Dynamic Path Planning for Autonomous Vehicles: A Neuro-Symbolic Approach

Omar Elrasas[1], Nourhan Ehab[1], Yasmin Mansy[1] and Amr El Mougy[2]
[1]*Department of Computer Science and Engineering, German University in Cairo, Cairo, Egypt*
[2]*Department of Computer Science and Engineering, American University in Cairo, Cairo, Egypt*

Keywords: Neuro-Symbolic AI, Dynamic Path Planning, Autonomous Vehicles.

Abstract: The rise of autonomous vehicles has transformed transportation, promising safer and more efficient mobility. Dynamic path planning is crucial in autonomous driving, requiring real-time decisions for navigating complex environments. Traditional approaches, like rule-based methods or pure machine learning, have limitations in addressing these challenges. This paper explores integrating Neuro-Symbolic Artificial Intelligence (AI) for dynamic path planning in self-driving cars, creating two regression models with the Logic Tensor Networks (LTN) Neuro-Symbolic framework. Tested on the CARLA simulator, the project effectively followed road lanes, avoided obstacles, and adhered to speed limits. Root mean square deviation (RMSE) gauged the LTN models' performance, revealing significant improvement, particularly with small datasets, showcasing Neuro-Symbolic AI's data efficiency. However, LTN models had longer training times compared to linear and XGBoost regression models.

1 INTRODUCTION

Self-driving vehicles are a transformative technology with the potential to revolutionize global transportation systems. The ability of autonomous cars to sense, decide, and navigate dynamic environments relies on sophisticated computational models and complex algorithms. Among the challenges faced, dynamic path planning is crucial for real-time decision-making and adaptability to changing road conditions.

Conventional approaches to path planning in self-driving cars include rule-based methodologies and pure machine learning techniques (González et al., 2015). However, rule-based methods struggle with adaptability, while machine learning methods lack transparency. Addressing these challenges requires an approach that combines clear rule-based thinking with machine learning capabilities. Neuro-Symbolic approaches aim to achieve this integration by combining symbolic reasoning with deep neural networks (Sarker et al., 2021).

Our objective is to explore Neuro-Symbolic Computing for dynamic path planning in autonomous vehicles. The aim is to develop a path planning system that can swiftly adapt to changing road conditions and potential risks. The paper investigates the integration of symbolic reasoning techniques with deep learning models using the Logic Tensor Network framework (Badreddine et al., 2022), resulting in a data-efficient Neuro-Symbolic regression model.

The rest of the paper is organized as follows. Section 2 provides background information, Section 3 discusses related research, and Section 4 presents the proposed neuro-symbolic path planning system. Section 5 explains model evaluation and presents results, including comparisons with other machine learning methods. Section 6 concludes the paper by summarizing the work and suggesting future research directions.

2 BACKGROUND

In this section, we delve deeper into the background of our research landscape, shedding light on crucial aspects that underpin our exploration of Logic Tensor Networks (LTN), path planning for autonomous vehicles, and the CARLA simulator.

2.1 Logic Tensor Networks

Logic Tensor Networks (LTN) is a novel Neuro-Symbolic Python framework that seamlessly com-

Elrasas, O., Ehab, N., Mansy, Y. and El Mougy, A.
Dynamic Path Planning for Autonomous Vehicles: A Neuro-Symbolic Approach.
DOI: 10.5220/0012374700003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 584-591
ISBN: 978-989-758-680-4; ISSN: 2184-433X

bines key properties of both neural networks (for training on data) and symbolic logic (for logical reasoning). Two versions of LTN have been published—one using TensorFlow and the other using PyTorch, the widely-used Python machine learning frameworks. In Logic Tensor Network (LTN), the framework utilizes a first-order logic knowledge base with axioms, functions, predicates, or logical constants. LTN employs logical axioms as a loss function, seeking solutions that maximally satisfy all the knowledge base axioms (Badreddine et al., 2022).

Real Logic, a fully differentiable logical language introduced by LTN, facilitates learning by grounding elements of a first-order logic signature onto data through neural computational graphs and first-order fuzzy logic semantics. Grounding, defined by Real Logic, involves mapping logical domain elements (constants, variables, and logical symbols) to tensors, enabling the incorporation of data and logic. LTN converts Real Logic formulae into PyTorch or TensorFlow computational graphs, depending on the version used (Figure 1) (Badreddine et al., 2022).

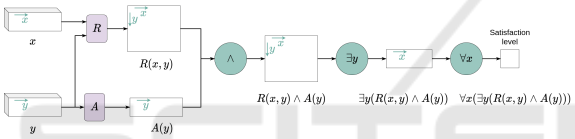

Figure 1: Real Logic Computational Graphs (Badreddine et al., 2022).

2.2 Path Planning for Autonomous Vehicles

Path planning is vital for autonomous vehicles to navigate safely and efficiently in complex environments, prioritizing passenger comfort. It encompasses two main types: local and global path planning. Local path planning focuses on generating short-term trajectories to avoid immediate obstacles, while global path planning finds a path from the starting point to the destination, considering long-term constraints (González et al., 2015).

A popular local path planning approach is the Artificial Potential Field (APF) method, creating a virtual potential field to guide the vehicle and avoid obstacles. This method, widely used in robotics, is also applied in autonomous vehicle path planning (González et al., 2015). For global path planning, the graph-based method represents the environment as a graph and uses algorithms like Dijkstra's or A* to find the shortest path. Another method is the Rapidly-exploring Random Tree (RRT) algorithm, which generates a tree-based search space for optimal path finding (González et al., 2015).

Machine learning techniques like deep learning and reinforcement learning have recently enhanced both local and global path planning for autonomous cars. These approaches, such as a supervised reinforcement learning-based method proposed by Hebaish et al. (Hebaish et al., 2022), demonstrate promising outcomes in navigating complex situations.

2.3 CARLA Simulator

The CARLA simulator, standing for Car Learning to Act, is an open-source platform designed for testing and validating autonomous driving systems in research and development. Built on the Unreal Engine, CARLA provides a realistic environment for algorithm and model testing across various driving scenarios, featuring customizable elements such as obstacles, autonomous traffic, and weather conditions in different towns with complex layouts (Dosovitskiy et al., 2017).

CARLA supports various sensors, including LiDAR, radar, collision detection, and RGBA cameras, enabling simulation of diverse perception scenarios and testing perception algorithms. Researchers, such as Dworak et al. (Dworak et al., 2019), have utilized CARLA to evaluate LiDAR object detection deep learning architectures based on artificially generated point cloud data from the simulator.

In addition to sensor support, CARLA allows the integration of various control algorithms for autonomous vehicles. For instance, Vlachos et al. (Vlachos and Lalos, 2022) proposed a cooperative control algorithm for platooning using CARLA, demonstrating its effectiveness in different scenarios to improve stability and fuel efficiency. CARLA has also been extensively used in various research tasks, including path planning, motion control, and behavior modeling. Hebaish et al. (Hebaish et al., 2022) tested their supervised reinforcement learning-based path planning algorithm for autonomous driving in CARLA, demonstrating its ability to generate safe and efficient driving trajectories in complex scenarios.

3 RELATED WORK

In examining related work, it is observed that many research papers proposing machine learning approaches for self-driving vehicles predominantly employ reinforcement learning. Sallab et al. (Sallab et al., 2017) introduced a deep reinforcement learning framework that handles partially observable scenarios through recurrent neural networks. This framework

integrates attention models, focusing on relevant information to reduce computational complexity on embedded hardware. Effective in complex road curvatures and vehicle interactions, the framework demonstrates maneuvering capabilities. Another example by Zong et al. (Zong et al., 2017) utilizes reinforcement learning for obstacle avoidance, incorporating vehicle constraints and sensor information with a deep deterministic policy gradients algorithm. Both techniques, however, face challenges such as long training times for achieving acceptable reward scores and a lack of crucial context in potentially life-threatening situations. The features of Neuro-Symbolic Artificial Intelligence (AI), such as explainability and data-efficiency, offer potential solutions to these drawbacks.

Given that Neuro-Symbolic AI is a relatively new research field, practical examples of its full potential are limited. An application in healthcare, proposed by Lavin et al. (Lavin, 2022), involves a probabilistic programmed deep kernel learning tool predicting cognitive decline in Alzheimer's disease. Effective in neurodegenerative disease diagnosis, the tool is speculated to perform well in other disease areas. This work will utilize Neuro-Symbolic AI for path planning in self-driving vehicles to explore its potential in this field, leveraging its advantages over other machine learning techniques.

4 A NEURO-SYMBOLIC PATH PLANNER

This section delves into the architecture, implementation, and dataset generation for Neuro-Symbolic models with the main aim of facilitating dynamic navigation for autonomous vehicles in complex scenarios.

4.1 System Design

In the CARLA simulator, vehicle control involves three main parameters: throttle (ranging from 0 to 1), brake (ranging from 0 to 1), and steer (ranging from -1 to 1). The objective of the LTN models is to predict these parameters continuously based on the current situation and environment. To implement these models in CARLA and control the vehicle, a main script is required to connect to the CARLA server, load the map, and spawn the vehicle.

The project's architecture, depicted in Figure 2, comprises three main components: CARLA, the LTN models, and the main script linking them together.

1. **Route Planning:** User-defined inputs, such as the vehicle's starting location and destination, initiate CARLA's global route planner. This planner calculates an optimal route with a list of waypoints. This stage occurs once at the script's beginning.
2. **Data Collection:** Real-time information about the vehicle and its environment is gathered in the second stage. CARLA's obstacle detection sensor records distances to potential obstacles, while the Python API retrieves details like speed, location, orientation, and the next waypoint's information continuously.
3. **Control and Prediction:** The third stage involves feeding the data collected in the second stage into the Neuro-Symbolic LTN models. These models predict throttle, brake, and steering commands based on the current scenario using Neuro-Symbolic techniques. The predictions control the ego vehicle in CARLA continuously, ensuring adaptability to unpredictable scenarios. The models are trained on datasets captured within CARLA, minimizing preprocessing and enhancing performance.

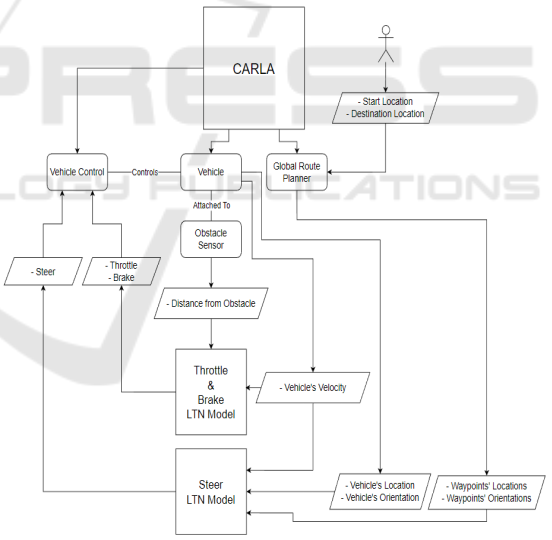

Figure 2: Detailed Architecture.

4.2 LTN Models Dataset Generation

Training data for both models was collected using CARLA's autopilot, and pandas was employed to generate Comma-separated Values (CSV) files containing the data. Pandas, a Python library designed for data analysis, facilitates the conversion of regular Python arrays into pandas dataframes—a two-dimensional data structure. These dataframes can then be transformed into CSV files using pandas' li-

brary functions.

4.2.1 Throttle and Brake Model

The Python script for the throttle and brake model in CARLA begins by loading town 10, a medium-sized urban city map, chosen for its diverse traffic scenarios. It spawns 100 Non-player character (NPC) vehicles randomly across the map, with one designated as the ego vehicle for data collection. Utilizing CARLA's autopilot function, the script configures the ego vehicle to ignore traffic lights and abstain from lane switching, ensuring dataset consistency. An obstacle detection sensor, configured for a 30-meter range and a hit radius matching the vehicle's width, captures dynamic obstacle information. The script runs for 15 minutes, collecting data on obstacle distances, vehicle speed, and autopilot-controlled throttle and brake values. The gathered data is stored in an array, converted to a CSV file using pandas (see section 4.2), and saved on disk.

4.2.2 Steer Model

In the steer model script, town 4, a large map with a small town and a substantial highway, is selected to emphasize extended lane following without distractions. Unlike the throttle and brake model, only the ego vehicle is deployed, and CARLA's world debug function visualizes spawn point locations for suitable routes.

No NPC vehicles are spawned to maintain dataset clarity. CARLA's autopilot trajectory being random requires the use of CARLA's vehicle Proportional-Integral-Derivative (PID) controller class for trajectory control. This class is instantiated with parameters suitable for the chosen Tesla Model 3, allowing precise route following.

The script involves traversing two lengthy routes around the highway, collecting vehicle and waypoint coordinates, yaw angles, and steering data. Dataset precision is maintained by adjusting waypoint yaw and approximating small steer values. The collected data is stored in an array, converted to a CSV file using pandas, and saved on disk, following the approach used for the throttle and brake model.

4.3 LTN Models Architecture and Training

Both models, based on the PyTorch version of LTN (LTN) known as LTNtorch, are Neuro-Symbolic regression models utilizing an equality predicate for training. This predicate measures similarity between

Figure 3: Spawn Points.

input tensors using the Euclidean similarity formula.

$$Eq(x,y) = \frac{1}{1+0.5*\sqrt{\sum_i (x_i - y_i)^2}} \quad (1)$$

The loss calculation involves a built-in aggregation operator, subtracted from 1, utilizing the First-order logic (FOL) ForAll quantifier for universal quantification. Bound variables from the diagonal quantifier express statements about specific pairs of values, ensuring accurate prediction of relationships between corresponding inputs and outputs in the CSV file. The loss equation is described as follows.

$$Loss = 1.0 - SatAgg(Forall(diag(x,y), Eq(f(x),y))) \quad (2)$$

Both models use pandas for loading and shuffling CSV file data, a dataloader for batching, and the Adam optimizer as the chosen optimizer.

4.3.1 Throttle and Brake Model

For the throttle and brake model (Figure 4), the neural network comprises 2 input layers, 3 hidden layers, and 2 output layers. The 2 input layers correspond to distance and speed, while the 2 output layers correspond to throttle and brake. The dataset, totaling approximately 30,000 rows, is batched into 256-row segments, with 85% allocated for training and 15% for testing. The learning rate is set to 0.001, and training spans 100 epochs.

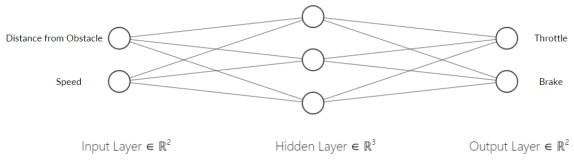

Figure 4: Throttle and Brake Model Neural Network.

4.3.2 Steer Model

For the steer model (Figure 5), the neural network comprises 3 input layers, 4 hidden layers, and 1 output layer. The first two input layers capture the difference in x and y coordinates between the vehicle and the waypoint, while the third input layer represents the difference in yaw angle between the vehicle and the waypoint. The single output layer corresponds to steer. The dataset, totaling approximately 500,000 rows, is batched into segments of 19,000 rows, with 90% allocated for training and 10% for testing. The learning rate is set to 0.001, and training spans 20 epochs.

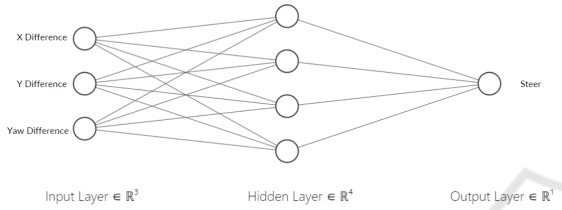

Figure 5: Steer Model Neural Network.

Both LTN model scripts feature a function designed for ease of use. This function takes necessary inputs, processes them, and outputs the model's predictions. The scripts commence by loading the CSV files for the dataset, instantiating the predicates and the neural network according to the specified architecture. Subsequently, the training process begins. Once trained, the models are ready for import and utilization in the main script, as detailed in the next section (4.4).

4.4 Main Script

CARLA operates on a client-server architecture, requiring the creation of a Python script to connect as a client to the running CARLA simulator server. For this project, a local host connection was established. Subsequently, both LTN models are imported and instantiated. Town 10, chosen for its diversity and medium size, is loaded for testing both models.

After loading the map, the user-specified spawn location is used to spawn the ego vehicle, along with the obstacle detection sensor. The spectator camera is instantiated and transformed to hover over the vehicle, providing the default camera view out of three implemented views. The second camera view is a hood view, and the third is a top view. As an example, the top view can be seen in Figure 6.

Once the spectator camera is configured, 50 additional NPC vehicles are spawned to create moderate traffic in the map. Following the overview provided in section (4.1), the ego vehicle's spawn point and destination are determined, supplying the global route planner with this information to generate the list of waypoints. CARLA's world debug function is employed to draw these waypoints, ensuring accurate tracking by the ego vehicle (Figure 3). Subsequently, the ego vehicle begins moving, continuously invoking the functions in the LTN models to predict throttle, brake, and steer using the necessary inputs. The predicted outputs of the LTN models are then utilized to control the ego vehicle through CARLA's vehicle control function, as explained in section (4.1). This process continues until the ego vehicle reaches the specified destination. Upon reaching the destination, CARLA's destroy actor command removes all spawned vehicles, concluding the simulation.

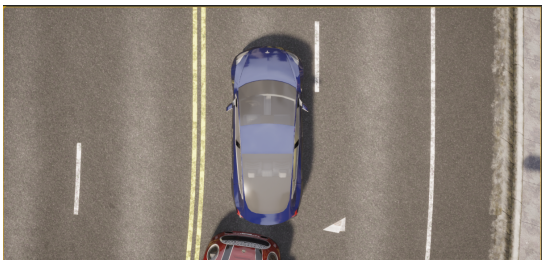

Figure 6: Top Camera View.

5 RESULTS AND DISCUSSION

This section outlines the evaluation metrics employed to assess the effectiveness and efficiency of the Neuro-Symbolic regression models. Additionally, a comparative analysis of the results with other regression models is presented.

5.1 Evaluation Metrics

To highlight the advantages of employing our proposed neuro-symbolic approach, a statistical linear regression model is established as a benchmark, alongside another regression model using XGBoost (Chen and Guestrin, 2016), which is an efficient implementation of gradient boosting suitable for regression predictive modeling. Both models are created to facilitate a performance comparison with the Neuro-Symbolic LTN models. To ensure a fair comparison, the linear regression model utilizes the same machine learning framework (PyTorch), neural network architecture, optimizer, etc., as the Neuro-Symbolic LTN models. Additionally, both the linear and XGBoost models share the same learning rate, number of epochs, etc., as the Neuro-Symbolic LTN models. The only distinction lies in the loss function, where

the linear and XGBoost models rely on the mean squared error criterion, a common choice for regression models, as opposed to the Neuro-Symbolic LTN models based on LTN predicates. Evaluation metrics such as Root Mean Square Error (RMSE), alongside training and prediction times, are utilized to assess the models' performance.

5.1.1 Root Mean Square Error

Also known as the standard error of regression, the RMSE stands out as one of the most widely utilized metrics for assessing the performance of machine learning regression models. Recognized as a proper scoring rule by Gneiting et al. (Gneiting and Raftery, 2007), it provides an intuitive measure of how far predictions deviate from actual measured values, with lower RMSE values indicating superior performance. The calculation of RMSE is expressed by the following equation, where n represents the number of data rows, y(i) is the i-th output, and $\hat{y}(i)$ is its corresponding prediction.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} ||y(i) - \hat{y}(i)||^2}{n}} \quad (3)$$

5.1.2 Training Time and Prediction Time

Training time refers to the duration required for the model to complete all epochs and become prepared for predictions, excluding the initial instantiation of predicates and loading the CSV file. On the other hand, prediction time is the duration taken by the model to process an input and produce its prediction.

The RMSE and training time metrics are computed for both Neuro-Symbolic LTN models and compared to benchmark models across varying data row sizes from the dataset. This analysis aims to assess the models' accuracy with different data volumes, emphasizing one of Neuro-Symbolic AI's key advantages—data efficiency. Prediction time is computed once at the end, with further details discussed in the following section.

5.2 Results

5.2.1 Throttle and Brake Model RMSE

The RMSE of the throttle and brake LTN model and both benchmark models was calculated while using 500, 1000, 2000, 5000, and 20000 data rows to train the models, the following results in Figure 7, and 8 were observed.

Data Rows	LTN Model RMSE	Linear Model RMSE	XGBoost Model RMSE
500	0.449	1.013	0.511
1000	0.416	0.447	0.441
2000	0.303	0.369	0.415
5000	0.289	0.321	0.377
20000	0.265	0.268	0.343

Figure 7: Throttle and Brake RMSE Table Comparison.

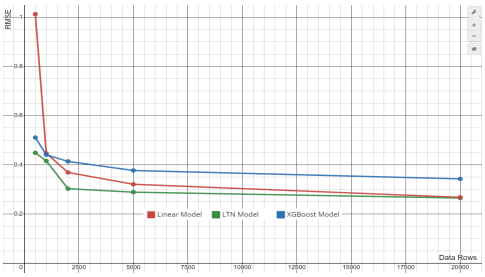

Figure 8: Throttle and Brake RMSE Graph Comparison.

5.2.2 Steer Model RMSE

The RMSE of the steer LTN model and both benchmark models was calculated while using 50000, 125000, 250000, and 500000 data rows to train the models, the following results in Figure 9, and 10 were observed.

Data Rows	LTN Model RMSE	Linear Model RMSE	XGBoost Model RMSE
50000	0.105	0.302	0.504
125000	0.034	0.067	0.504
250000	0.015	0.047	0.465
500000	0.005	0.016	0.465

Figure 9: Steer RMSE Table Comparison.

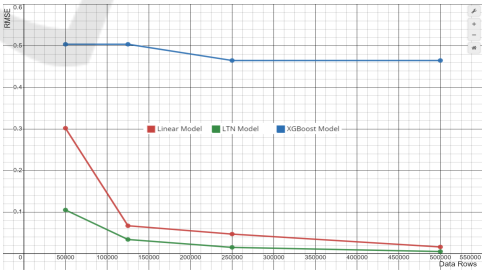

Figure 10: Steer RMSE Graph Comparison.

5.2.3 Throttle and Brake Model Training Time

The training time of the throttle and brake LTN model and both benchmark models was calculated while using 500, 1000, 2000, 5000, and 20000 data rows to train the models, the following results in Figure 11, and 12 were observed.

Data Rows	LTN Model Training Time	Linear Model Training Time	XGBoost Model Training Time
500	0.496	0.307	0.170
1000	0.954	0.570	0.197
2000	1.750	0.985	0.264
5000	4.045	2.331	0.486
20000	15.334	8.454	1.565

Figure 11: Throttle and Brake Training Time Table Comparison.

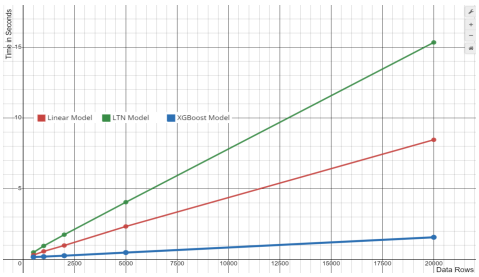

Figure 12: Throttle and Brake Training Time Graph Comparison.

5.2.4 Steer Model Training Time

The training time of the steer LTN model and benchmark models was calculated while using 50000, 125000, 250000, and 500000 data rows to train the models, the results in Figure 13, and 14 were observed.

Data Rows	LTN Model Training Time	Linear Model Training Time	XGBoost Model Training Time
50000	0.870	0.697	0.115
125000	1.887	1.605	0.233
250000	3.530	2.845	1.990
500000	7.240	5.540	5.200

Figure 13: Steer Training Time Table Comparison.

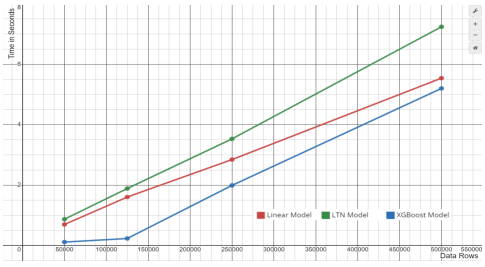

Figure 14: Steer Training Time Graph Comparison.

5.3 Discussion

The outcomes presented in the preceding section substantiate one of the primary advantages of Neuro-Symbolic AI—data efficiency. In contrast to the two benchmark models, the Neuro-Symbolic LTN models exhibit notably lower RMSE values, particularly when employing a smaller dataset. However, this efficiency is accompanied by a trade-off: longer training times, as elucidated in the ensuing discussion.

5.3.1 Throttle and Brake Model RMSE

As anticipated, Figures 7 and 8 illustrate that the throttle & brake LTN model exhibits a 55.6% lower RMSE than the linear model and a 12.1% lower RMSE than the XGBoost model with a dataset size of 500 rows. Nevertheless, as the dataset size increases, the disparity in RMSE between the throttle & brake LTN model and the two benchmark models diminishes. For instance, with 1000 data rows, the LTN model demonstrates a 6.9% reduction in RMSE compared to the linear model and a 5.6% reduction compared to the XGBoost model. While this reduction may seem modest, it can hold significance in many practical scenarios. The trend of diminishing differences is not consistent, as seen with a dataset size of 2000 rows, where the RMSE difference expands to 17.8% less than the linear model and 26.9% less than the XGBoost model. However, as the dataset size continues to grow from 2000 to 20000 rows, the RMSE difference gradually decreases, becoming negligible between the LTN model and the linear model, and reducing from 26.9% to 22.7% between the LTN model and the XGBoost model.

5.3.2 Steer Model RMSE

In line with the results observed in the throttle & brake model, Figures 9 and 10 underscore a significant divergence in RMSE between the LTN model and the two benchmark models. Commencing with the 50,000 data rows dataset, the steer LTN model exhibits a 65.2% reduction in RMSE compared to the linear model and a substantial 79.1% reduction compared to the XGBoost model. Similar to the pattern seen with the linear model in the throttle & brake model results, the gap gradually narrows when using 125,000 data rows. The LTN model displays a 49.2% reduction in RMSE compared to the linear model, but this difference increases again to 68% with 250,000 data rows. Subsequently, the difference gradually decreases until it becomes negligible for the expected output of this model.

Unlike the throttle & brake model, the XGBoost model performs poorly in the steer model compared to the other two models, exhibiting a substantial difference in RMSE. This discrepancy suggests that the XGBoost regression model's performance is suboptimal when the number of epochs is relatively low, as in the steer model, where it only undergoes training

for 20 epochs to expedite the process due to the larger dataset size compared to the throttle & brake dataset.

5.3.3 Training Time

As anticipated, the training time for the LTN models is longer than that of the benchmark models. Figures 11, 12, 13, and 14 illustrate that the training time for the LTN models exceeds that of the linear model by an average of 41.8% in the throttle & brake model and an average of 19.8% in the steer model, with the difference generally increasing as the dataset grows larger. A similar trend is observed for the XGBoost model, where the LTN model's training time surpasses that of the XGBoost model by an average of 81.5% in the throttle & brake model and an average of 61.5% in the steer model. The longer training time for the LTN models is expected due to the additional steps in the LTN framework, as outlined in Section 2.1, which are necessary to calculate the loss function described in Section 4.3.

6 CONCLUSION AND FUTURE WORK

The paper aims to implement a practical application for Neuro-Symbolic AI in dynamic path planning for autonomous vehicles. Two Logic Tensor Network (LTN) regression models were developed using the Neuro-Symbolic paradigm—one for controlling throttle and brake parameters, and another for steering. These models were tested and evaluated in the CARLA simulator, demonstrating effective vehicle control in complex scenarios.

The Neuro-Symbolic models were then compared with a linear regression model and an XGBoost regression model using similar datasets and configurations. Evaluation metrics, including Root Mean Square Error and training time, were employed to assess model performance. Results indicated a significant improvement in the RMSE of the Neuro-Symbolic models, particularly with smaller datasets. However, this enhancement came at the expense of longer training times compared to the linear and XGBoost models.

Limitations were encountered during development, due to the computational demands of the CARLA simulator, necessitating a constraint on the number of simulated vehicles. Future studies should explore additional Neuro-Symbolic features, such as explainability, to enhance the analysis of decision-making processes and provide drivers with valuable insights. Additionally, it is worth delving deeper into the symbolic aspects of Neuro-Symbolic AI, incorporating diverse predicates to further refine decision-making by enforcing specific rules.

REFERENCES

Badreddine, S., Garcez, A. d., Serafini, L., and Spranger, M. (2022). Logic tensor networks. *Artificial Intelligence*, 303:103649.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*.

Dworak, D., Ciepiela, F., Derbisz, J., Izzat, I., Komorkiewicz, M., and Wójcik, M. (2019). Performance of lidar object detection deep learning architectures based on artificially generated point cloud data from carla simulator. In *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 600–605.

Gneiting, T. and Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378.

González, D., Pérez, J., Milanés, V., and Nashashibi, F. (2015). A review of motion planning techniques for automated vehicles. *IEEE Transactions on intelligent transportation systems*, 17(4).

Hebaish, M. A., Hussein, A., and El-Mougy, A. (2022). Supervised-reinforcement learning (srl) approach for efficient modular path planning. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 3537–3542. IEEE.

Lavin, A. (2022). Neuro-symbolic neurodegenerative disease modeling as probabilistic programmed deep kernels. In *AI for Disease Surveillance and Pandemic Intelligence: Intelligent Disease Detection in Action*, pages 49–64. Springer.

Sallab, A. E., Abdou, M., Perot, E., and Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *arXiv preprint arXiv:1704.02532*.

Sarker, M. K., Zhou, L., Eberhart, A., and Hitzler, P. (2021). Neuro-symbolic artificial intelligence. *AI Communications*, 34(3):197–209.

Vlachos, E. and Lalos, A. S. (2022). Admm-based cooperative control for platooning of connected and autonomous vehicles. In *ICC 2022 - IEEE International Conference on Communications*, pages 4242–4247.

Zong, X., Xu, G., Yu, G., Su, H., and Hu, C. (2017). Obstacle avoidance for self-driving vehicle with reinforcement learning. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, 11(07-11-01-0003):30–39.