) is employed within the Graph Neural Network (GNN) framework to model interactions between tabular features. The model outperforms DL benchmarks from a recent survey and is competitive against boosted-tree solutions. T2G-Former (Yan et al. (2023)) introduces a Graph Estimator, estimating relationships between tabular features and organizing them into relation graphs. Using this, the Transformer network processes tabular data based on interaction patterns from relation graphs. Experiments reveal superior performance amongst DNNs, competitive with Gradient Boosted Decision Tree models.

Promising progress has been made in using data transformations on tabular data to extract deep relationships between features that could then be exploited by a DL framework, Liao and Li (2023), Rao et al. (2023)). TabGNN, a novel methodology introduced in the paper Guo et al. (2021) leverages a multiplex graph to model multifaceted sample relationships in tabular data prediction (TDP). By merging learned and original embeddings, it outperforms the standard tabular solution, AutoFE in 4Paradigm, across multiple TDP datasets. Another study Khan et al. (2023), transforms tabular patient data into a knowledge graph to improve Low Birth Weight (LBW) predictions. By extracting node-related features, including embeddings using the node2vec algorithm, the method achieves over a 6% performance boost in a real-world dataset from the UAE when compared to traditional risk factors. Li et al. (2023) introduces a systematic approach to applying GNNs to Tabular Data Learning (TDL). The emphasis is on creating graph structures from input tabular data and employing GNNs for enhanced performance. The guide encompasses a taxonomy of graph structure construction and GNN application in various TDL scenarios. Node2Vec and other node embedding methods have been successfully applied on graphs to extract relevant information and have provided good results compared to previous studies, Palumbo et al. (2018), Xu (2021). An analysis of popular and promising node embedding methods(n=7) is conducted by Panayotov et al. (2022), where the Node2Vec(both default and optimized) model outperforms all other node embedding methods significantly.

Deep Learning (DL) has been progressively integrated into Simulation Modeling with notable advancements using combined neural network architectures. While there's an acknowledgment of the computational advantages of DL models, a comparison of computational times across models remains an area of exploration. Even though deep learning has shown promise, it often competes with traditional tree-based methods in the context of tabular data. However, the application of Graph Neural Networks and data transformations for tabular data prediction has seen a significant trend with multiple methodologies emerging to harness deep relationships within such data structures.

3 Methodology

The methodology used in this paper could be broken into 4 broad sections. A brief outline of the methodology flow is shown in Figure 1

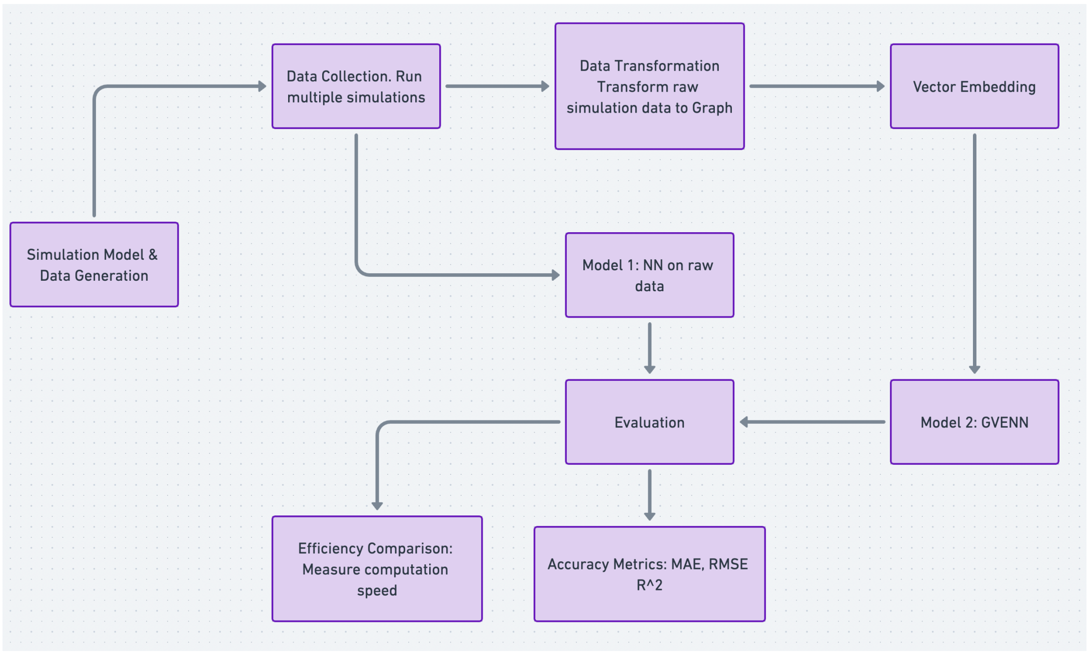
Simulation Model & Data Generation
Data Collection. Run multiple simulations
Data Transformation Transform raw simulation data to Graph
Vector Embedding
Model 1: NN on raw data
Model 2: GVENN
Evaluation
Efficiency Comparison: Measure computation speed
Accuracy Metrics: MAE, RMSE R^2

Figure 1: Methodology Used for Research.

3.1 Data Collection

Modeling the real-world behavior of traffic as closely as possible, the data for this project was generated in-house. A simulation model of traffic behavior in an interconnected Road network was designed. This SM would later give us the data needed for training our deep-learning models. The design of the simulation presented here is straightforward and rudimentary. While there are more intricate models available, the objective of this study is not to construct a complicated model. Instead, the aim of this research is to evaluate how a deep learning model could substitute a simulation model. The SM created here is a discrete-event-based simulation model that aims to simulate the behavior of vehicles transiting on a road network. The SM created here has a number of parameters on which it runs. The road network could be represented as a vector space where the nodes are traffic intersections and the edges are the roads. The SM also takes into account traffic signals and delays caused due to them. The method used to add a delay to account for accidents that happen on the road and can affect travel time and take into account. The wait time is then calculated by the simulation model by running through the different states of the system. The wait time is the culmination of different factors affecting the simulation models including a lot of parameters such as speed factor which represents different speeds of vehicles of different sizes. The simulation is run for a number of iterations with changing parameters. The simulation model is run for 1000<n>5000 iterations to curate a dataset of sufficient size. This process is repeated multiple times with varying values of n to understand the DL model's training of different kinds of data. The data collected at this step had 4 features:

- **Inter-Arrival time** is the mean arrival time between each generated vehicle. The generation of vehicles is exponentially distributed following a Poisson Process(PP).
- **Max Cars** is the number of cars that the road network can handle at a time.
- **Duration** is the length of the duration of the simulation in unit time.
- **Average Wait Time** Is the average wait time for a car in one iteration of a simulation.

3.2 Data Transformation

The data was obtained from the Multiple iterations of simulation with varying and unique parameters. The data is copied and one set of the data set is set aside to train our feed-forward Neural Network. The second data set is processed through network x and is converted into a graph. Each row of the data set is set as a note and the Euclidean distance between the values of these rows is set as the edge. A threshold of 5 units is implemented and the notes are only connected to each other if their distance is lower than this threshold. Node2Vec module is then used to perform vector embedding in this graph's structure.

3.3 Model Building

Two models are used in this project to predict the average wait times of a simulation given its parameters. in Model 1 a feed-forward neural network (FFNN) is trained upon the tabular data set obtained from the simulation runs. Model 2 is the Graph and Vector Embedded Neural Network (GVENN) model trained on the transformed data. While Model 1 aims to model the relationship between the features of each simulation run, Model 2 tries to grasp at the relationship between each pair of simulation runs and doesn't look inside the model but rather at the connection between runs.

3.4 Evaluation

The models are evaluated in two areas. How accurately are these models able to capture the relationship and predict the target variable? And how efficient are the models computationally and their running Times compared to the simulation model. For accuracy, we are using Root Mean Squared Error (RMSE), Mean Absolute Error, (MAE), and r-squared values. To calculate the difference in the processing speed, the simulation model and both the deep learning models are given previously and seen data on a single machine To calculate the difference in their running Times.

4 Design Specification

4.1 Simulation Model

The simulation modeled here is a discrete-event model of traffic behavior. The movement logistics of vehicles are simulated to gauge the average time spent by each vehicle(agent) in each state. The states, not necessarily in any order are: i) arrived ii) queued iii) transiting(on road) iv) finished(exited the road network). SimPy, a process-based discrete-event simulation framework, was used to create the simulation. Figure 2 shows the exponential distribution of the IAT times, which is a Poisson process. The Poisson process is a stochastic model used in queuing theory to model random events such as the arrival of a car on a road.

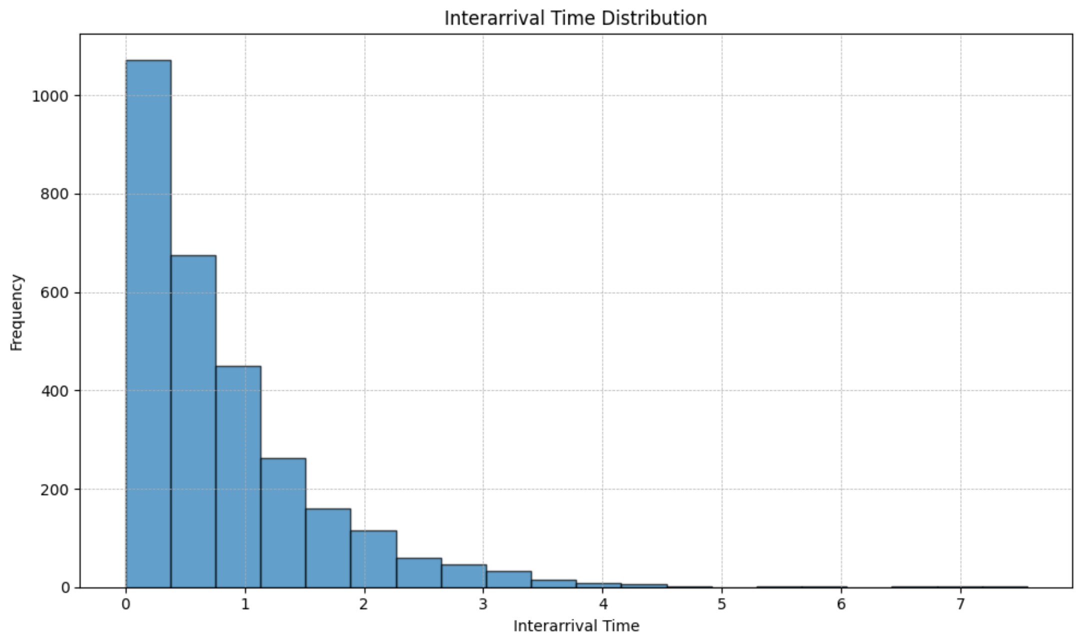

Figure 2: Inter-arrival Times of consecutive vehicles as a Poisson Process.

4.2 Graph Representation

The NetworkX framework is used to create a graph representation of the simulation data. NetworkX is a Python library used for creating, manipulating, and studying the structure of complex networks Hagberg and Conway (2020). It provides tools to work with both undirected and directed graphs.

The advantage of using graph representation here is that it looks at the relationship between each data point(row) of the simulation data rather than the relationships between the features of the dataset(column). Each node in this graph is a data point and the edges are the Euclidean distances between them. This graphical representation allows the NN model to extract and train on the relationships between different instances of the simulation in contrast to learning the relationships between features, which would be limited to just each particular simulation run.

4.3 Vector Embedding

Node2Vec is a technique to embed nodes from a graph into a low-dimensional space and operates by performing random walks on the graph to find optimal embeddings, Grover and Leskovec (2016). It produces continuous feature representations for nodes in networks which can capture complex patterns of node neighborhoods.

4.4 Deep Learning Models

Two Neural Network architectures are used here. Both the NNs used in this paper are simple Feed-Forward-Neural-Network (FFNN). Other flavors and more advanced mutations of these simple FFNNs such as CNN, LSTM, RNN were considered for the research but the literature review and further probing demonstrated that these simple NNs are the best application for structured tabular data. Both the NN models were implemented using the TensorFlow Keras framework which provides a straightforward way to implement these networks.

4.5 Machine Learning Models

It has been observed by many studies that although DL models have performed extremely well on unstructured data, they still lag behind simple ML models. Two ML models; i) XGB and ii) RF are used as control models to compare the results obtained from the DL models. Both these models were trained on the same raw simulation data and were optimized for performance.

4.6 Hyper-parameter Optimization and Evaluation

Hyper-parameter Optimization and Tuning (HOT) was carried out for all the models including Graph with Vector Embedded Neural Network(GVENN) model to ascertain a proper and just application of these models. The techniques used vary from model to model but it was an iterative process using established standard practices. For instance, Walk Sampling (WS) was used to determine the number of walks needed in vector embedding.

Evaluation metrics(RMSE, MAE, r-Squared) were documented and stored for each run of all the models used. Additional loss function graphs, residual plots, etc. were also plotted at each run for evaluation. The computing speed of each model's run was also stored for evaluation and comparison.

5 Implementation

5.1 Simulation Modeling

The Simulation Modeling process is described in the following sections. Figure 3 shows the wait time distribution and the relation between arrival vs. wait time in a stable run. The wait time too is exponentially distributed due to the exponential distribution of the IAT times.

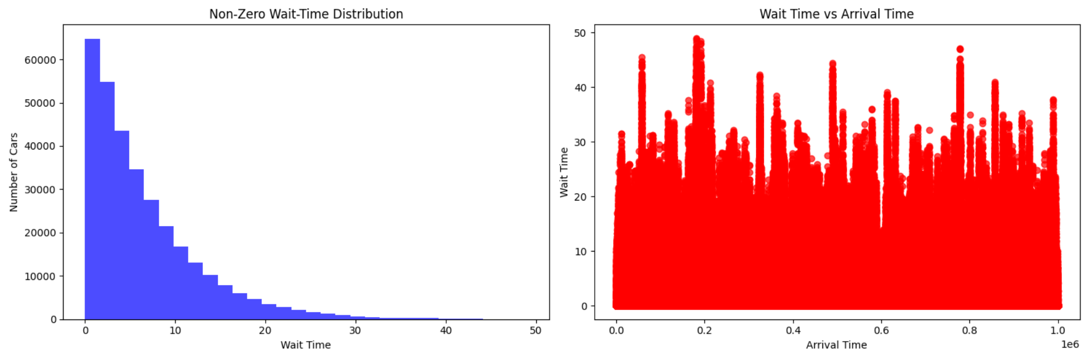

Figure 3: i) Wait time distribution and ii) Relation between Arrival and wait times.

Input Parameters of the SM:

- **mean_interarrival_time:** Average time between two consecutive vehicle arrivals.
- **max_cars:** Maximum number of vehicles that could occupy the road network at any moment.
- **until:** Duration of the simulation run in unit time.

Implementation Process:

- A Simpy Environment is instantiated with a resource object 'road' with the capacity set to max_cars.
- Cars are generated at intervals based on an exponential distribution with a mean value of mean_interarrival_time.
- Cars are categorized as either 'fast' or 'slow' which is assigned randomly. Fast cars have a base speed of 12 units and slow cars have a base speed of 6 units. Occasionally, a car's base speed is halved with a 10% probability.
- A feedback mechanism reduces car speeds by 20% when road occupancy is at or above 80%.
- As cars arrive, they are queued and wait for road space if the road is at full capacity.
- The time a car spends on the road is dependent on its speed and road capacity.
- Upon a car's exit, relevant data such as Car ID, speed, times of arrival, entry, and exit, as well as wait time, are recorded.

- The mean wait time is recorded for each simulation run. This is the target variable for the models used in this paper.

The mean_interarrival_time, max_cars, until, and the calculated average wait time for each simulation run are recorded and stored for training. The simulation is run for 5,000 iterations with different parameters to obtain substantial training data.

5.2 Graph representation and Vector Embedding

Data Loading: The simulation_data.csv file stores the recorded simulation data. Euclidean Distance Computation: Pairwise Euclidean distances between each data point(row) are evaluated based on the attributes: Mean_Interarrival_Time, Max_Cars, and Duration.

NetworkX Graph Implementation:

- Using the NetworkX framework, a graph G is initialized with each data point as a node.
- The Euclidean distance between each node pair is calculated. A threshold value = 2.5 is provided to the NetworkX framework at the time of Graph creation. This means that only distances in the **lowest 2.5 percentile** of all computed distances will be created as edges.
- Edges are systematically added to the graph if the distance between respective data points is less than this threshold. This ensures that edges in the graph are only created if both nodes are close and relevant to each other.

Node Embedding with Node2Vec:

5.2.1 Node2Vec initialization parameters:

- **dimensions=64:** The number of dimensions indicates the size of the vector representation for each node. Each node will be represented as a 64-dimensional vector.
- **walk_length=10:** Random walks are paths that start at a particular node and move to its neighbors (and neighbors' neighbors and so on) in a random manner. Every random walk spans 10 steps.
- **num_walks=100:** The number of walks dictates how many different random walks should be started from each node. Initiates 100 random walks per node.
- **workers=4:** Number of processor threads assigned. 4 threads are used here

5.2.2 Node2Vec model Parameters:

- **window=10:** This denotes the maximum span between the current and predicted node within a walk influencing the context size.
- **min_count=1:** Ensures all nodes find representation regardless of their frequency in walks.

5.3 Neural Networks

Two NNs are trained on the raw simulation data and the transformed data, respectively. Both datasets are split into train and test sets with 80/20 sizes. The datasets are scaled using the Standard Scaler which normalizes the data points from their distance to the mean and unit variance.

The NN trained on raw simulation data has:

- Three layers with one input layer = 64 neurons, one hidden layer = 32 neurons, and one output layer = 1 neuron.
- The input and the hidden layers use the Rectified Linear(ReLu) activation function while the output layer uses a Linear activation function since there is just one target variable.

The GVENN trained on transformed data has:

- Three layers with one input layer = 128 neurons, two hidden layers = 64, 32 neurons respectively, and one output layer = 1 neuron.
- The activation functions used here are identical to the previous model.

The loss function used for both models is the RMSE. It punishes deviations from the actual values harshly and is a good loss function to measure the models' performances. R-squared, Mean Square Error(MSE), and MAE errors are also recorded here for evaluation.

5.4 ML Models

Standard RF and XGB models were applied with default parameters.

6 Evaluation

The models were rigorously tested on the selected evaluation metrics. The experiment was run >30 times to understand the behavior and the output of the model. While the NN on raw simulation data and both ML models provided consistent results, the GVENN model trained on the transformed significant fluctuations in performance. Significant changes to the inter-relation of the different simulation runs elicited a new behavior in the GVENN model.

Any new significantly different data set required adjustments and changes in the parameters of the Graph construction and Node embedding process. Most significant changes in the GVENN model's performance were observed by changing the edge-cutoff threshold parameter. This parameter is provided to the NetworkX module at the time of graph creation. Number of walks and the walk length parameters also affected the GVENN's performance but to a much lesser degree.

Both the ML models performed well against each dataset, as expected. The NN model proved to be marginally better than the RF, XGB models in all the cases.

The metrics considered:

1. MAE: is the absolute difference between the actual and predicted values.

$$\sum_{i=1}^{D} |x_i - y_i|$$

2. RMSE: is the root of the square of the differences between the actual and predicted values.

$$\sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(\frac{d_i - f_i}{\sigma_i}\right)^2}$$

3. r-squared value: is the measure of how closely the predictions fit the actual regression line.

$$R^2 = 1 - \frac{sum\ of\ squared\ regression(SSR)}{total\ sum\ of\ squares(SST}$$

6.1 n = 5148. GVENN threshold = 1.9

Table 1 shows the performance of all the models with 5148 runs of the SM for training data. NN outperforms all the models. GVENN had to be tuned with the threshold = 1.9 to give a good result. RF and XGB models perform really well and fit the data correctly.

Table 1: Evaluation scores. n=5148

Metric	NN	GVENN	RF	XGB
RMSE	19.73	69.90	22.09	22.63
MAE	8.35	40.57	8.89	9.83
$R\hat{2}$	99.16%	89.52%	98.95%	98.90%

The Figure 4 shows the fit and residual distribution of the NN and GVENN models respectively. The NN does a good job of minimizing the residuals and performs well.

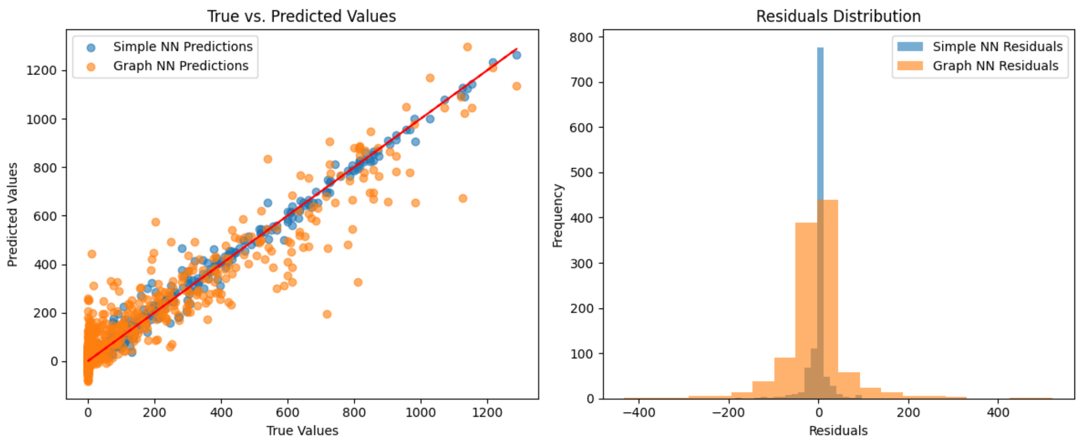

Figure 4: Fit and Residuals for the DL models. n = 5148

6.2 n = 3300, GVENN threshold = 2.5.

The Table 2 shows the performance of all the models with 3300 runs of the SM for training data. The threshold value of edge construction for the graph is 2.5.

6.3 n = 900. GVENN threshold = 5

The Table 3 shows performance with 900 iterations and a threshold value of 5.

Table 2: Evaluation scores. n=3300

Metric	NN	GVENN	RF	XGB
RMSE	37.18	49.01	41.04	42.32
MAE	29.49	37.69	31.89	32.42
R^2	98.71%	97.76%	98.43%	98.33%

Table 3: Evaluation scores. n=900

Metric	NN	GVENN	RF	XGB
RMSE	25.13	41.69	29.65	31.93
MAE	19.90	32.84	23.39	25.80
$R\hat{2}$	98.70%	96.42%	98.19%	97.90%

6.4 Comparing Required Computation Times

The Table 4 displays and critically contrasts the difference in the running speeds of Simulation vs all other models. All 4 models are exponentially faster than the SM. The SM as well as the 4 ML and DL models were run on the same machine for the comparison. The relationship between the increase in the computation time of the SM vs the other models doesn't seem linear.

Table 4: Running times in seconds.

Iterations	SM	NN	GVENN	RF	XGB
2100	190	0.18	0.17	0.029	0.0056
3300	230	0.23	0.33	0.042	0.0080
5481	130	0.37	0.40	0.058	0.015
10962	340	0.61	1.4	0.10	0.022

6.5 Discussion

The results clearly demonstrate the superiority of the simple NN model for this task. The GVENN model fails to reach the standards of either the NN or both the ML models but nonetheless, it scores well against the simulation data and has a high accuracy. The GVENN model does require extra attention as the data transformation step heavily depends on the simulation data and any change to the structure of the raw simulation data should be accommodated in the Graph construction and Vector embedding steps first. The GVENN architecture follows a totally different approach from the other 3 models and could actually outperform the other models given the right set of data. It doesn't take into account the relation between the features of the simulation but the connection and the distance between each simulation run.

The models are much faster than the SM. The calculation of the n=2100 iteration between the GVENN and SM shows that the GVENN model is $\tilde{1}000$ faster than the SM model.

7 Conclusion and Future Work

The research conducted in this paper focuses on the adoption of DL methodologies as an alternative to SMs. DL models have already started replacing SM given there is ample external data. The key findings in this research revealed that Deep Learning models, particularly the simple Neural Network can significantly outpace traditional Simulation Models in computational speed. The GVENN model also shows promise but with some sensitivity to data structures. These results carry implications suggesting a paradigm shift in how simulations might be approached in the future using the efficiency of Deep Learning for faster more agile decision-making processes. The efficacy of the research is evident in the clear computational advantages demonstrated by the DL models over traditional SMs. However, it's essential to acknowledge the study's limitations. Particularly the GVENN model's dependency on data structure and re-training with different parameters with changes in the dataset. A more detailed study could not be conducted due to the limited computational resources and the high resource demand of Graph construction and Node embedding process.

Building on the findings of this research, there are several options for future exploration. One promising direction is to delve deeper into the GVENN model's sensitivity to data structure changes. Understanding the nuances of this model's behavior across various data structures could lead to more robust and adaptable DL models. The application of more advanced neural network architectures might offer further performance improvements over traditional SMs. This DL emulation approach could be extended to other domains where SMs are prevalent, such as meteorological forecasting or financial modeling where time is of utmost importance and the speeds of SMs is proving to be a bottleneck.

8 Acknowledgements

This research wouldn't have been possible without the continuous support of my research supervisor, Prof. Dr. Hicham Rifai. His guidance and help proved to be really helpful for the implementation of the project.

The traffic SM used in this research paper was modeled after the teachings and course content provided by Prof. Dr. Christian Horn without whom this research might not have been possible. I express my deep gratitude towards Prof. Dr. Christian Horn for providing guidance and course content for my research.

References

Bi, H., Mao, T., Wang, Z. and Deng, Z. (2019). A deep learning-based framework for intersectional traffic simulation and editing, *IEEE Transactions on Visualization and Computer Graphics* **26**(7): 2335–2348.

Fayaz, S. A., Zaman, M., Kaul, S. and Butt, M. A. (2022). Is deep learning on tabular data enough? an assessment, *International Journal of Advanced Computer Science and Applications* **13**(4): 466–473.

Gillier, T. and Lenfle, S. (2019). Experimenting in the unknown: lessons from the manhattan project, *European Management Review* **16**(2): 449–469.

Grinsztajn, L., Oyallon, E. and Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data?, *Advances in Neural Information Processing Systems* **35**: 507–520.

Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks, *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864.

Guo, X., Quan, Y., Zhao, H., Yao, Q., Li, Y. and Tu, W. (2021). Tabgnn: Multiplex graph neural network for tabular data prediction, *arXiv preprint arXiv:2108.09127* .

Hagberg, A. and Conway, D. (2020). Networkx: Network analysis with python, *URL: https://networkx. github. io* .

Hu, C., Wu, Q., Li, H., Jian, S., Li, N. and Lou, Z. (2018). Deep learning with a long short-term memory networks approach for rainfall-runoff simulation, *Water* **10**(11): 1543.

Kadra, A., Lindauer, M., Hutter, F. and Grabocka, J. (2021). Well-tuned simple nets excel on tabular datasets, *Advances in neural information processing systems* **34**: 23928–23941.

Khan, W., Zaki, N., Ahmad, A., Bian, J., Ali, L., Mehedy Masud, M., Ghenimi, N. and Ahmed, L. A. (2023). Infant low birth weight prediction using graph embedding features, *International Journal of Environmental Research and Public Health* **20**(2): 1317.

Kwak, J., Han, H., Kim, S. and Kim, H. S. (2021). Is the deep-learning technique a completely alternative for the hydrological model?: A case study on hyeongsan river basin, korea, *Stochastic Environmental Research and Risk Assessment* pp. 1–15.

Li, C.-T., Tsai, Y.-C. and Liao, J. C. (2023). Graph neural networks for tabular data learning, *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, IEEE, pp. 3589–3592.

Liang, X., Garrett, K., Liu, Q., Maddy, E. S., Ide, K. and Boukabara, S. (2022). A deep-learning-based microwave radiative transfer emulator for data assimilation and remote sensing, *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **15**: 8819–8833.

Liao, J. C. and Li, C.-T. (2023). Tabgsl: Graph structure learning for tabular data prediction, *arXiv preprint arXiv:2305.15843* .

Padgett, M. J. and Boyd, R. W. (2017). An introduction to ghost imaging: quantum and classical, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **375**(2099): 20160233.

Palumbo, E., Rizzo, G., Troncy, R., Baralis, E., Osella, M. and Ferro, E. (2018). Knowledge graph embeddings with node2vec for item recommendation, *The Semantic Web: ESWC 2018 Satellite Events: ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers 15*, Springer, pp. 117–120.

Panayotov, P., Shukla, U., Sencar, H. T., Nabeel, M. and Nakov, P. (2022). Greener: Graph neural networks for news media profiling, *arXiv preprint arXiv:2211.05533* .

Rao, P. K., Chatterjee, S., Nagaraju, K., Khan, S. B., Almusharraf, A. and Alharbi, A. I. (2023). Fusion of graph and tabular deep learning models for predicting chronic kidney disease, *Diagnostics* **13**(12): 1981.

Shwartz-Ziv, R. and Armon, A. (2022). Tabular data: Deep learning is not all you need, *Information Fusion* **81**: 84–90.

Thomas, J., Thomas, S. and Sael, L. (2017). Feature versus raw sequence: Deep learning comparative study on predicting pre-mirna, *arXiv preprint arXiv:1710.06798* .

Tolk, A. (2015). The next generation of modeling & simulation: integrating big data and deep learning, *Proceedings of the conference on summer computer simulation*, pp. 1–8.

Villaizán-Vallelado, M., Salvatori, M., Martinez, B. C. and Esguevillas, A. J. S. (2023). Graph neural network contextual embedding for deep learning on tabular data, *arXiv preprint arXiv:2303.06455* .

Wang, F., Wang, H., Wang, H., Li, G. and Situ, G. (2019). Learning from simulation: An end-to-end deep-learning approach for computational ghost imaging, *Optics express* **27**(18): 25560–25572.

Xu, M. (2021). Understanding graph embedding methods and their applications, *SIAM Review* **63**(4): 825–853.

Yan, J., Chen, J., Wu, Y., Chen, D. Z. and Wu, J. (2023). T2g-former: Organizing tabular features into relation graphs promotes heterogeneous feature interaction, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37, pp. 10720–10728.

Yeo, K. and Melnyk, I. (2019). Deep learning algorithm for data-driven simulation of noisy dynamical system, *Journal of Computational Physics* **376**: 1212–1231.

Zhang, Y., Schlueter, A. and Waibel, C. (2023). Solargan: Synthetic annual solar irradiance time series on urban building facades via deep generative networks, *Energy and AI* **12**: 100223.