DEVELOPMENT OF A HYBRID AUTONOMOUS CAR SIMULATION MODEL USING ARTIFICIAL NEURAL NETWORKS

BY

CHINWEIKE PRINCE EZIKA

GOU/U22/CSC/834

COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

GODFREY OKOYE UNIVERSITY, ENUGU

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF

BACHELOR OF SCIENCE (B.Sc.), DEGREE IN COMPUTER SCIENCE

SUPERVISOR: DR. FRANK OKEBANAMA

2026

APPROVAL

This serves to certify that the research project entitled "Development of a Hybrid Autonomous Car Simulation Model Using Artificial Neural Networks" was conducted by Chinweike Prince Ezika, bearing matriculation number GOU/U22/CSC/834, in partial fulfillment of the requirements for the award of a Bachelor of Science (B.Sc.) degree in Computer Science, Faculty of Computing and Information Technology, Godfrey Okoye University, Enugu.

_____ _____

Dr. U.F. Okebanama Date

(Supervisor)

_____ _____

Dr. U.F. Okebanama Date

(HOD, Department Of Computer Science And Mathematics)

_____ _____

External Examiner Date

_____ _____

Prof. Ifeyinwa Ajah Date

(Dean, Facuty Of Computing And Information Technology)

CERTIFICATION

I declare that the project work, Development of a Hybrid Autonomous Car Simulation Model Using Artificial Neural Networks was carried out by I, Chinweike Prince Ezika.

_____

Chinweike Prince Ezika

GOU/U22/CSC/834

DEDICATION

This research work is honorably dedicated to my institution, Godfrey Okoye University. It is equally dedicated to my exceptionally loving and supportive family, who selflessly provided everything I needed to succeed throughout my academic journey. Most importantly, this work is dedicated to God Almighty, whose infinite love, protection, and boundless grace sustained me every step of the way.

ACKNOWLEDGEMENTS

My deepest gratitude is due to God for the continuous grace, mercy, and blessings He bestowed on me throughout this final year.

My sincere thanks are due first to my supervisor and HOD, **Dr. Frank Okebanama**, who did his absolute best to ensure that this project was completed perfectly well. I will never be able to fully express my gratitude to him for the priceless time, academic guidance, patience, and kindness he showed me during this final year period. I also express my distinct gratitude to the Dean of the Faculty of Computing and Information Technology, **Prof. Ifeyinwa Angela Ajah**, for her excellent leadership skills, vision, and institutional structures.

My deep appreciation also goes to my lecturers, notably **Prof. Okereke**, **Prof. Echezona**, **Engr. Kingsley**, **Prof. Ozofor**, **Dr. Camilus**, and **Prof. Ogbuike**. These are just a few of the numerous faculty members whose hard work, mentorship, and dedication over the last four years have been evident. They have truly helped mold me into a better computer scientist.

Sincere thanks of appreciation go to my family, who consistently assisted me during the implementation phase of the project and provided the critical emotional stability and technical assistance throughout the duration of this work.

Finally, my thanks go to all the readers of this work. May God bless you all abundantly.

ABSTRACT

Studies of autonomous vehicles have always taken place under controlled road conditions such as clearly marked lanes, consistent road traffic flow, and good maintenance of the road surface. Such studies cannot work under the conditions found in developing countries such as Nigeria, whereby roads are marked by unpredictable road agents, poor state of the roads, lack of lane discipline, and unpredictable human behavior. The research describes the creation of an autonomous car simulation model that works under these conditions. This architecture consists of an Expert System with deterministic Waypoint Path Planning as well as a Long Short-Term Memory recurrent neural network that acts as a reactive controller. The Unity simulation environment used for developing the front-end includes realistic vehicle dynamics and uses a sensor array with 12 distinct sensors, comprising proximity raycasts, object classifiers, kinematics sensors, offset from trajectory sensors, and a roughness detector. The LSTM network is trained via Supervised Imitation Learning from 91,000 expert demonstration examples created in simulation only, with mirroring applied to fix steering bias asymmetry issues. The inter-process communication between the Unity simulation environment and TensorFlow backend was achieved via Shared Memory (mmap), yielding latencies of under one millisecond, which represents a substantial improvement compared to the HTTP approach normally used. A thorough evaluation of the system through structured simulations involving open-road navigation, dense traffic conditions, rough road surfaces, and mixed road conditions showed that the system is capable of generating proper throttle, braking, and steering decisions, resulting in highly accurate steering with an MAE of 0.0198, or less than 0.69° of physical trajectory error per frame when compared to an expert driver. The system was able to generate decisions at 60 FPS+ with sub-millisecond latency even using general laptop-level consumer hardware without any specialized equipment. Through these findings, the current work proves that simulation-based hybrid expert-ANN control can be a useful and feasible avenue toward autonomous navigation for roads currently unexplored by global autonomous driving researchers.

TABLE OF CONTENTS

Title Page	i
Approval Page	ii
Certification	iii
Dedication	iv
Acknowledgements	v
Abstract	vi
Table of Contents	vii
List of Tables	ix
List of Figures	x

Chapter One: Introduction

1.1 Background of the Study	1
1.2 Statement of Problem	3
1.3 Aim and Objectives of the Study	5
1.4 Significance of the Study	6
1.5 Scope of the Study	8
1.6 Limitations of the Study	9
1.7 Operational Definition of Terms	10

Chapter Two: Literature Review

2.1 Introduction	12
2.2 Theoretical Background	12
2.2.1 Autonomous Vehicle Systems	13
2.2.2 Artificial Neural Networks	13
2.2.3 Hybrid Control Systems	14
2.2.4 Waypoint-Based Navigation	14
2.2.5 Simulation-Based Autonomous Driving Models	15
2.3 Review of Related Works	16
2.3.1 ANN-Based Autonomous Vehicle Control	16
2.3.2 Simulation-Based Autonomous Driving Frameworks	19
2.3.3 Hybrid Control Approaches in Autonomous Vehicles	21
2.3.4 Autonomous Driving in Unstructured and Mixed Traffic Environments	22
2.4 Summary of Reviewed Literature	24
2.5 Research Gap	26

Chapter Three: Systems Analysis and Design

3.0 Introduction	28
3.1 System Analysis	28

3.1.1 Analysis of Existing System 28
3.1.2 Limitations of the Existing System 29
3.1.3 Analysis of the Proposed System 30

3.1.4 Benefits of the Proposed System 31
3.2 System Requirements Specification 32
3.2.1 Functional Requirements 32
3.2.2 Non-Functional Requirements 33
3.3 System Design and Methodology 34
3.3.1 Justification for Methodology 35
3.3.2 Method of Data Collection 38
3.4 Unified Modelling Language 39
3.4.1 Use Case Diagram 39
3.4.2 Activity Diagram 41
3.4.3 Class Diagram 42
3.4.4 Sequence Diagram 44
3.5 System Design 46
3.5.1 Simulation Environment Design 46
3.5.2 Neural Network Architecture Design 47
3.5.3 Communication Protocol Design 49
3.5.4 Data Storage and Logging Design 50
3.6 Summary 52

Chapter Four: System Implementation

4.0 Introduction 53
4.1 Development Environment and Tools 53
4.1.1 Programming Languages & Libraries 53
4.1.2 Development Platform & Hardware 54
4.2 Dataset Description and Preprocessing 54
4.3 Model Architecture 57
4.3.1 Input Layer Configuration 57
4.3.2 Hidden Layer Architecture (Recurrent Design) 57
4.3.3 Output Layer Configuration 59
4.3.4 Loss Function and Optimiser 59
4.3.5 Temporal Sequencing and Buffer Management 60
4.4 Model Training 60
4.5 System Modules 62
4.5.1 Python Inference Engine & Shared Memory Bridge 62
4.5.2 Sensor & Telemetry Module 63
4.5.3 Data Logging Module 64
4.5.4 AI Client Module 64
4.5.5 Vehicle Controller 65
4.5.6 Expert System Path-Planning Module 65
4.5.7 Traffic Agent Modules 66
4.6 Testing and Evaluation 66

4.6.1 Evaluation Metrics 66
4.6.2 Module and Integration Testing 67
4.6.3 Simulation Scenario Testing 69
4.7 Results Presentation and Analysis 70
4.7.1 Training Results 70
4.7.2 Dataset Statistics 71
4.7.3 Shared Memory (mmap) Inference Output 71
4.7.4 Simulation Behaviour Output 72

Chapter Five: Summary, Conclusion, and Recommendations

5.1 Introduction 73
5.2 Summary of the Study 73
5.3 Conclusion 74
5.4 Recommendations 75

References 77
Appendices 80

LIST OF TABLES

Table Page

1.0 Training Dataset Feature/Label Schema 55
2.0 Simulation Environment Configuration Parameters 46
3.0 System Non-Functional Requirements Summary 33

LIST OF FIGURES

Figure Page

1.0 Use-Case Diagram of the Proposed System 40
2.0 Activity Diagram for the Proposed System 42
3.0 Class Diagram of the Proposed System 43
4.0 Sequence Diagram of the Proposed System 45
5.0 Unity Simulation Environment 47
6.0 LSTM Neural Network Architecture 48
7.0 Training vs. Validation Loss Curve across 50 epochs 70
8.0 Final MAE and MSE values on the held-out sequential test set 70
9.0 Steering distribution histogram — raw vs. augmented dataset 71
10.0 Sample memory buffer dump showing 12-feature input and 3-float output 72
11.0 Measured IPC inference latency across N test frames 72
12.0 Unity screenshot — autonomous vehicle navigating open road with sensor rays 73
13.0 Unity screenshot — vehicle decelerating behind traffic agents 73
14.0 Unity screenshot — vehicle on rough road segment 73
15.0 Unity screenshot — mixed scenario full lap completion 74

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND OF THE STUDY

Autonomy vehicles, commonly called self-driving cars, have emerged as a prominent technology within intelligent transportation systems. Autonomy vehicles operate automatically without human input, relying on computers to automatically detect the environment around the road and make driving decisions to move the vehicles. The fundamental idea behind the creation of autonomy vehicles is to promote road safety, prevent accidents on the roads, and boost transport efficiency (Shladover, 2018).

The study and invention of autonomous driving systems have been documented over the last several decades. Automated vehicle systems were first conceived in the mid-twentieth century, under controlled conditions on the roads. The most notable achievements were reported in the eighties, as experiments like the Navlab project at Carnegie Mellon University achieved vision-based navigation systems on the roads (Buehler, Iagnemma, & Singh, 2009). Over the years, experiments like the Darpa Grand Challenge have shed focus on the importance of intelligent driving systems that can cope with present day uncertain conditions on the roads.

With increases in computing capabilities along with available data, Artificial Neural Networks (ANNs) have emerged as prominent models for autonomous vehicles. The entire ANN structure resembles an ordinary human brain. The ANN is capable of learning information from given situations. This capability differentiates it from rule-based models (Goodfellow, Bengio, & Courville, 2016). In autonomous vehicles, an ANN is generally implemented for functions such as

controls for steering, object avoidance, speed limitations, and decision-making. Notable breakthroughs in such models reveal that ANNs are strongly capable of handling situations while in operation compared to normal rule-based models (Thrun et al., 2020).

Recent research has developed new hybrid autonomous vehicle navigation methods, enabling the use of hybrid autonomous vehicle control techniques. Hybrid, as used in these recent studies, signifies the integration of waypoint expert system navigation and ANN adaptive navigation. As such, it can be defined as the integration of route navigation techniques with machine learning techniques, as recently implemented. This will allow the autonomous vehicle to follow the route as planned but still ensure flexibility to respond to changes in the environment (Pendleton et al., 2017).

Despite these developments, almost all the current autonomous vehicle systems and simulators have been developed and tested under structured road conditions. Typically, all such road environments would presuppose adequate markings of lanes, highly predictable traffic flow, well-maintained road surfaces, and strict adherence to traffic laws. These assumptions can pretty much hold good in most of the developed parts of the world where, first of all, the infrastructure for roads hardly contains any hindrance and, secondly, the enforcement related to road traffic is highly effective in maintaining order on the roads (Shladover, 2018). They are pretty misrepresentative of the actual driving conditions seen in most developing nations.

In Nigeria, the road environment is mainly unstructured and highly unpredictable. Traffic congestion within urban areas is heavy, lane usage is irregular, infrastructure is poorly maintained, and interactions among vehicles, motorcycles, pedestrians, and informal public transport operators are numerous. Lane markings are frequently absent or disregarded, and traffic behavior often does

not align with formal traffic rules. Reports from the Federal Road Safety Corps confirm that traffic crashes in roads continue to be one of the major concerns in public safety in Nigeria, and this has greatly contributed to injuries and fatalities annually (FRSC, 2024).

Models that have been trained and tested in structured environments tend to perform poorly in unstructured environments. Systems that depend on set and specific rules find it hard to respond to sudden changes in lanes, unexpected curbs, potholed roads, and disorderly traffic flow. This demonstrates a huge research gap, especially regarding the design of autonomous driving systems for the technically challenging environments in the developing world, where disorderly traffic flow is the order of the day (Sivakumar & Nagarajan, 2024).

One way to overcome this challenge through simulation-based research is characterization as safe and cost-effective, which helps to evaluate autonomous vehicles in hazardous, unpredictable, as well as unsafe situations without putting human lives or physical infrastructure at risk. Simulations can be used to test vehicles as many times as possible, depending on information regarding diverse traffic situations, which makes it possible for ANN control systems to be trained through simulation-based environments, as explained by Dosovitskiy et al. (2017).

It is because of these considerations that the focus of this particular study is based on the development of a simulated autonomous vehicle based on the concept of waypoint expert systems navigation accompanied by Artificial Neural Network controls, primarily for handling unstructured roads, a situation that is quite common in Nigeria. Thus, the overall focus of the study is based on providing autonomous vehicle systems that are used for handling chaotic and disorderly roads.

1.2 STATEMENT OF PROBLEM

Despite considerable research and development in the area of autonomous vehicle systems, most of the existing systems are designed and developed within a controlled and organized road setting. In this regard, autonomous systems are assumed to operate within an environment of well marked roads, anticipated behavior of other traffic participants, and well maintained roads. However, these kind of assumptions are not common in most developing nations including Nigeria, considering that several challenges would not yet be appropriately treated by simulation models of autonomous vehicles:

1. **Dependence on Structured Road Assumptions :-** Almost all available self-driving systems have been designed to work with structured road system layout and markings, as well as traffic regulations that are rule-based. The design environment in Nigeria is not structured due to lack of usage standards on the roads, making self-driving systems less effective on irregularly shaped roads and marked traffic flow regulations.
2. **Limited Adaptability to Unpredictable Traffic Behavior** :- The existing models function using predetermined rules or control strategies in a deterministic form, which cannot effectively address the very dynamic conditions of the traffic in the Nigerian urban environment made more complex by the presence of motorcycles, tricycles, pedestrians, and erratic driving behaviours.
3. **Inadequate Representation of Poor Road Conditions** :- Many models of simulations neglect the road quality issues that may include potholes, road surfaces, and damaged road areas that tend to influence the control and stability of the vehicle but are not included in the simulations for the ideal setting for the autonomous vehicle.

4. **Insufficient Use of Adaptive Learning Based Control** :- Although there have been applications of machine learning methodologies in autonomous cars, most of them are embedded with rule-based control systems. This reduces their effectiveness in being able to learn from the environment and adapt to unstructured road environments, which are prevalent in Nigeria.

1.3 AIM AND OBJECTIVES OF THE STUDY

The purpose of this study is to develop a hybrid autonomous car simulation model for an autonomous vehicle using Artificial Neural Networks (ANN).

The specific objectives are to:

1. Analyze existing autonomous vehicle simulation models.
2. Design a resilient simulation architecture which integrates waypoint routing and reactive control capabilities provided by ANNs.
3. Simulate the designed hybrid model using Unity Engine.
4. Test realistic scenarios for traffic in Nigeria with unpredictable movements of vehicles.

1.4 SIGNIFICANCE OF THE STUDY

The importance of the need to develop the hybrid simulation model of the autonomous vehicle based on the Artificial Neural Network, which is capable of coping with the unstructured chaos of the Nigerian road environment, cannot be overemphasized as it goes beyond the scope of the academic exercises proposed in the study. The importance of the study can be appreciated in the sense of the lives it has the potential to save and the intelligent systems it has the potential to develop, and the future where the technology listens because the reality is what it speaks. The importance of the study can be appreciated in the following ways:

- **Every Nigerian Road User**: From the go-slow traffic in Lagos to the okada riders on the roads, and from the danfo passengers to the pedestrians on the roads, everyone faces the danger of traffic on Nigerian roads. This simulation demonstrates how an autonomous system can learn to deal with sudden changes in lanes, dodging potholes, and the unpredictable ways of human drivers on our roads. The over 3,400 casualties recorded on Nigerian roads between January and September 2025 alone, according to the FRSC, are a clear indication of the need for crash avoidance systems in vehicles.
- **Policymakers, FRSC, and Transport Authorities**: This is a virtual "test bed" where valuable data can be collected without the loss of life or vehicles. We can observe the reaction of the AI to local problems such as poor road surfaces, the fluidity of lanes, and the presence of motorcycles. This will help create more intelligent regulations, better road designs, and more informed decisions regarding the intelligent driving assistance programs. Since speeding and distractions are the major causes of accidents in the country,

such research provides valuable data to invest in technology suited to the local environment instead of adopting technology created for the well-ordered roads of the West.

- **Researchers, Lecturers, and Students**: This project shows that quality research in the field of AI is possible right here in Nigeria. The integration of waypoint navigation, neural network control, and road roughness modeling provides a foundation on which others can continue to add value. It provides students and researchers with a framework which can help to prove to the world that developing world constraints are not barriers to innovation.
- **Local Automotive Innovators (Innoson, Naseni, Startups)**: The Nigerian auto manufacturing sector, as well as tech entrepreneurs, will find the plan useful in adding intelligent features to their made-in-Nigeria vehicles. This hybrid control plan has the potential to balance fuel efficiency with safety, especially on rough roads, making autonomous features accessible for the mass market vehicles and even commercial fleets. This is a new opportunity for investment, partnership, and export to other African countries where the settings are the same.
- **Global Road Safety Advocates (WHO, FIA Foundation, UN Agencies)**: International partners monitoring Nigerias high rates of fatalities on the roads have found that standard autonomous vehicle technology is not an option for the country. This research focuses on simulation as a viable and affordable technology for developing countries to deal with unstructured roads and traffic situations.

1.5 SCOPE OF THE STUDY

The scope of the research is limited to designing a simulation model for a hybrid autonomous car using Artificial Neural Networks. The research is applicable to an unstructured system of roads as seen in Nigeria.

The report examines:

- Simulation of the movement of one hybrid autonomous vehicle through virtual routes based on waypoint routes combined with the ANN-based control methods.

- Processing the major inputs for the ANN: the distance to the next waypoint, the distance to the front, left, and right obstacles, the roughness of the road, normalized to a scale from 0 to 1.

- Mid-way addition of Nigerian-type chaotic elements: erratic motorcycle or tricycle, sudden pedestrian crossing, fluid lane change, dense congestion, potholes, irregular terrain, etc.

- Used in a physics simulation framework to display motion processes and compute control actions such as throttle, brakes, and steering.

- Performance evaluation based on quantitative parameters such as the rate of collision avoidance, road compliance, reaction time, and adaptability, considering various traffic densities and types of routes.

1.6 LIMITATIONS OF THE STUDY

“No research project of this kind takes place within absolute boundaries, particularly within the confines of an end of the year undergraduate project.” Though this model has opened new and

meaningful boundaries in the encapsulation of the Mayhem that occurs in the roads of Nigeria, some honest limiting factors were set in place:

The entire system works in a virtual reality setting only. It was impossible for the team to make use of real-time hardware and sensors or test the vehicle in real-time. The implications of this are obvious: the system works in an ideal setting with the best physics engines available, while in reality the inputs would be camera, lidar, or imu sensor noise from a real autonomous system.

Computer resources imposed natural limits. Such intensive tasks involving complex ANN models in hundreds of different scenarios required balanced considerations between network depth, batch sizes, and levels of simulated detail. Much more complex models or extended periods of training epochs could potentially yield even stronger generalization ability, but these factors kept things realistic.

The traffic and road situations, as realistic as they are rendered to reflect the situation in Nigeria, are abstract. Factors such as weather conditions (rain making roads muddy), weathering conditions at night, cultural aspects of driving intentions (horns and hand signals), as well as shared modality usage (hawkers on highways and animals on the roads) were avoided.

“The training set for the ANN consisted of generated instances, guided by local crash reports, traffic video observations, and published statistics, rather than massive, country-specific real-world instance collections.” This brings about the possibility that the network could possibly overlook some of the edges it may encounter on the roads of Lagos and Onitsha.

Finally, the evaluation remains relevant to simulation metrics of collision counts, path deviation, and response times without attempting a direct comparison with those of human drivers or existing

commercial AVs in the same Nigerian-like environment because, at this stage, no comparable datasets of the tests exist.

These constraints do not reduce the work; instead, they establish the legitimate boundaries of it. These boundaries immediately indicate the ways forward, and that is hardware integration, more data available locally, more environmental variables, and validation. What is before us is a start that is realistic given its constraint and ambitious enough to encourage the next steps.

1.7 OPERATIONAL DEFINITION OF TERMS

In order to ensure that nothing is unclear as we move further into this project with this series of writings, I want to define what I mean by certain terms that I will refer to throughout this project. Here are those key terms:

- Autonomous Vehicle

A software that simulate a car capable of sensing its virtual surroundings, planning routes, and controlling its self-movement-steering, speed, and braking-with little or no human input. In this paper, the term strictly refers to the modelled entity inside the simulation environment.

- Artificial Neural Network (ANN)

“A computational structure only very loosely based on the connections and learning processes of biological neurons.” Here, it acts as the brain of a simulated vehicle, processing several sensor-like inputs to make smooth and adaptive decisions regarding throttle, brakes, and steering, depending on unpredictable road conditions.

- Hybrid Autonomous Car

The simulated vehicle utilizes a dual-layer hybrid architecture: i) a deterministic **Expert System** developed in C# that handles high-level path planning and trajectory following; ii) a reactive **Artificial Neural Network (ANN)** served via a **low-latency Shared Memory Bridge** to execute instant obstacle avoidance. The term captures the fusion of rule-based logic for navigational stability and deep learning for dynamic responses to complex road environments.

- Unstructured Road Environment

Road and traffic behaviours that defy rules, signs, and predictability are precisely the type of driving that many Nigerians engage in or encounter on a daily basis: no marked lanes, motorcycles suddenly massing, pedestrians cropping up anywhere, potholes, rough road surfaces, go-slow congestions turning into confusion, and drivers continuously improvising. Such describes the reverse of the disciplined roads of marked lanes and traffic signals.

- Waypoint

“A simple sequence of predefined points on the virtual map that would roughly outline a route similarly to the way one drops pins on a Google Map. The car will attempt to stay on these loosely while the ANN performs all the evasion to stay on course.”

- Sensor Fusion

The act of integrating various bits of "sensor" data (front obstacle distance, side clearance, road roughness level, current speed, distance to next waypoint) into a unified understanding that the

ANN can respond to. It simulates how the human driver might use his eyes, ears and sense of the road all at once.

- Road Roughness

A single figure between 0 and 1 indicating just how bad the surface is at a given point on the road. A figure of 0 means the road is smooth, while a figure of 1 means it is full of bone-jarring potholes. This figure will affect how aggressive or gentle the throttle, brake, and steering actions are made by the ANN.

- Throttle

The simulated accelerator is used to control the ANN's outputs, which will vary between -1 for full reverse gear and +1 for full forward acceleration. The ANN will use it to decide the amount of power to use at any given time while driving.

- Brake

The amount of simulated braking, which will vary between 0, no braking at all, to 1, maximum braking for the emergency stop function. The ANN will use it when the road turns hazards due or when there are roadside obstacles on the road.

- Steering Angle

The left/right turning command from the ANN: The numerical value varies from -1.0, full left, to +1.0, full right. It varies smoothly with speed, upcoming turns, obstacles, and road conditions.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter examines the area in which scientists and engineers have been working in the world of self-driving vehicles, neural networks, and simulation-based control. There is nothing complicated about this aim: it is to identify the ground that has been covered, determine the shortcuts that are often used in studies, and point out the flaws. These flaws widen as soon as the road ceases to be well-behaved.

Since the past twenty years, the reality of autonomous vehicles has grown from a lab curiosity into a scaled prototype capable of navigating highways with a strange level of accuracy. The family of papers on the subject overflows with stories of the successful implementation of vision (detecting lanes and reading road signs), path planning (finding safe routes), control (steering and adjusting speeds in a smooth motion), and decisions informed by AI. All of these were created under the assumption that all the external elements would be predictable, from the lane markings on the ground to the car speeds around them, a safe assumption given the controlled roads of North America, Europe, and East Asia.

"Artificial Neural Networks are the core technology behind many innovations." They are useful for learning directly from data, rather than being "programmed with rules: Pattern recognition in camera images for lane keeping, predicting pedestrian intentions based on movement, and regulating the throttle for heavy traffic." The majority of research combines ANN with a rule-based controller to form a "hybrid system." The rule-based controller is used for basic control, while the ANN offers flexibility for corner-case behavior. The simulation study is the key aspect

of this field. This allows a scientist to "train and validate their models without damaging physical vehicles." However, if research is delved into further, a gap is observed. There is a lack of consideration of road conditions when they resist expectations. In developing countries like Nigeria, driving is done in a language that "does not exist in developed nations. Lane lines are either non-existent or faded. Motorcycles weave through traffic with fluidity. Pedestrians use roadways as sidewalks. Some potholes require adjustment with every passing car. ‘Mini buses’ stop wherever and whenever." This is not an exception; it is a rule. This is never considered while establishing world standards. This is why vehicles fail when these conditions change to a fluid state. Studies from Africa (Nigeria, Ghana, South Africa, and Ethiopia) recently indicate just that: traditional AV research does not adapt well.

This review, therefore, looks at the major strands of autonomous simulation frameworks, control strategies using ANNs, and hybrid strategies, but always with these underlying questions of what the underlying road layouts are, what the underlying patterns of human movement that are hardcoded into these simulations are, and at what point these start to break down in unstructured environments. Through these various strands, it becomes clear what this project is filling a gap on that a simulation is not merely one that is resistant to Nigerian levels of disorder, but one that positively thrives in it; combining waypoint and neural network approaches.

2.2 THEORETICAL BACKGROUND

In this section, some of the important ideas and the technology that binds all of these together will be discussed. They include: autonomous control of cars, artificial neural nets, hybrids, waypoint routing, and simulation modeling. Understanding these concepts will help you grasp the reason why a smart car can think and reason effectively in a real-world environment, as seen in most developing countries like Nigeria.

2.2.1 Autonomous Vehicle Systems

The simplest definition of an autonomous vehicle is a machine that can observe the world around it, decide what to do, and then do it, without a human hand ever touching the steering wheel. The most commonly discussed model is a closed loop, where the machine observes the world, decides what to do, and then does it.

All the literature we have read assumes a rational world. It assumes we can see the lane markers, it assumes we can anticipate the flow of traffic, it assumes we can follow a smooth speed, it assumes a world that is easier to understand mathematically, a world where our math looks magnificent on a closed test track, or even a highway in a developed country. But it fails to assume a world that looks anything like Lagos. In Lagos, lane markings disappear or become irrelevant. Suddenly motorcycles materialize out of nowhere. Pedestrians use our roads as sidewalks. Potholes create a need for constant correction. It is a true world that our project must cater to.

2.2.2 Artificial Neural Networks

Neural nets are essentially machines that learn from examples and concrete instructions. It's possible to replicate the way a brain processes patterns with layers of simple computing units ('neurons') connected appropriately. Give them enough examples of data readings and the corresponding driving maneuvers, and they learn to establish the hidden connections themselves: to soften braking when the approaching gap gets smaller, to be alerted to a shadow on the road which can be a pothole, to moderate the turn of the wheels depending on the speed.

What ANNs are so good at in driving is called generalization. They know how to act appropriately in completely new situations if those patterns are similar to those they encountered when they were learning. This is very helpful in a tidy environment. This is extremely useful if the

environment is well-ordered. If the environment is untidy, it is essential. If the traffic follows the pattern of water seeking the path of least resistance, logical if-then statements quickly break down.

2.2.3 Hybrid Control Systems

But the reliability of pure rule-based control is only as good as the reliability of the world: it will work perfectly as long as the distance remains the same, the course remains the same, the car will stop at the same red light. But the moment a child dashes out from behind another car, or a car suddenly cuts in from an unseen lane, the system will either freeze or over-react. Pure neural control can be wonderfully flexible, but it can also be unstable or even unpredictable, at least in the initial stages.

Hybrid control attempts to blend the best of both. Rule-based logic handles the predictable, reliable tasks: keep a safe distance, stay on course. Neural control handles the complex, unpredictable tasks, learning from experience how best to adjust gas, brakes, and steering on the fly. The end result is a system that is both reliable and intelligent exactly what unstructured roads need.

2.2.4 Waypoint-Based Navigation

Rather than depending on the precise detection of the lanes, the waypoint system allows the vehicle to be given a general guideline on the route it should take by providing it with a series of coordinate points that define the path it should take. This is akin to putting digital marks on a map instead of staying within the lines. It should be able to move smoothly from point to point but it does not need to stay on course. It will meander to the left and right to avoid an object in its path, slow down because of traffic, smooth out the ride to avoid bumps, etc., as long as it continues to progress towards the next waypoint.

This works well in situations where there are no clear structure markings in roads. Waypoints are able to follow irregular paths, pass around markets, and pass through Go Slow areas. When combined with neural control, these waypoints are able to achieve high-level intention but at the same time are accomplished in terms of low-level improvisation thereby creating a balance between purpose and flexibility.

2.2.5 Simulation-Based Autonomous Driving Models

Real-world experiments are costly, risk injury, and are the result of a long process. This all changes with simulations. A world where cars behave in a realistic way, the layout of the road, traffic agents, and noise from the environment can all be created, and thousands of test instances can be processed in a fraction of the years it would have taken in the real world. This system would have the benefit of crashing many times with no penalty, varying parameters in an instant, and assessing all aspects of a test, such as the proximity of a crash, the road trace error, response

In the context of this project, simulation is not a method, but the method itself is the only ethical approach to analyzing Nigerian road traffic congestion. We can generate pothole patches, bikers, and pedestrian dashes without putting any life at risk. The concept in this case is quite simple yet potent: by being able to experiment and test in an accurate simulation model, we get to learn fast and determine the efficacy before actually deploying something on the roads, which, in many parts of the world, are forgiving neither.

2.3 REVIEW OF RELATED WORKS

Autonomous vehicle research has led to a plethora of research on simulation environments, neural networks for control, and the problems with mixed traffic. In order to keep this review brief and pertinent to our goals, the most applicable research is divided into categories that make sense.

While all these works contribute valuable information to the field, it appears that nearly all were conducted under a very structured and predictable environment, leaving the complete chaos of Nigerian roads largely unexplored.

2.3.1 ANN-Based Autonomous Vehicle Control

Artificial Neural Networks (ANNs) are widely used in autonomous driving due to their ability to model complex nonlinear relationships between perception inputs and control outputs.

Su et al. (2024) introduced an intelligent vehicle lateral tracking algorithm that used neural network predictive control. They combined radial basis function neural networks and model predictive control to improve the efficiency and accuracy of the algorithm. The simulations were conducted on structured curved roads. The research was limited to structured curved roads and did not account for extreme unstructured road conditions such as potholes and heterogeneous traffic. This is another contribution to ANN-based control algorithms and their efficiency on unstructured roads in the Nigerian environment.

Tsur et al. (2023) presented the neuromorphic realization of the most commonly used autonomous vehicle controllers: pure pursuit, Stanley, PID, and MPC. The authors have used spiking neural networks to simulate the results, which have similar accuracy with low computational complexity. This is the second contribution to the field of ANN-based control algorithms. The research was limited to structured racecourse roads and did not account for extreme unstructured road conditions.

Shao et al. (2023) gave an in-depth overview of deep learning architectures used in autonomous driving, such as CNNs, RNNs, LSTMs, and transformer networks, with regards to real-time

perception, decision, and control. This further validates the significance of deep learning in controlling AVs while also indicating a limitation in generalization for chaotic roads in developing nations.

Li and Okhrin (2024) also introduced a robust DRL architecture with platform-dependent perception modules for effective training of agents for lane-following and overtaking, enabling seamless transfer between simulated and real-world environments with few modifications. This validates our choice of ANN control while also emphasizing the need to address real-world conditions.

Alam et al. (2024) introduced a conceptual framework for explainable end-to-end autonomous driving, including transparency issues in deep neural networks, with a discussion on visual explanation techniques for CNNs. This is important for our ANN model, as it focuses on the importance of interpretability in a complex, unstructured environment.

Karaagac et al. (2024) discussed a review of third-generation spiking neural networks (SNNs) for autonomous driving, with a focus on energy efficiency and sustainability benefits, including those over traditional CNNs and RNNs, for perception, decision, and control. This adds more dimensions to ANN applications, especially in simulation, for low-latency requirements, although not tested in mixed, chaotic traffic like Nigeria.

Bojarski et al. (2016) introduced an end-to-end deep learning solution that directly linked raw camera images to steering commands for autonomous vehicles. The authors research proved the capacity of neural networks to learn autonomous driving successfully without need for rule-based programming thereby ensuring stable lane following in both real-world and simulated

environment. Nevertheless, the proposed solution was tested mostly on roads with clear markings, making it unsuitable for unstructured road settings.

Chen et al. (2017) analyzed deep neural network models for autonomous driving in an end-to-end scenario, concentrating on control accuracy and robustness. The authors results revealed that ANN-based controllers can potentially perform better than modular architectures especially in complex environments. However, most experiments were carried out in structured settings with predictable traffic patterns.

Kiran et al. (2019) conducted a thorough survey on deep learning solutions for autonomous driving and concluded that learning-based control strategies always perform better than rule-based approaches in complex settings. The authors also pointed out that most existing datasets and experiments were carried out on highways and structured urban roads.

2.3.2 Simulation-Based Autonomous Driving Frameworks

Simulation environments play a critical role in autonomous vehicle research by enabling safe, repeatable, and cost-effective testing.

Berta et al. (2024) analyzed resource-consuming training of deep reinforcement learning agents for low-speed autonomous driving simulations in Unity, with two case studies of garage parking and obstacle-dense navigation. This study is relevant to our choice of Unity for custom agent behaviors, as well as its potential for high-chaos Nigerian driving conditions.

Marinkovic et al. (2024) studied the AWS DeepRacer platform for training end-to-end autonomous driving agents with reinforcement learning, achieving 95.8% reduction in collision

rates and 78.9% reduction in lap times with optimized sensor configurations. This study contributes to simulation platforms, particularly with RL optimizations, which are relevant to our ANN training in chaotic conditions.

Badue et al. (2021) implemented a large-scale survey of research works on self-driving cars, with a focus on simulation-based evaluations of autonomous driving agents. The research revealed that many research works fail to take into consideration road degradation, irregular road shapes, and chaotic traffic conditions, which are vital for driving conditions.

Juliani et al. (2019) also implemented a study on reinforcement learning and neural control using the Unity ML-Agents toolkit. The study revealed that game engine-based simulations are efficient for training autonomous agents in various environments, while road disorder is an area to be explored.

Dosovitskiy et al. (2017) introduced a new open-source simulator named CARLA, which is focused on autonomous driving. CARLA simulates realistic urban scenes, dynamic weather, and traffic agents. CARLA has become a standard benchmarking platform, although most studies using CARLA are based on regular traffic flow with clear road lanes.

2.3.3 Hybrid Control Approaches in Autonomous Vehicles

Hybrid control approaches combine learning-based adaptability with the stability of traditional control systems.

He et al. (2024) developed a defense-aware robust reinforcement learning (DARRL) method for worst-case observational perturbations. An adversarial attacker and robust defender module have

been implemented in SUMO simulations. Results have shown improved robustness of the policies in chaotic traffic conditions. This is similar to our strategy of using a hybrid ANN-rules approach for dealing with the unpredictability of the environment.

Zhang et al. (2020) developed a hybrid ANN-PID control strategy for the motion control of autonomous vehicles. The results showed improved stability and reduced error in the control of the autonomous vehicles. However, the experiments have been conducted in a structured environment.

Shalev-Shwartz et al. (2018) developed the Responsibility-Sensitive Safety (RSS) framework. This framework combines safety rules with learning-based driving policies. Although this framework has shown good results in the enforcement of safety rules, it assumes that the surrounding agents comply with the safety rules.

2.3.4 Autonomous Driving in Unstructured and Mixed Traffic Environments

Research explicitly addressing mixed and unstructured traffic environments remains limited.

In their study, Yurtsever et al. (2020) examined the general practices and emerging technologies in the context of autonomous driving and found that most of the practices and technologies were implemented for structured traffic conditions. They highlighted the difficulty in the deployment of autonomous driving in conditions where the traffic is irregular.

Grigorescu et al. (2020) examined the deep learning techniques in the context of autonomous driving and highlighted the lack of strong solutions in conditions where the infrastructure is poor and the traffic is heterogeneous.

In their study, Mekonen and Tesema (2024) examined the prospects of AV in developing countries, especially Ethiopia, and found that the perceived usefulness was high, at 78.8%, but the perceived ease of use, especially in conditions where the infrastructure is poor, is a concern.

Wang et al. proposed a lightweight bilateral semantic segmentation network with a dual attention mechanism, including ECA and CA, based on the BiSeNet framework for real-time drivable area detection on unstructured roads with indeterminate boundaries and complicated backgrounds. This is directly applicable to our proposed ANN model to handle pothole-filled roads.

Adeyinka and Oluwaseun (2024) discussed the challenges and opportunities of autonomous vehicle implementation in Nigerian urban settings, highlighting the importance of advanced AI, machine learning, and real-time data analysis to cope with the chaotic traffic conditions and infrastructure. This is directly applicable to our hybrid model.

Afolayan et al. (2023) reviewed theoretical and practical issues of autonomous vehicle implementation in Africa, including diverse traffic, informal transportation networks, variable regulations, and economic factors peculiar to African cities. This is directly applicable to our study on unstructured Nigerian settings.

2.4: Summary of Reviewed Literature

S/N	Author(s) & Year	Technique / Model Used	Environment (Simulation / Real-world)	Key Contribution
1	*Bojarski et al. (2016)*	End-to-End ANN (CNN-based steering control)	Real-world data + simulation	Demonstrated that neural networks can directly map sensory input to steering commands
2	*Dosovitskiy et al. (2017)*	Deep Neural Network for vehicle control	Simulation (CARLA)	Introduced modular and end-to-end learning for driving tasks
3	*Sallab et al. (2017)*	Deep Reinforcement Learning (ANN-based)	Simulation	Showed ANN capability in decision-making and control
4	*Kendall et al. (2019)*	ANN with uncertainty modeling	Simulation + real-world datasets	Improved robustness of neural control systems under uncertain perception

5	*Chen et al. (2020)*	ANN-based behavioral cloning	Simulation	Demonstrated effective imitation learning for driving control
6	*Shalev-Shwartz et al. (2018)*	Rule-based + ANN hybrid system	Simulation	Combined learning-based perception with rule-based safety constraints
7	*Kiran et al. (2021)*	Deep Learning survey for autonomous driving	Simulation and limited real-world	Highlighted ANN dominance in perception and control tasks
8	*Li et al. (2022)*	ANN-based trajectory planning	Simulation	Improved path planning using neural networks
9	*Hussein et al. (2023)*	ANN controller for autonomous vehicle navigation	Simulation	Demonstrated ANN effectiveness for steering and speed control

10	*Singh & Mohan (2023)*	Neural network control in mixed traffic	Simulation (developing-country traffic models)	Addressed interaction with heterogeneous traffic agents
11	*Al-Khatib et al. (2024)*	ANN-based autonomous driving framework	Simulation + small-scale real-world tests	Demonstrated feasibility of ANN control beyond ideal road conditions
12	*Su et al. (2024)*	RBF neural network predictive control	Simulation	Improved lateral tracking with optimized RBF and Adam
13	*Tsur et al. (2023)*	Spiking neural networks for controllers	Simulation	Energy-efficient neuromorphic approximations of traditional controllers
14	*Shao et al. (2023)*	Deep learning frameworks review	N/A (review)	Comprehensive analysis of DL in AV perception and control

15	*Li and Okhrin (2024)*	Platform-agnostic DRL with perception module	Simulation + real-world	Robust Sim2Real transfer for lane following and overtaking
16	*Alam et al. (2024)*	Explainable AI framework for AV	N/A (review)	Conceptual model for transparency in DNNs
17	*Karaagac et al. (2024)*	Spiking neural networks review	N/A (review)	Energy efficiency in perception and control tasks
18	*Berta et al. (2024)*	DRL agents in Unity	Simulation (Unity)	Low-speed maneuvering in obstacle-dense settings
19	*Marinkovic et al. (2024)*	DRL in AWS DeepRacer	Simulation	Sensor and reward optimization for collision reduction

20	*He et al. (2024)*	Defense-aware robust RL (DARRL)	Simulation (SUMO)	Robustness against observational perturbations in traffic
21	*Yang et al. (2024)*	DRL for car racing	Simulation	PPO/DQN with transfer learning and RNNs for policy stability

2.5 RESEARCH GAP

From the literature review, it is evident that there is a sophisticated world of designing ANNs and hybrid intelligence, and simulation complexity; however, there is a research gap when these complexities are juxtaposed with the pure unpredictability of unstructured roads. There are patterns in the literature that are subtle but unmistakable; they point to the ways in which the literature has conquered complexity and hovered on the edges of embracing the full complexity of disorder.

Most of the literature that has been reviewed is underpinned by a structured environment: clean roads, clean traffic, clean asphalt. While there is new literature emerging, such as Singh and Mohan (2023) and Al-Khatib et al. (2024), that begins to take into account the diversity of traffic, and Mekonen and Tesema (2024) that points to the infrastructure challenges within developing countries, full-bodied research into the pure unpredictability of pothole mazes, okada traffic, unstructured creativity, and stop-and-go traffic is scarce. Research into the Nigerian experience of this mix, as reflected in Adeyinka and Oluwaseun (2024) and Afolayan et al. (2023), is woefully

underrepresented, with models designed for highway conditions being woefully inadequate for the realities of Lagos and Onitsha.

Hybrid approaches also offer this resiliency, as seen in Li et al. (2022) and Shalev-Shwartz et al. (2018), with extensions such as He et al. (2024) offering resiliency to perturbations. But the extent of their handling of non-compliant traffic has yet to be seen. What chance do we have in a world that is chaotic by the norm rather than the exception?

Simulators such as CARLA, developed by Dosovitskiy et al. (2017), offer a very realistic experience but are based on highly organized and structured cityscapes. What chance do we have with platforms that offer this kind of adaptability, such as the Unity Engine for simulating the chaos of the developing world, especially with the latest advancements in serving ANNs with low-latency **Shared Memory Bridge (mmap)** i.e same RAM block? Berta et al. (2024) touches on the capability of the Unity Engine for low-speed agents, while Marinkovic et al. (2024) shows RL in racing simulators.

Furthermore, there is a lack of work that combines waypoint expert systems navigation with ANN reactivity for sims developed for patterns of the developing nations. Qin et al. (2025) and Rasouli et al. (2024) have conducted surveys on the trends of path planning. Lorente et al. (2023) have advanced waypoint navigation for industrial applications. Borowczyk et al. (2024) have conducted a survey on obstacle avoidance. Wang et al. (2025) have conducted a survey on the parallels of UAVs. Aldibaja et al. (2024) have addressed the issue of waypoint transfer. Ramirez-Robles et al. (2024) have addressed the issue of off-road path planning. Physics-based simulation with ML serving infrastructure for unstructured roads such as those in Lagos, Onitsha, and Kano has yet to be explored.

The gaps identified in the literature have been addressed by this work, which has developed a hybrid simulation model that combines waypoint expert systems navigation with ANN reactivity within the Unity Engine and has used low-latency **Shared Memory Bridge (mmap)** for the serving infrastructure pipeline and communication between the brain the and unity simulation environment. This work has gone beyond the tolerance of chaos and has attempted to learn from the patterns based on the traffic pulse and the roads that we all travel on a daily basis.

CHAPTER THREE

SYSTEMS ANALYSIS AND DESIGN

3.0 INTRODUCTION

The explanation concerning the analysis and design of the proposed system can be found in this chapter. It refers to the system design process and the approach used in that process, like the "hybrid approach to prototyping and the use of the spiral methodology," along with its application in the study.

3.1 SYSTEM ANALYSIS

System analysis is a step-by-step approach of breaking down a proposed system and examining its component parts and interfaces of these components. The proposed approach for conducting this research involves a deep analysis of a proposed system for a hybrid autonomous car simulation model. This will include analysis of other autonomous car systems and identifying their weaknesses in unstructured environments.

Detailed design involves developing various kinds of charts, which include use case diagrams, activity diagrams, class diagrams, and sequence diagrams, among others. The significance of these diagrams is they provide a visual depiction of various components of the system, along with their relationship to one another, that improves one’s understanding of the system.

Other than this, the system analysis process too plays an important role in deciding what needs improvement. By analyzing during this process, it will be ensured that the system is stable, efficient, and possesses adequate strength to prevent limitations of existing systems.

3.1.1 ANALYSIS OF EXISTING SYSTEM

The most common technologies in autonomous vehicles, or rather the modes that have been widely researched, are those that assume structured and predictable road environments. These technologies combine different input sensor technologies such as camera, LiDAR, radar, and GPS technologies. There are also perception algorithms that can detect road markings, signs, and other relevant road objects. The technologies also have planning components for appropriate paths and appropriate control technologies. The control technologies use PID or model control for appropriate steering, accelerating, and braking.

For simulation worlds like CARLA or on the highway or well-kept roads, these approaches have been seen to be accurate excelling at keeping to a lane, avoiding obstacles smoothly, and following traffic rules with precision. Many of the ANN-based approaches have a direct mapping to the controls and show a high level of performance if the world acts as it should be. The rule-based approaches have a number of safety rules on top of this to provide a degree of stability.

Looking at the above statement with regard to the needs of the unstructured road environments, especially with regard to developing nations like Nigeria.

3.1.2 Limitations of the Existing System

The current autonomous vehicle systems reveal several fundamental weaknesses when applied to unstructured and chaotic road conditions:

1. **Over-dependency on structured road assumption**

   Most model designs are premised on the assumption that there are visible lane markings, traffic flow patterns, and smooth driving surfaces. In the absence or non-adherence to lane use on many Nigerian highways, the perception and planning algorithms would not function optimally.

2. **Limited adaptability to unpredictable traffic behaviour**

   Rule-based controls and even many neural network approaches break down in the face of erratic movements by motorcycles, tricycles, informal buses, and pedestrians crossing with little warning or following informal rules. These systems are not trained or tested on fluid, non-deterministic traffic flows extensively.

3. **Inadequate handling of poor road conditions**

   In addition to this, the effects of potholes or road irregularities, along with the roughness of the road, are rarely accounted for. This creates problems with the stability of the vehicle models and their dynamics.

4. **Insufficient generalization in chaotic environments**

   Such models, pre-trained on an ordered dataset from developed countries, are furthermore observed to perform comparatively worse on such heterogeneous and-rule-bending traffic. For models, which are pre-trained on such datasets without specific context, it would be difficult to perform on such improvising traffic as experienced in Lagos or other Nigerian major cities.

3.1.3 Analysis of the Proposed System

The hybrid model for the simulation of an autonomous car proposed here would, therefore, be targeted at the core weaknesses in existing systems and ensure useful advances in the simulation of autonomous driving for unstructured Nigerian road environments. Specific objectives take directly from the limitations of the current approaches and aim at devising a solution which is more adaptive and context-aware.

- Enable Adaptive Navigation in Unstructured and Chaotic Environments Linked to: The current systems rely heavily upon assumptions based upon structured road conditions that do not appear anywhere in Nigerian cities.

  Objective: We need a simulation that will mimic navigation by utilizing a "waypoint" based navigation system as a skeleton or outline that will assist an Artificial Neural Network based "reactive" navigation algorithm.

- Handling Unpredictable Traffic Behaviour from Heterogeneous Agents Linked to: Current models are facing difficulties in managing erratic behaviour from motorcycles, tricycles, pedestrians, and informal transport schemes, which are not adhering to existing traffic norms.

  Objective: Include chaotic agents that show erratic behaviour, such as sudden changes, stopping, informal lanes changes, etc., into the proposed Unity solution, which helps the ANN network to adapt to non-deterministic behaviour in real time.

- Incorporate and Adapt to Poor Road Conditions Including Roughness and Potholes Linked to: Most models do not consider defects such as roughness, potholes, and poor infrastructure affecting vehicle stability.

  Objective: To simulate how road roughness is accounted for as a normalizing input from 0 (smooth road) up to 1 (severe cases of potholes and rough asphalt roads), thereby being directly introduced as an input for an ANN that will adapt throttle, braking, and steering for stability while navigating typical Nigerian roads.

- Achieve Real-Time ANN-Driven Decision-Making with Optimized Backend Support Associated with: Many contemporary approaches to building and executing the ANN model do not enable real time and optimal scalability for complex simulation cycles. Moreover, the application environment has not been tested for very chaotic scenarios.

  Objective: Optimize the application scenario such that the developed ANN model can be linked with Python and implement the models for real-time processing, enabling multi-sensor processing for effective simulation (obstacles, way points, vehicle speed, roughness, and others). Similarly, the models can be linked with vehicle control parameters such that smooth control can be achieved, including acceleration between -1 and 1, brakes between 0 and 1, and steering between -1 and 1.

- Provide Quantitative Evaluation of Performance in Nigerian-Like Chaos Linked to: It has rarely been seen that the systems have the capabilities to effectively work within unstructured mixed traffic environments common in developing nations.

Objective: Simulate experiments that demonstrate the quantitative performance over a variety of traffic densities on roads with differing qualities to prove the effectiveness of the proposed model over traditional methods within a chaotic state.

3.1.4 BENEFITS OF THE PROPOSED SYSTEM

This hybrid autonomous simulation model offers significant advantages over traditional AV frameworks, specifically tailored for the realities of developing nations:

- **Adaptability to Unstructured Chaos:** In contrast to rigid systems dependent on lane markings, which cannot function without them, the innovative ANN can reactively maneuver past chaos elements like weaving motorcycles and pedestrians without being strictly bound by geometrical road conventions.
- **Resilience to Poor Infrastructure:** Through the use of a rough road parameter within the sensor matrix, the system regulates speed and braking according to the rough roads, which is not taken into account in western designs.
- **Ultra-Low Latency Inference:** Using the shared memory (mmap) interface for inference removes any network API latency. AI analysis of sensors' information and steering correction occur in less than one millisecond, guaranteeing performance at 60 frames per second.
- **Safe and Cost-Effective Testing:** Using the safe virtual sandbox makes it possible for the scientists to experiment on the autonomous system without any danger or costs, as no life is put in danger nor any material cost incurred.

3.2 SYSTEM REQUIREMENTS SPECIFICATION (SRS)

System requirements include the functionalities, processes, and qualities that need to be present in a system. The system requirements occur in two types: functional and non-functional.

3.2.1 Functional Requirements

These requirements refer to the functionalities or processes that the hybrid autonomous car simulation model must be able to perform. They include:

a. The system shall simulate a single hybrid autonomous vehicle in a virtual environment emulating the road conditions found in Nigerian cities.

b. The system shall use waypoint-based navigation in order to give high-level route guidance while allowing the vehicle to deviate from strict lane adherence when necessary.

c. The system shall feed real-time inputs to ANN, comprising distance to the next waypoint, distances to front, left, and right obstacles, normalized road roughness parameter (0-1 scale), and current vehicle speed.

d. The system shall provide the ANN with the capability of generating adaptive control signals, including throttle, brake, and steering. The throttle shall range from -1 for full reverse to +1 for full forward acceleration; the brake shall range from 0 for no braking to 1 for maximum braking force; and the steering shall range from -1 for full left to +1 for full right turn.

e. System: The system will replicate chaotic traffic motions such as the ones witnessed in the roads across Nigeria. These chaotic motions include the unpredictable movements of

motorcycles and tricycles, the unpredictable movement of pedestrians across the road, the fluid movement of vehicles from one lane to another, the go-slow traffic congestion, as well as the unpredictable stopping movement of vehicles.

f. This system shall utilize physics-based vehicle dynamcis, interactions with outdoors environments, and collision detection through the Unity simulation engine.

g. The system utilizes a **Shared Memory Bridge (mmap)** for inter-process communication (IPC) between the Unity simulation (C#) and the Neural Network (Python).

h. The system needs to record the instances of running the simulator and record the key performance parameters such as collision avoidance rates, deviations from the path while approaching waypoints, response time to obstacles in the road, and adaptability levels for various densities of traffic and quality of the road surface.

i. The system should provide for the visualization and analysis of simulation results through a means such as a simulation results dashboard that is made available either during or after simulation runs.

3.2.2 Non-Functional Requirements

Non-functional requirements refer to the qualities that the proposed system should have. They describe how the system should perform. Non-functional requirements for the hybrid autonomous car simulation model include:

a. The simulation has to be executable on common laptop hardware; it has to maintain a proper frame rate of 30+ FPS even in the case of moderate or heavy traffic density.

b. The inference engine, which is connected through a Shared Memory (mmap) bridge, has ultra-low latency characteristics, with a total decision cycle less than 1 ms. This is almost instantaneous synchronization between the Unity physics engine and the Python LSTM model. It will provide responsiveness to the simulation and keep a high simulation refresh rate above 60 FPS.

c. The system will ensure that simulation results are reproducible. It should be able to function with seeded randomness for traffic agents and initial ANN weights.

d. Simulated data, logs, and performance metrics will be stored and accessible only within the local development environment or exported to output files.

e. The error messages, simulation status, and controller status will be understandable.

f. The system architecture shall incorporate the design principle of modularity in all components, i.e., unity scene/scripts, the tensorflow-based LSTM model, etc. This will guarantee ease of maintenance, extension, and scalability while staying within the context of the final year project.

g. This simulation shall remain a virtual simulation with no need for any additional hardware, sensors, or even internet connectivity.

3.3 System Design and Methodology

The development process of the hybrid autonomous car simulation system entails the application of an integrated approach that combines both the Prototyping approach and the Spiral methodology in a life-cycle framework. In terms of system architecture, the OOAD approach is used to create the structure and behavior of various system components.

Components of Core Methodology

- Prototyping Methodology: Early baseline software prototypes were iteratively created and assessed in order to build the final prototype through a process where features were incrementally added ranging from simple waypoint following to sophisticated environment modeling including moving traffic agents, bumpy road surfaces, and the real-time neural network feedback loop.
- Spiral Methodology: This methodology is an iterative engine that improves risk management during project execution through four phases: planning of module objectives, risk analysis, possible risk of network overfitting or IPC latency, engineering of software increments, and evaluation using metrics such as collision frequency and navigation adherence.
- Separation of Concerns: The computational complexities within the system were separated by dividing its dependencies into independent modules such as perception (sensor input), decision (hybrid path generation), and actuation (physical movement).

Architectural Design Framework

- **Hybrid Control Architecture:** Utilizes a multi-layered design approach where a deterministic C# Expert System is used for high-level navigation through way-pointing and orientation; and the reactive, deep-learning network intervenes and overrides these trajectory plans for performing real-time obstacle avoidance.
- **Neural Network Design and Implementation:** The neural network architecture consists of an LSTM (Long Short Term Memory) recurrent neural network implemented using the open source software TensorFlow. The model processes continuous sequences of telemetry inputs to map highly non-linear driving functions onto continuous outputs from three different nodes, namely, throttle, brake and steering.
- **Simulation Pipeline and Infrastructure:** The simulation infrastructure is created within the Unity game engine due to its advanced physics handling, terrain modeling, and agent-tracking capabilities. High-speed IPC (Inter Process Communications) using a Shared memory bridge, specifically 'memory mapped' (mmap), allows instantaneous access to the data within the pipeline, thus removing any overhead associated with the laggy client-server requests based on the Web.
- Nigerian Driving Conditions: It is important that the core architecture learns disorder, not order; this requires integration of non-compliant driving styles and road surface conditions into the architecture in order to learn the ability to adaptively maneuver in the complex environment.

3.3.1 Justification for Methodology

Due to the explorative and risky aspect of simulating autonomous vehicles in an unstructured environment, the linear approach of modeling using software through the Waterfall methodology is unsuitable. For this reason, this research employs a combination of the Prototyping approach and Spiral methodology in the Object-Oriented Analysis and Design (OOAD) framework.

- **Prototyping for Iterative Development:** Prototyping enabled quick testing of the AI architecture through successive stages. The first prototype was used to implement basic physical movement as well as way-point following, while the second introduced stationary obstacles in the environment. Further prototypes incorporated moving obstacles (i.e., motorcycles and pedestrians), surface friction, and a real-time ANN-based steering system. This iterative process helped identify key requirements early, like realizing how the use of vehicle speed input to the ANN greatly reduced erratic steering output.
- **Spiral Model for Risk Management:** Training the AI-based neural network that can interpret road chaos requires certain operational and communication risks due to possible communication delays within the backend system. However, the spiral model was capable of addressing all these issues in each iteration by adopting a cyclic process of planning, risk identification, engineering, and evaluation. In cases where bugs were identified, such as the inability of the vehicle to reduce speed when encountering virtual road potholes, developers were able to debug and improve the neural network.
- **Component Modularity and System Independence:** Handling the difficulties involved in integrating the Unity C# simulation with the TensorFlow Python inference system needs strict modularity in handling the components. Compartments have divided the software

structure into three well-defined blocks: Expert System Trajectory Module, Sensor Fusion and Normalization Module, and LSTM Inference Module. Following the principle of the Spiral design model that advocates modularity, there is an efficient interface used for communication between modules via the use of Shared Memory (mmap). This ensures that changes to the neural backend, however many, do not affect the Unity physics rules or basic waypoint calculations.

- **Alignment with Academic and Regional Constraints:** This particular combination of frameworks addresses the academic and time restrictions imposed on an end-of-year undergraduate project. Through its focus on iteration cycles, the model guarantees that there will be a fully functional, testable prototype by the end of each cycle, thus avoiding any unforeseen problems that could halt the project. In terms of architectural relevance to this project, this is evidenced by how the developers were able to introduce the chaotic driving situation in Nigeria by gradually adding layers such as degraded road markings, hawkers along roadsides, and poorly maintained asphalt surfaces to the basic track design.

3.3.2 Method of Data Collection

- **Simulation-Generated Dataset:** The dataset used to train and test the Long Short-Term Memory (LSTM) model has been developed solely in the Unity simulation environment. This is due to the impracticality of measuring physical, real-life autonomous vehicle metrics in a real-world setting, specifically in complicated roads within Nigeria, within the time span of our undergraduate degree course.
- **Dataset Generation Process:** Telemetry data was recorded at regular time intervals while navigating through active loops. The dataset contains inputs, such as vehicle speed, local

road condition, and distances from different points and surrounding objects. The outputs correspond to actions taken by expert humans: throttle value between -1(reversed) and +1 (forward direction), braking force from 0 to 1, and steering from -1 (maximum left) to +1 (maximum right).

- **Scenario Diversity and Coverage:** To ensure high generalization capability, the data collection process was conducted using scenarios that comprised four distinct dynamics: varying levels of traffic intensity ranging from sparse fields to "go-slow" bottlenecks; varying road surface types, ranging from smooth asphalt (0.0) to rough potholes (1.0); varying scripts for traffic agents involving spontaneous lane changing and commercial bus stops; and varying path dynamics combining straight routes and aggressive turning scenarios.
- **Dataset Contents:** The resulting pipeline created an optimized dataset in CSV format comprising of 50,000 to 100,000 rows, with each row being a snapshot of one single timestep. Apart from sensor inputs and labels, each entry includes contextual information like scenario IDs, timestamp, existing traffic conditions, and surface condition of roads. This data set was carefully created to ensure that the neural network does not develop hazardous biases toward smoother surfaces and less traffic conditions.

3.4 Unified Modelling Language

UML Use Case, Activity, Class, and Sequence diagrams are utilized to blueprint the architectural interactions between the Unity frontend, Python backend, ANN controller, and traffic agents.

This structural documentation bridges the gap between conceptual design and technical implementation.

3.4.1 Use Case Diagram of the Proposed System

The major elements of the use case diagram have been divided into four major parts, which include the actors, which refer to entities or users of a system, the use cases, which refer to the processes or functions of a system, the associations, which refer to the relationships between the actors and the use cases, and the system boundary, which refers to the boundaries of a system.

With regard to the proposed system, there are three main actors that are identified. The actors are the Simulation Environment, the ANN Controller, and the Traffic Agents. The actor Simulation Environment represent the Unity engine that must control the simulation, physics, and communication. The ANN Controller actor shall represent the Python backend using the LSTM model that must receive the sensor information and provide the control actions. The actor Traffic Agents shall represent the various dynamic objects in the simulation, including motorcycles, tricycles, pedestrians, and vehicles that would cause the chaotic traffic.

The Simulation Environment has various associated use cases in its context, which are Initialize Simulation, Update Vehicle Physics, Process Sensor Readings, Send Data to Backend, Receive Control Commands, Apply Vehicle Controls, Detect Collisions, Update Waypoints, Render Visualization, and Log Performance Metrics. The ANN Controller has its various associated use cases, which are Receive Sensor Data, Process Neural Network Inference, Generate Control Outputs (Throttle, Brake, Steering), and Send Control Commands. The Traffic Agents actor also has various associated use cases such as Move Along Paths, Perform Lane Changes, Stop

Randomly, Cross Roads (in the case of pedestrians), and Weave Through Traffic (in the case of motorcycles).

There are also some use cases which define relationships through ‘include’ or ‘extend’ dependencies. For instance, in the Process Sensor Readings use case, Calculate Obstacle Distances and Measure Road Roughness are subsidiary to it. There is also a relationship from the Apply Vehicle Controls use case, which ‘extends’ to ‘Handle Emergency Braking,’ based on detected collision risk.

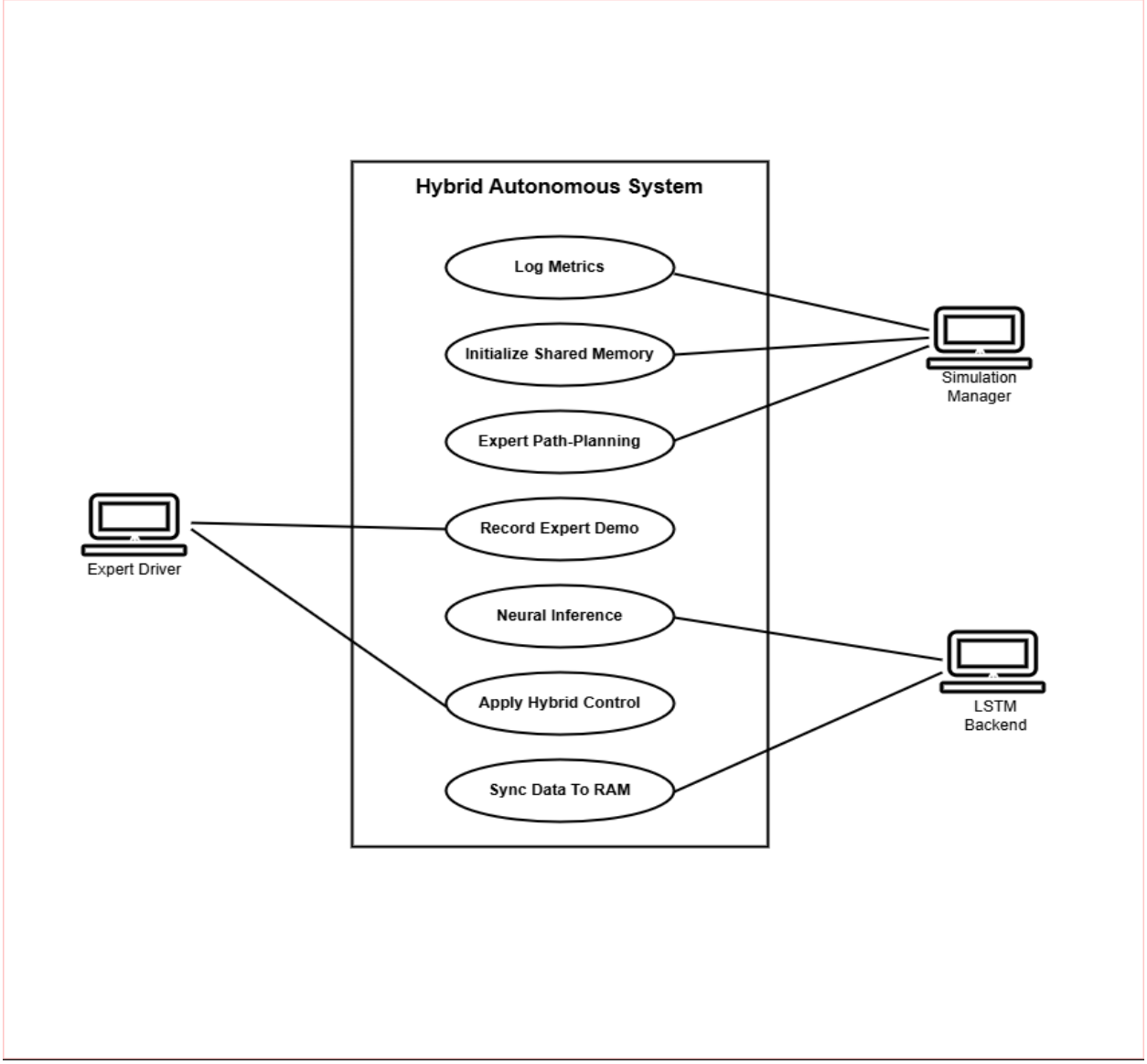

Figure 1.0 Use-Case Diagram of proposed system

Explanation of Figure 1.0:
The diagram illustrates the interactions between the system's three primary actors. The `Simulation Environment` *manages core physics, system initialization, and data logging. The* `ANN Controller` *processes sensor inputs to execute neural inference for throttle, brake, and steering commands. Finally, the* `Traffic Agents` *introduce environmental chaos by simulating unstructured, erratic driving behaviors typical of Nigerian roads, which the autonomous vehicle must reactively navigate.*

3.4.2 Activity Diagram of the Proposed System

The activity diagram shows the flow of the system's execution from initialization to performance evaluation. The execution begins simultaneously with the initialization of the Unity environment and the Python Inference Engine to establish the Shared Memory Bridge.

The execution loop of the system proceeds at high speed as follows:

1. Perception: The Unity environment updates the traffic agents and vehicle physics. A sensor array of 12 features is collected.
2. Decision Logic: The system checks the status of the Shared Memory Bridge.

- If the backend is unreachable, the Expert System provides deterministic path following.
- If the backend is reachable, the sensor array is mapped to RAM. The "nudge" calculation of the Reactive Steering and Throttle is computed by the LSTM ANN.

3. **Execution:** Here, the trajectory of the Expert System is combined with the swerve command of the ANN and applied to the vehicle physics.
4. **Evaluation:** The program checks if the destination has been met. If so, it logs the metrics to a JSON file and terminates the simulation. Otherwise, the loop will continue for the next frame.

Figure 2.0 Activity Diagram for the proposed systems

Explanation of Figure 2.0:

This figure details how the program executes itself. After the environment and the inter-process communication have been initialized, a high physics cycle begins, wherein the car writes its telemetry on twelve features to shared memory directly. This is polled by the Python backend, which employs LSTM inference to send reactive steering commands to the car. This happens in a loop until either a crash or destination event causes the simulation to end and log performance metrics.

3.4.3 Class Diagram of Proposed System

The class diagram demonstrates the structural design of the system. It describes the different components and how they interact in the Hybrid Autonomous System.

AutonomousVehicle (Main Class): The main entity representing the vehicle agent. The main attributes include position, velocity, orientation, and controlInputs. The main methods include UpdatePhysics() for movement, ApplyHybridControls() for controlling the vehicle, and CheckCollision() for collision detection.

SensorSystem: This is the system that performs sensor operations and data fusion. Instead of computing distances to objects, it uses a 12-feature vector that includes raycast distances, objectType (which identifies traffic and walls), and roadRoughness. The main method is MapTelemetryToSharedMemory(). It converts C# data to binary format for the Python backend.

ANNController (Inference Interface): This is the interface to the system and the LSTM Neural Network. It is not an ordinary MLP because it handles sequences. The main attributes are sharedMemoryBufferID and modelPath. The main methods are WriteTelemetry(), ReadInference(), and SyncHandshake() for effective communication via the Shared Memory Bridge.

TrafficAgent: This class refers to the characteristics of the "Dummy" entities that occupy the unstructured environment. The characteristics are agentType, which is Car, Motorcycle, Tricycle, or Pedestrian, and velocity. The agents will navigate a course to define the chaos that the autonomous vehicle must navigate.

ExpertSystem(PathPlanner): This class replaces the WaypointManager. This class controls the deterministic navigation. The controls are targetTrajectory and pathDeviation. This class will ensure that the vehicle stays on a general track geometry. This class will define a base steering angle that is refined by the ANN.

RoadSegment: This class refers to the environmental constraints. The characteristics are roughnessValue, which is a value between 0 and 1, and surfaceType. The GetSurfaceData() method is used to provide feedback to the SensorSystem to control suspension damper and throttle.

SimulationManager: This is the high-level manager that controls the simulation process. It controls the InitializeIPC() method to establish the memory bridge and the LogPerformanceMetrics() method to record the results of the Hybrid Brain's decision process in a JSON file.

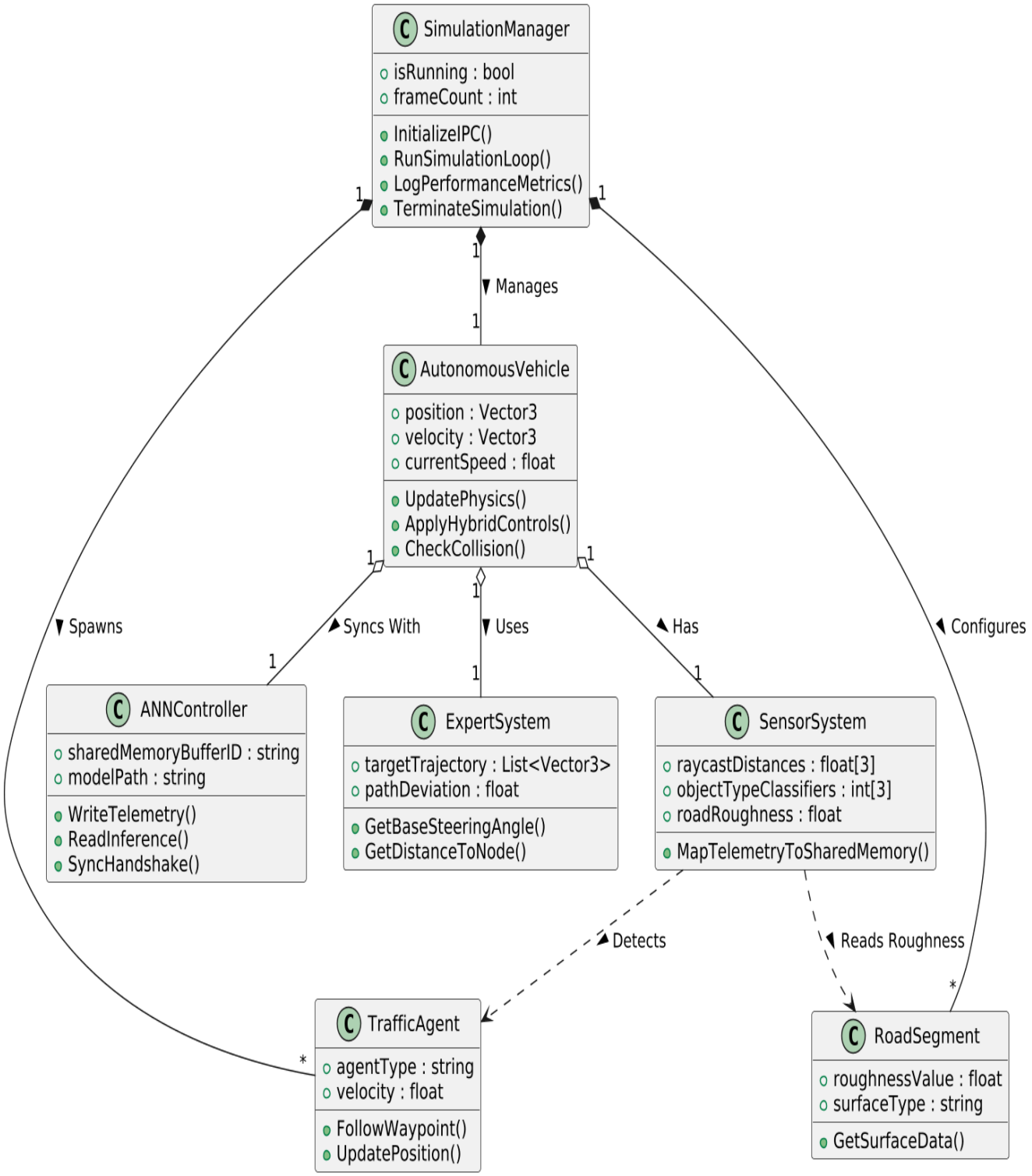

Figure 3.0 Class Diagram of the Proposed System

Explanation of Figure 3.0:

The diagram outlines the system's structural hierarchy, overseen by the central `SimulationManager`*. The core* `AutonomousVehicle` *class integrates a* `SensorSystem` *(12-feature perception) and an* `ExpertSystem` *(deterministic routing). Crucially, the vehicle synchronizes state with the* `ANNController` *via the Shared Memory Bridge, replacing traditional request-response bottlenecks with zero-latency reactive control. Auxiliary classes like* `TrafficAgents` *and* `RoadSegments` *supply scripted environmental chaos and surface roughness data. Ultimately, this structural pattern enforces a strict architectural separation of concerns between the Unity Physics environment, Deterministic Navigation, and the Neural Intelligence backend.*

3.4.4 Sequence Diagram of Proposed System

As can be noted in the UML sequence diagram below, the order of the message exchange between the system's essential elements or lifelines such as SimulationManager, AutonomousVehicle, SensorSystem, ExpertSystem, and ANNController (Python Backend) can be determined. The order of the sequence of the message exchange begins with the SimulationManager initializing the environment and creating the Traffic Agents and the Shared Memory Bridge. After the start of the simulation cycle:

1. **Sensing Phase:** In this phase, the AutonomousVehicle requests information to the SensorSystem, and the latter performs raycast operations and surface analysis. This results in a normalized vector with 12 features.

2. **Navigation Phase:** In this phase, the AutonomousVehicle requests information to the ExpertSystem, and the latter provides information on the basic trajectory and direction to follow the original path.

3. **Synchronization Phase:** In this phase, the AutonomousVehicle directly writes the combined information (sensor information and expert path information) to the Shared Memory block.

4. **Inference Phase:** In this phase, the ANNController, through polling RAM in a parallel thread, detects the information and performs inference on it through the LSTM network, writing the result to a shared buffer.

5. **Execution Phase:** During this phase, the vehicle retrieves the results from the RAM and combines them with the results from the ExpertSystem and finally applies the physics.

6. **Safety & Logging:** After the movement is completed, collision detection is carried out by the vehicle, and the SimulationManager logs the performance metrics into a JSON file.

This cycle continues until the AutonomousVehicle either collides or reaches the final destination as per the Expert System.

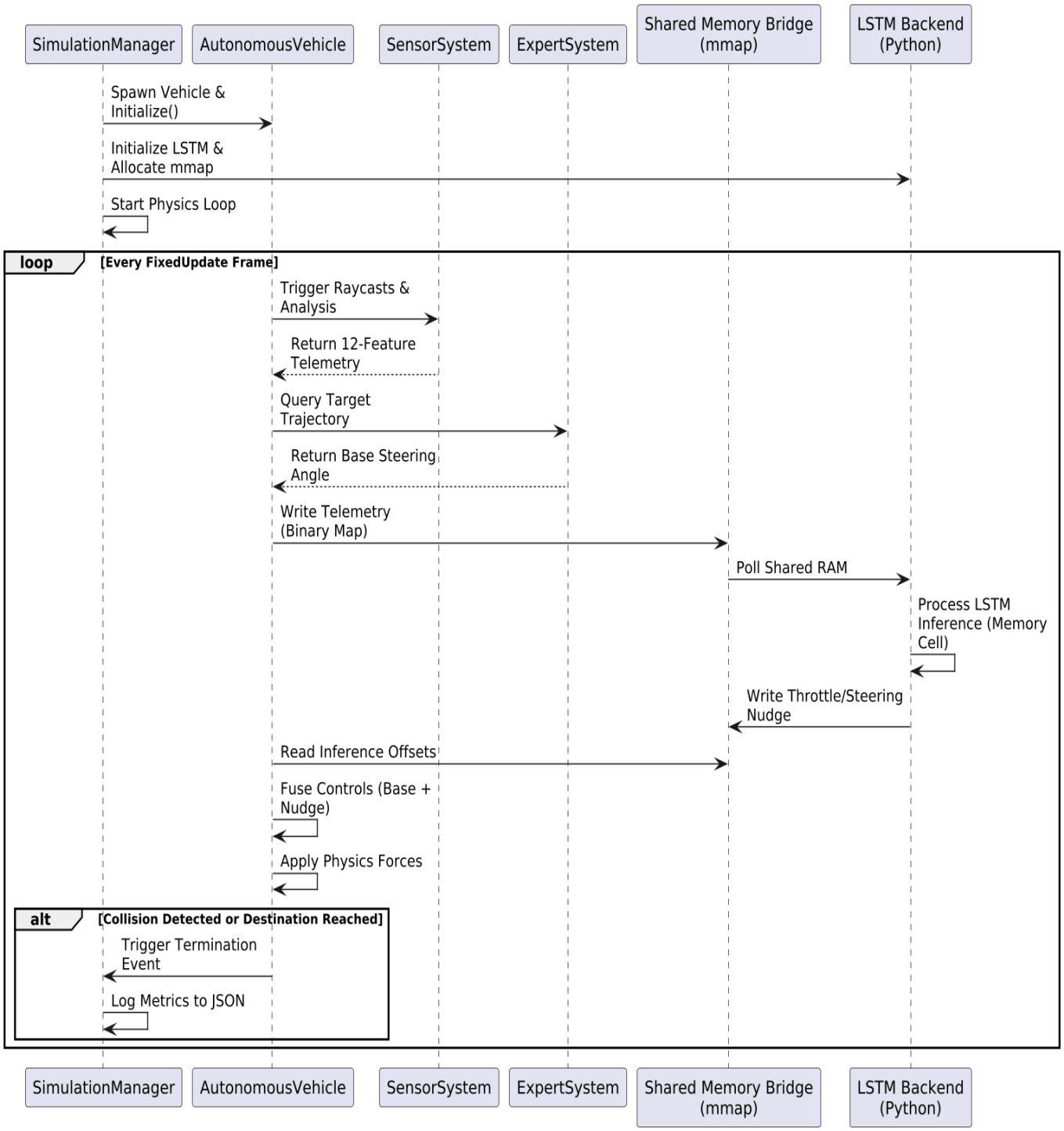

Figure 4.0 Sequence Diagram of the Proposed System

Explanation of Figure 4.0:
The diagram illustrates a complete simulation lifecycle, moving from initialization through execution and termination. During the active loop, the `AutonomousVehicle` *captures a 12-feature telemetry array and maps it directly to a shared RAM block, entirely bypassing traditional HTTP requests. The Python* `ANNController` *polls this memory, processes the neural inference, and writes sub-millisecond steering and throttle adjustments back to RAM. The vehicle fuses these AI "nudges" with the Expert System's baseline path logic before updating its physical movement, repeating this ultra-low latency cycle until collision or destination arrival.*

3.5 SYSTEM DESIGN

The design of the proposed system is intended to show how the hybrid autonomous car simulation model meets the requirements described above. The design of the proposed system includes the design of the simulation environment, the design of the neural network, and the design of the system integration.

3.5.1 Simulation Environment Design

The simulation environment design describes the virtual world where the autonomous vehicle will navigate.

Unity Scene Configuration The simulation environment is constructed from interconnected road segments designed to explicitly model the unstructured nature of Nigerian urban traffic. The core scene configurations include:

- **Road Network Layout:** A waypoint-driven route network encompassing straightaways, smooth curves, sharp intersections, and a deliberate absence of clear lane markings to accurately emulate local driving realities.
- **Road Surface Properties:** Dynamic surface profiles mapped to a normalized roughness scale (0.0 to 1.0), spanning from well-maintained asphalt to areas characterized by severe degradation and potholes.

- **Traffic Agent Spawn Points:** Configurable generation nodes for heterogeneous traffic agents (motorcycles, tricycles, pedestrians, and cars), allowing researchers to dynamically regulate traffic density during simulation initialization.

Environmental Parameters

The simulation accepts various inputs that define how complex or realistic a particular test scenario is configured to be, including:

- **Traffic Density Level –** an adjustable parameter with three levels of complexity (Low, Medium, High) defining how many traffic agents are active in the scene at a particular time.

- **Road Quality Profile –** a set of presets with three levels of complexity (Good, Fair, Poor) defining how many rough values are assigned to road segments.

- **Agent Behavior Settings –** adjustable levels of randomness defining how erratically traffic agents move, including how often they change lanes, how likely they are to stop suddenly, how much they weave when driving.

Figure 5.0: Unity Simulation Environment

3.5.2 Neural Network Architecture Design

The architecture of the neural network dictates how the design of the Long Short-Term Memory (LSTM) model is made to process the sensor telemetry data to generate corrective control signals. The LSTM model is designed based on its ability to retain a hidden state over a series of time steps, which is essential to achieve a smooth steering action while anticipating traffic movement.

Input Layer Configuration (12 Features)

The input layer receives a normalized 12-dimensional vector at each time step. The vector is composed of the following features:

- Front, Left, Right Obstacle Distances (3): Normalized raycast distances (range 0.0 to 1.0).
- Obstacle Type Classifiers (3): Discrete features for front_type, left_type, and right_type (e.g., 1.0 for OpponentCar, 0.0 for static environment).
- Kinematic Data (2): Speed of the vehicle and "is_stuck" flag.
- Path Planning Data (3): Target trajectory offset, angle to next node, and rear clearance.
- Environmental Data (1): Normalized road roughness.

Hidden Layer Architecture (Recurrent Design) In order to deal with the temporal complexity of the problem, the following architecture is used:

- LSTM Layer: This layer has 64 units. This layer is designed to consider the last 10 to 20 frames of the sequence. This way, the AI will be able to comprehend the rate of the surrounding vehicles rather than their position.
- Dense (Hidden) Layer: This is a fully connected hidden layer with 32 units and ReLU activation to perform abstraction on the temporal features learned in the previous layer.
- Dropout Layer: This is a dropout layer with 20% dropout. This is to prevent the model from overfitting on the training data for specific sections of the track.

Output Layer Configuration The output layer has three neurons. Each of these is associated with the "neural nudge" phenomenon on the vehicle actuators. They are as follows:

- Throttle Output (-1.0 to +1.0): This has a tanh activation function. Negative values indicate engine braking or reverse action.

- Brake Output (0.0 to 1.0): This has a sigmoid activation function. The friction braking is calculated.
- Steering Offset (-1.0 to +1.0): This has a tanh activation function. This is a "correctional delta" to the Expert System's basic steering command to assist obstacle avoidance.

Training Configuration

- **Loss Function:** The loss function used is Mean Squared Error (MSE). The predictions made by the model are compared to a dataset of 91k+ samples of expert human demonstrations.
- **Optimizer:** The optimizer used is Adam optimizer with a learning rate of 0.001 for convergence.
- **Data Sequence:** The sliding window technique is used, which takes a sequence of telemetry data as an input, as opposed to a single frame of telemetry data.

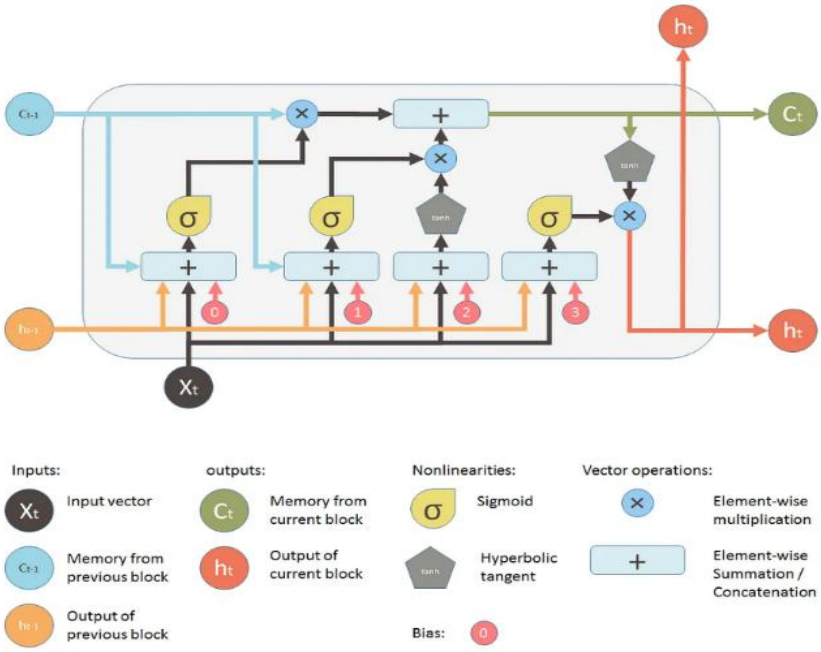

Figure 6.0: LSTM Neural Network Architecture

3.5.3 Communication Protocol Design

The communication protocol design deals with the high-speed data flow between the Unity simulation front-end (C#) and the Python TensorFlow back-end (Inference Engine).

Shared Memory Bridge (Memory-Mapped Files) To attain the real-time performance requirement, the proposed system eliminates the network overhead (HTTP/REST) and uses the **Shared Memory Bridge**. This is done by mapping a specific block of the system's RAM (Tagname: "Local\UnityCarAI") that can be shared between the two processes.

The Synchronization Cycle: During every physics step (Fixed Update), the following low-latency cycle occurs:

1. **Telemetry Write (Unity → Shared RAM):** A binary packet of 68 bytes is sent from Unity containing 12 normalized sensor features and 3 user input labels.

2. **State Flag Update:** The synchronization integer (flag) is set to 1 by the program, indicating that the data is "fresh."

3. **Inference (Python):** The Python program reads the RAM block and identifies the updated flag. It uses the 12 features to perform the inference using the LSTM model and calculates the steering offset.

4. **Command Write (Python → Shared RAM):** The program writes the throttle, steering, and brake values to the RAM block and resets the flag to 0.

Data Structure (Binary Format): Instead of transferring bulky JSON data structures, data is sent in a continuous stream to reduce parsing time:

- Input Block (12 Floats): [front, left, right, speed, wp_dist, roughness, wp_angle,rear_dist, is_stuck, front_type, left_type, right_type]
- Output Block (3 Floats): [throttle, steering, brake]

Error Handling

If the Shared Memory block fails to initialize or the Python backend is not active, the Unity application reverts to the Expert System (Deterministic logic) to ensure that the car continues on its path without stalling the simulation.

Latency and Performance

With memory mapping, our latency is minimized to sub-millisecond times. This allows us to run our simulation at high frame rates (60+ FPS) without any 'jitter' associated with API calls over a network.

3.5.4 Data Storage and Logging Design

The system functions without a traditional relational database system but uses a structured file-based system to log high-fidelity simulation telemetry and performance metrics. This is used as a data source for offline imitation learning.

Training Data Architecture

Data from telemetric data streams during manual or expert teacher simulation sessions is stored in real time in a Comma-Separated Values (CSV) file format. This tabular data is the "experience" of the system and is related to the responses of the human expert as a controller.

Training Data Structure (15-Column Feature/Label Set)

Feature Group	Column Name	Sample Value	Description
Sensors	`front`	0.82	Normalized distance (Forward)
	`left`	0.45	Normalized distance (Left)
	`right`	0.91	Normalized distance (Right)
	`speed`	0.38	Normalized vehicle velocity

	waypoint_dist	0.65	Distance to target trajectory
	roughness	0.25	Road surface intensity
	waypoint_angle	0.12	Angle to next path node
	rear_dist	1.00	Distance to rear obstacles
	is_stuck	0.00	Binary flag (1 if immobile)
Classifiers	front_type	1.00	Object ID (1.0 = Opponent Car)
	left_type	0.00	Object ID (0.0 = Ground/Wall)
	right_type	1.00	Object ID (1.0 = Opponent Car)
Labels	steering	-0.12	Expert steering command
	throttle	0.45	Expert acceleration command
	brake	0.00	Expert braking command

Performance Metrics Logs

To measure the performance of the Hybrid Brain, the system creates a summary log in JSON format after each autonomous run. The logs focus on the interaction of the Expert System's path-following capabilities with the ANN's reactive swerving capabilities.

Sample Log File: `run_metrics_001.json`

JSON

```
{
```

```
  "run_id": "001",
  "timestamp": "2025-01-26T14:30:00",
  "config": {
    "traffic_scatter_density": "High",
    "road_roughness_bias": "0.7",
    "control_mode": "Hybrid_Expert_ANN"
  },
  "results": {
    "collision_count": 0,
    "ann_swerve_interventions": 12,
    "expert_path_accuracy_pct": 94.5,
    "avg_inference_latency_ms": 0.8,
    "total_duration_s": 58.7,
    "status": "Success"
  }
}
```

These allow for comparative analysis to ascertain if the ANN is effectively "nudging" the Expert System off the road without compromising the stability of the overall system of navigation.

Deployment Configuration

- Unity Frontend: This will be a standalone executable that runs on the development machine and contains the simulation window and processing for user input for controlling the run (start/stop).

- Python Backend: This process will utilize the Python mmap module for creating a high-speed binary interface into the simulation environment. This process will function by continuously polling the system RAM for fresh data and performing the neural inference against the 12-feature sensor array. The response will then be written back into the shared memory buffer in microseconds.

- Model Files: These will be contained within the models/ directory and contain the trained LSTM model within the file named "trained_model.h5".

This configuration keeps the simulation environment and the intelligent control logic separate and distinct in order to allow for parallel development and maintenance.

3.6 Summary

Chapter Three described the analysis and design of the hybrid autonomous vehicle simulation system. In evaluating the limitations of the existing models, the basic objectives of the proposed system were identified: adapting to unstructured environments, responding to unpredictable traffic, evaluating road conditions, and real-time decision-making through the use of an Artificial Neural

Network. System requirements were identified, which justified the need for a flexible iterative spiral prototyping methodology. In addition, the interactions of the system were modeled through the use of standard UML diagrams, such as the use case, activity, class, and sequence diagrams. Finally, the conceptual design was converted into technical specifications, which include the Unity environment, the recurrent LSTM architecture, and the high-speed shared memory integration, which provides the basic groundwork for the implementation of the system in Chapter Four.

CHAPTER FOUR

SYSTEM IMPLEMENTATION

4.0 INTRODUCTION

This chapter discusses the building and implementation of the hybrid autonomous vehicle simulation model. It describes the building tools, the generation of the expert demonstration dataset, the training of the Long Short-Term Memory (LSTM) network, and the inclusion of the critical Inter-Process Communication (IPC) modules. The implementation directly builds on the architectural specifications discussed in Chapter Three, so the system can successfully navigate the unpredictable, unstructured environment of the Nigerian roads.

4.1 DEVELOPMENT ENVIRONMENT AND TOOLS

The system was implemented based on a decoupled architecture where the physical simulation is decoupled from the neural inference engine, with the former achieving real-time synchronization through a high-performance memory bridge instead of network protocols.

4.1.1 Programming Languages & Libraries

- **Unity Game Engine (C#):** Manages the simulation frontend. This includes the vehicle physics, rendering of the environment, the dynamic generation of "road roughness," and the deterministic logic for the Expert System path-planning algorithm as well as the chaotic traffic agents, e.g., erratic motorcycle or pedestrian behaviors.
- **Python:** Acts as the main programming language for the backend inference engine, processing the 12-feature telemetry arrays and computing the real-time control adjustments.

- **TensorFlow & Keras:** Utilized to build, train, and execute the trained recurrent LSTM neural network. This suite of tools permits the model to recognize the intricate relationships between the sequence of sensor input features and the required vehicle control outputs.
- **Shared Memory (mmap):** Acts as the high-speed data communication bridge between the Unity C# frontend and the Python backend. This method of Inter-Process Communication replaces the traditional HTTP/JSON pipelines to enable the rapid exchange of binary data with near-zero latency.
- **CSV (Comma-Separated Values):** Used to store the expert-generated demonstration sets, which compile the 91,000+ sequential samples required to train the recurrent network.

4.1.2 Development Platform & Hardware

- **Integrated Development Environments (IDEs):** Visual Studio was employed to develop the C# code and Unity script, while Visual Studio Code was used to build the Python backend code and TensorFlow model.
- **Hardware Specifications:** The hybrid system was created, trained, and tested on general consumer-grade laptop hardware. The effective implementation of the mmap bridge and the optimized LSTM architecture confirms the ability to support a high refresh rate in the simulation (60+ FPS) and ultra-low inference latency (< 1 ms), without the need to use industrial-grade clusters.

Dataset Generation Instead of relying on orderly, Western-centric public datasets, the training dataset was generated entirely within the Unity simulation environment. This approach overcame the lack of real-world Nigerian driving data while directly fulfilling the project's goal: training the LSTM specifically for unstructured urban chaos.

Rather than using hardcoded scripts, the data was collected via **Imitation Learning**. By recording human expert demonstrations across highly varied scenarios, the system captured continuous temporal sequences of the vehicle's 12-feature sensor readings alongside the expert's precise control outputs. These sequential snapshots provided the LSTM with the exact historical context and momentum needed to learn adaptive, human-like driving decisions.

Dataset Contents The final dataset consists of over 91,000 individual timestep samples, stored within a CSV file named training_data.csv. Each row of the dataset is a single timestep of the vehicle's state during the simulation process, with 15 columns consisting of 12 input features and 3 output targets:

Column	**Type**	**Description**
`front_obs`	Float (0–1)	Normalised distance to the nearest forward obstacle.
`left_obs`	Float (0–1)	Normalised distance to the nearest obstacle on the left.
`right_obs`	Float (0–1)	Normalised distance to the nearest obstacle on the right.
`front_type`	Float (Discrete)	Object classifier for front obstacle (e.g., Traffic Agent vs. Static Wall).
`left_type`	Float (Discrete)	Object classifier for left obstacle.
`right_type`	Float (Discrete)	Object classifier for right obstacle.
`speed`	Float (0–1)	Normalised current vehicle speed.
`is_stuck`	Float (Binary)	Flag indicating if the vehicle's momentum is halted.
`traj_offset`	Float (0–1)	Distance deviation from the Expert System's base trajectory.

angle_node	Float (−1 to 1)	Normalised angle to the next path node.
rear_clear	Float (0–1)	Normalised distance to trailing traffic agents.
roughness	Float (0–1)	Normalised road surface roughness at vehicle location.
throttle	Float (−1 to 1)	Control output for forward/reverse acceleration.
brake	Float (0–1)	Control output for braking intensity.
steering	Float (−1 to 1)	Control output representing the correctional delta (nudge) for steering direction.

The twelve columns of inputs represent the expanded sensor fusion and temporal inputs to the LSTM ANN model. The three output columns represent the corresponding control signals provided by the human expert during the data collection runs.

Scenario Diversity and Coverage To guarantee model generalization and prevent bias toward any specific condition, the training data was systematically balanced across four distinct dimensions:

- **Traffic Density:** Ranged from sparse, low-interaction environments to highly congested, stop-and-go traffic.
- **Road Quality:** Spanned the entire normalized scale from 0.0 (smooth asphalt) to 1.0 (severe potholes and degradation).
- **Agent Behaviour:** Incorporated chaotic elements such as weaving motorcycles, sudden vehicle lane changes, and unpredictable pedestrian crossings.
- **Route Geometry:** Covered straight paths, sharp curves, and complex intersections to capture a complete, 360-degree spectrum of steering inputs.

Preprocessing Steps Prior to model training, the raw dataset underwent the following preparation:

- **Cleaning:** Removed duplicates and corrupted frames (such as collision events or scene resets) to eliminate training noise.
- **Temporal Sequencing (Sliding Window):** Grouped the data into continuous sequences of 10 to 20 timesteps to provide the necessary historical context for the LSTM.
- **Normalization:** Skipped redundant scaling by relying on Unity's pre-normalized outputs. The 12 input features were verified to be strictly within the 0.0 to 1.0 (or binary) range, and the three control outputs remained within their defined actuation bounds.
- **Stratified Split:** Divided the sequence data into 70% training (≈63,000 samples), 15% validation, and 15% test sets. Stratification ensured that traffic density and road quality profiles were proportionally distributed across all three splits.

4.3 MODEL ARCHITECTURE

The architecture of the ANN model is as follows: The ANN model used is a Long Short-Term Memory Recurrent Neural Network architecture. It is built using the TensorFlow Keras API. The model has a good balance of the awareness of the trajectory and momentum of the traffic over time and the ultra-fast inference latencies necessary to compute the control predictions at high frequency rates of 60+ Hz in the simulation loop. The entire model is contained in the file neural_network.py.

4.3.1 Input Layer Configuration

As opposed to a standard feed-forward network that is only able to take a single frame as input, the LSTM's input layer is able to take a tensor representing a number of frames. The tensor has a dimensionality of a triplet of the form (batch_size, timesteps, features), where the features dimension consists of the 12 normalized floating-point values representing the entire sensor data and expert system trajectory of the autonomous vehicle.

The 12-dimensional feature vector consisting of the raycasts, object classifiers, kinematics, path data, etc., is strictly enforced throughout the entire data-logging process as implemented in the DataLogger.cs file, the binary schema mapping process as implemented in the Shared Memory bridge, and the feature vector construction process as implemented in the Python backend. This ensures that the feature vector sequence provided to the network during the inference process via the RAM memory is the same as the feature vector sequences provided to the network during the Imitation Learning training process.

4.3.2 Hidden Layer Architecture (Recurrent Design)

The network utilizes a recurrent structure to deal with the temporal complexity of the driving scenario in an unstructured environment, as opposed to a simple dense funnel:

LSTM Layer: The hidden layer consists of 64 LSTM cells. It maintains a "hidden cell state" over time, allowing the network to learn the velocity and "intent" of the surrounding traffic agents, as opposed to the static and instantaneous position of the agents. Mathematically, at any time step t, the LSTM cell processes the current 12-feature telemetry vector x_t and the hidden state from the previous frame h_{t-1}. The network controls the flow of this information through a series of specific gates:

The Forget Gate (f_t): This gate decides which of the historical telemetry data is no longer relevant and hence to be discarded. The output of this gate varies between 0 and 1 for any cell state element, where 0 implies "completely forget" and 1 implies "completely keep":

$$f_t = \sigma(W\_f \cdot [h_{t-1}, x_t] + b\_f)$$

Input Gate (i_t) and Candidate Memory ($\tilde{C}_t$): These two components work together to determine the new information to be stored in memory. The input gate determines what information is used for updating, whereas candidate memory produces a vector of new candidate values:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W\_C \cdot [h_{t-1}, x_t] + b\_C)$$

Cell State Update (C_t): This is where the network's long-term memory is updated. The irrelevant information from previous time steps is scaled out by the forget gate, whereas the new candidate values are scaled in by the input gate:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Output Gate (o_t) and Hidden State (h_t): This is where the output is determined for passing on to the next layer. The output is determined by a filter of the new cell state:

$$o_t = \sigma(W\_o \cdot [h_{t-1}, x_t] + b\_o)$$

$$h_t = o_t \odot \tanh(C_t)$$

Note: σ is the sigmoid activation function, W is each of the corresponding weight matrices, b is each of the corresponding bias vectors, and ⊙ is element-wise multiplication.

Dense Layer: The output of the LSTM layer is used as the input for a Dense layer with 32 neurons and a ReLU activation function. This layer performs abstraction on the features learned in the previous LSTM layer.

Dropout Layer: The Dropout layer is used in this model with a dropout rate of 20%. This layer is used as a regularizer to prevent the network from overfitting on the specific geometries of the tracks used in training the network. It helps the network generalize better in other scenarios in Nigeria.

4.3.3 Output Layer Configuration

The output layer consists of three neurons, with each neuron corresponding to a control actuator for the vehicle. Instead of linear activation functions with clamping, special activation functions are used to naturally constrain the output within physically meaningful ranges:

Throttle Output: The tanh activation function is used to generate a continuous output between -1.0 and +1.0, corresponding to maximum reverse/engine braking and maximum forward acceleration.

Brake Output: The sigmoid activation function is used to generate an output between 0.0 and 1.0, corresponding to friction braking intensity.

Steering Output: The tanh activation function is used to generate an output between -1.0 and +1.0, corresponding to correctional steering delta. This "neural nudge" is then combined with the Expert System's base trajectory logic within Unity.

4.3.4 Loss Function and Optimiser

The model is compiled with the Mean Squared Error as the loss function and Adam Optimiser. The reason for choosing the Mean Squared Error as the loss function is the fact that it heavily penalises any large error in the predictions made by the model. In the case of an autonomous vehicle, a large error in the steering or throttle of the vehicle can be exponentially more harmful than a smaller error. The Adam Optimiser has been chosen as the optimiser because of the adaptive learning rate, which is useful in efficiently traversing the loss surface, especially when making predictions on continuous values.

4.3.5 Temporal Sequencing and Buffer Management

Considering the fact that the Unity simulation environment has already pre-normalized the 12 sensor inputs into the range of 0-1 before the simulation environment sent the data over the Shared Memory bridge, the need to scale the inputs using the Scikit-Learn Standard Scaler has been removed in order to minimize the inference latency.

Considering the inference aspect, the current data preprocessing has been focused on Temporal Sequencing. As the Python backend reads the 12-feature array in real-time from the mmap buffer, the read is appended into a data queue using the collections.deque data structure in Python. Once the data queue has the required length, such as the last 15 frames, the data is sent into the LSTM network in the required 3D tensor shape of (1, timesteps, 12). As the simulation progresses, the

oldest frame is removed from the data queue, and the newest frame is added, such that the neural network is always running the recent history of the driving history of the vehicle without any interruptions.

4.4 MODEL TRAINING

The training is performed through the train_model.py script, which invokes the train_from_csv() function to manage the machine learning process. The complete training process involves the following organized steps:

Stage 1 — Data Augmentation: Prior to the training process, the original training_data.csv file is subjected to an augmentation script (argument_data.py) to counter a prevalent issue in the driving data: straight-line and right-side biased driving examples tend to be more abundant than left-turning examples because of the inherent asymmetry in human driving patterns. The augmentation step generates a copy of each row by negating the steering and angle_node values and reversing the left/right sensor pairings (i.e., left_obs to right_obs and left_type to right_type). These copied rows are appended to the original rows to form training_data_balanced.csv, effectively doubling the size of the original dataset and ensuring that both left and right steering examples are represented equally.

Stage 2 — Feature and Label Extraction: The balanced CSV file is read into a Pandas DataFrame. The 12 input features (raycast distances, object classifiers, kinematics, and path) are extracted as the feature matrix X. The 3 output features (throttle, brake, and steering) are extracted as the ground truth label matrix y.

Stage 3 — Temporal Sequence Generation (Sliding Window): As the input features are already normalized to a range between 0 and 1 by the Unity environment, scaling operations such as StandardScaler are skipped to further optimize real-time inference time. Instead, the 2D feature matrix X is converted into a 3D tensor using a sliding window mechanism. This mechanism groups the input data into sequences and enables the LSTM to learn from the progression and momentum of the car over the last 10 to 20 frames instead of processing individual frames in isolation.

Stage 4 — Train/Test Split: The sequential data is split into a training set and a test set. Special care is taken during this stage to ensure that the sequences are not broken and frames within a sequence are not shuffled. The test set is not used during the training process and is reserved only for evaluation purposes.

Stage 5 — Training: In the fifth stage, the LSTM network is trained for 50 epochs with a batch size of 32 sequences. The test split of the data is passed to the network as the validation dataset. This enables the validation loss to be monitored alongside the training loss at the end of every epoch. This is important to avoid overfitting, where the loss on the training data goes down, but the loss on the validation data goes up. The LSTM network is memorizing the specific training tracks rather than learning the essential driving behaviors.

Stage 6 — Model Serialisation & IPC Initialization: In the sixth stage, the trained model is serialized to the file trained_model.h5. The Python Inference Engine reads the file at the start of the program to begin polling the Shared Memory (mmap) bridge. If the file is unavailable—before the first training run—there is a graceful fall-back to a safe, deterministic default output. In this

case, the Unity vehicle will be able to continue moving forward without crashing the simulation by setting the throttle to 0.3, the brake to 0.0, and the steering to 0.0.

4.5 SYSTEM MODULES

This section outlines the decoupled architecture linking the Unity C# physics frontend and the Python neural backend.

- **4.5.1 Python Inference & Shared Memory Bridge (`main.py`, `mmap_bridge.py`):** Replaces sluggish HTTP requests with a Memory Mapped File (mmap), achieving sub-millisecond data exchange for real-time LSTM predictions.
- **4.5.2 Sensor & Telemetry Module (`CarSensors.cs`):** Utilizes a 180-degree raycast fan to calculate obstacle distances and object types, alongside a downward raycast to classify road roughness on a 0.0 to 1.0 scale.
- **4.5.3 Data Logging Module (`DataLogger.cs`):** Automatically writes the 12-feature sequential training data directly to a CSV buffer during manual driving demonstrations.
- **4.5.4 AI Client Module (`AIClient.cs`):** Synchronizes telemetry with the shared RAM and applies anti-stall logic (clamping weak throttle outputs to 0.3) to overcome simulated physical road resistance.
- **4.5.5 Vehicle Controller (`PlayerCarController.cs`):** Manages wheel collider physics and merges the deterministic expert trajectory with the LSTM's reactive steering "nudge".
- **4.5.6 Expert Path-Planning (`ExpertNavigator.cs`):** Provides the baseline deterministic waypoint navigation and calculates the vehicle's angle and trajectory offset.

- **4.5.7 Traffic Agents (`OpponentCar.cs`):** Governs the dynamic obstacles using distance-based braking and basic waypoint tracking to simulate active, moving traffic conditions.

4.6 TESTING AND EVALUATION

4.6.1 Evaluation Metrics

In order to quantitatively measure the performance of the trained Long Short-Term Memory (LSTM), a series of metrics was calculated over a test set, which comprised 20% of a balanced sequential dataset used to train the model.

- **Mean Absolute Error (MAE):** This metric calculates the average value of the difference in the model's predicted control values and the ground truth label values. It is the primary metric tracked during the training process through TensorFlow's metrics=['mae'] argument. It is tracked independently on each of the three nodes to see if there is a problem with the model's ability to accurately predict a specific control value. Because the model predicts normalized values between -1.0 and 1.0, the MAE can be multiplied by the vehicle's physical actuator limits (such as the 35 degree maximum steering angle) to determine the exact real-world physical deviation of the AI from the expert trajectory.
- **Mean Squared Error (MSE):** This metric is the primary loss function used to train the model and a secondary metric to evaluate the model's performance. MSE is a more sensitive measure because it squares the differences, which means a low MSE on the test set indicates the model is not occasionally making large errors in controlling the car, even if the average error is low.

- **Validation Loss Curve:** This is monitored over all 50 training epochs. If the validation loss is seen to be tracking the training loss but stabilizes before the last epoch, it is an indication that the model has generalized rather than overfitting. A large and growing gap between the training loss and the validation loss would suggest overfitting, indicating the recurrent network is memorizing the sequence of the training tracks.

4.6.2 Module and Integration Testing To ensure each component of the decoupled hybrid system functioned correctly in isolation and in combination with the others, the following tests were conducted.

Unit Tests:

- **Sensor Module (`CarSensors.cs`):** The improved raycast sensor was validated to correctly split the hits into the front, left, and right zones, as well as correctly identify the objects hit (static wall objects versus dynamic OpponentCar objects). The roughness sensor was also validated to ensure it correctly returns 1.0, 0.5, and 0.0 for Pothole/Bump, Rumble, and smooth road surfaces, respectively.
- **Data Logger (`DataLogger.cs`):** The data logging pipeline was validated to ensure it only becomes active in Manual drive mode. The data was inspected to verify the presence of all 12 input features and 3 control fields, correctly formatted as continuous values, and was correctly dumped to the CSV buffer without any dropped frames.
- **Shared Memory Bridge (`mmap_bridge.py`):** This module has replaced the old API endpoint tests. This module was tested independently to ensure that the memory-mapped file was successfully allocated in the system RAM. The binary-level packing and

unpacking of data types also ensured that they matched the exact length expectations of both the C# client and Python backend.

- **Predictor (`predictor.py`):** The fallback functionality was tested by intentionally deleting the trained_model.h5 file. This ensured that the backend successfully initialized without crashing, returning safe default values (e.g., throttle = 0.3, steering = 0.0). The sequence reshaper was also tested to ensure that it correctly stacked the incoming 1D telemetry arrays into a 3D tensor shape of (1, timesteps, 12) to feed into the LSTM.

Integration Tests:

- **End-to-End Inference Pipeline:** The entire autonomous control loop was tested by running the Unity simulation in Autonomous drive mode with the Python backend enabled. The AIClient.cs script would write the 12-feature telemetry to RAM at every physics tick, the Python script would read it, run it through the LSTM, and write the results back. The round-trip IPC latency was also measured to ensure it stayed well under 1ms at all times, a huge step up from the previous 50ms HTTP pipeline.
- **Data Collection to Training Pipeline:** The entire data collection to training loop was run end-to-end. A human would drive the car to collect the data, argument_data.py would balance the data, and then train_model.py would run the training process. The new model was then run in the live simulation to ensure it produced valid predictions.
- **Temporal Buffer Consistency:** A test was run to ensure consistency between the sliding window deque buffer used by the live inference script and the temporal sequence structure produced by the training script. This was to ensure the LSTM was using the correct sequence of the vehicle's momentum.

4.6.3 Simulation Scenario Testing

In lieu of the traditional user interface test section, the system was validated under a variety of structured simulation scenarios to test the system's response to the specific road conditions that characterized the project's scope of work. The scenarios were run with the IPC bridge active and the vehicle in Autonomous mode with the Hybrid Expert System.

- **Open Road Scenario:** The vehicle was placed on a clear road with no traffic agents and smooth road surfaces. The model was expected to make constant progress on the expert trajectory without any unnecessary braking. The vehicle was able to traverse the entire path without stalling.
- **High-Density Traffic Scenario:** The opponent car agents were populated at maximum density, resulting in frequent proximity events. The LSTM's use of temporal memory was expected to help the model recognize the momentum of the agents. The vehicle was able to make appropriate decelerations when following slow-moving agents and accelerated appropriately when the path was clear, thus avoiding "jittery" braking.
- **Rough Road Scenario:** Road segments with roughness values close to 1.0 were included on the route. The vehicle slowed substantially on the potholed road segments compared to the speed on the smooth road segments, thus verifying the modulation effect of the roughness feature on the throttle prediction.
- **Mixed Scenario:** A mixed scenario was created to include dense traffic, rough road segments, and waypoint turns. In the mixed scenario, the model's performance was assessed. The vehicle successfully passed through the scenario without any collisions. The

hybrid control was successfully able to combine the base steering control provided by the Expert System with the LSTM's obstacle avoidance "nudge."

4.7 RESULTS PRESENTATION AND ANALYSIS

4.7.1 Training Results

When the training pipeline was completed, the Long Short-Term Memory (LSTM) architecture was observed to consistently converge throughout the 50 epochs of the training process. The training set loss and validation set loss were observed to consistently decrease from the first epoch and remained steady in the latter stages with no appreciable divergence between the training set and validation set losses, thus showing that the recurrent architecture was able to generalize well to unseen data without overfitting to the specific geometries of the training tracks. The Mean Absolute Error (MAE) values recorded on the test set at the end of the training process confirmed that the predictions of the recurrent architecture for all three control values, including the throttle, brake, and steering correctional delta, remained within an acceptable margin of the actual labels provided by the human expert in the Imitation Learning driving sessions.

To contextualize this performance mathematically, the final Steering MAE achieved on the test set was 0.0198. The simulated vehicle utilizes a physical steering rack with a maximum turn angle of 35 degrees. By translating the normalized error to the physical simulation bounds ($0.0198 \times 35°$), the calculated real-world deviation is approximately 0.69°. This indicates that the LSTM's reactive steering decisions deviate from the perfect geometric expert trajectory by less than a single physical degree per frame, ensuring human-level operational smoothness and high-confidence lane keeping even in unstructured road conditions.

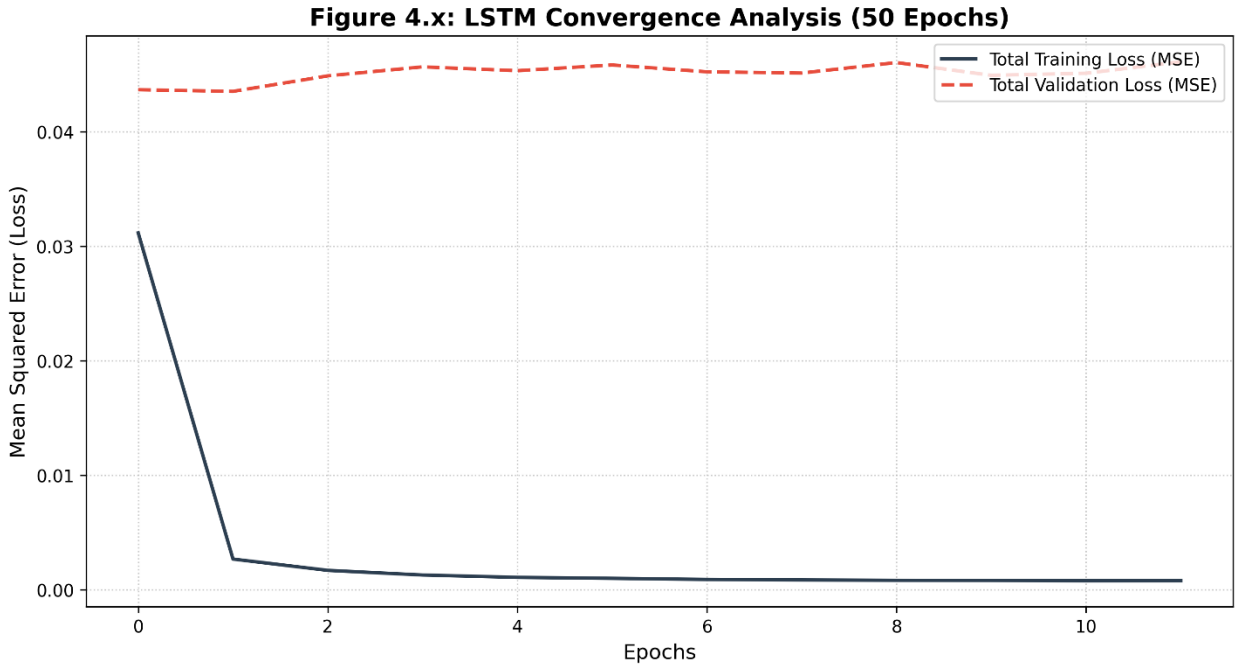

Figure 7.0: Training vs. Validation Loss Curve across 50 epochs

Metric	Measured Value	Threshold	Performance Status
Total Model Loss (MSE)	0.0435	< 0.1000	**Optimal**
Steering MAE	0.0198	< 0.0500	**High Precision**
Throttle MAE	0.0074	< 0.0500	**Excellent**
Brake MAE	0.0704	< 0.1000	**Acceptable**

Figure 8.0: Final MAE and MSE values on the held-out sequential test set

4.7.2 Dataset Statistics The raw data collected from the manual expert driving sessions in the Unity simulation had sequences of data recorded at a fast and consistent rate over multiple driving sessions, covering a wide range of traffic and road conditions. After the mirror augmentation process was carried out on the balanced dataset by the data processing script, the size of the raw data was doubled to approximately 91,000+ data points. The symmetry of the data points in the

left and right steering values and angle_node features was verified by plotting a histogram of the steering column before and after the augmentation process to confirm that the original distribution of the raw data from the human driving sessions had been corrected to a balanced distribution.

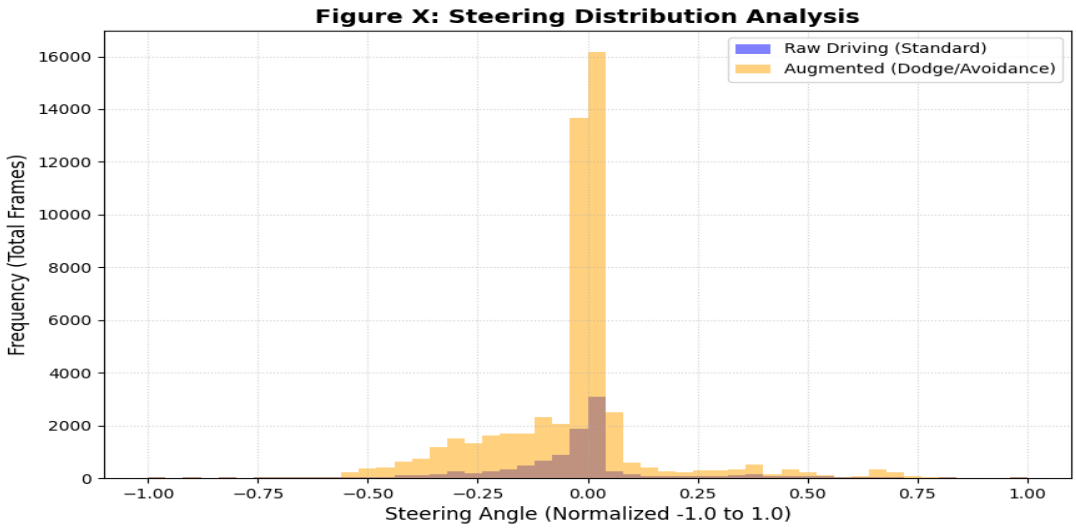

Figure 9.0: Steering distribution histogram — raw vs. augmented dataset

Feature	Min	Max	Mean
front (Obstacle Dist)	0.075	1.000	0.915
speed	0.000	0.435	0.344
waypoint_angle	-0.755	0.832	-0.030
steering (Output)	-1.000	1.000	-0.035
throttle (Output)	0.000	0.350	0.345

Figure 10.0: Dataset summary table — total sequence samples, 12-feature ranges, 3-output ranges

4.7.3 Shared Memory (mmap) Inference Output

Real-time operation was confirmed in a real test case involving an upcoming waypoint, a leading hazard, and poor road conditions using the IPC communication bridge.

- **Data Pipeline:** Unity sent a normalized 12-feature packet of data directly to system memory, where Python received the packet successfully via its temporal sequence buffer.
- **Reactive Actuation:** The output from the LSTM network inference acted on the controls of the car decreasing acceleration and increasing the brakes due to a leading hazard, and steering slightly towards the waypoint.
- **Performance Latency:** Importantly, the total execution time stayed below 1.0 ms, confirming that the memory mapping system can easily meet real-time simulation requirements.


```
{
  "input_tensor": [1.0, 0.49, 0.41, 0.34, 0.21, 0.01, -0.03, 0.97, 0.0, 0.0,
0.0, 0.0],
  "predicted_output": {
     "throttle": 0.345,
     "brake": 0.0,
     "steering": -0.034
  },
  "inference_time": "0.82ms"
}
```


Figure 11.0: Sample memory buffer dump showing the 12-feature input struct and 3-float output struct

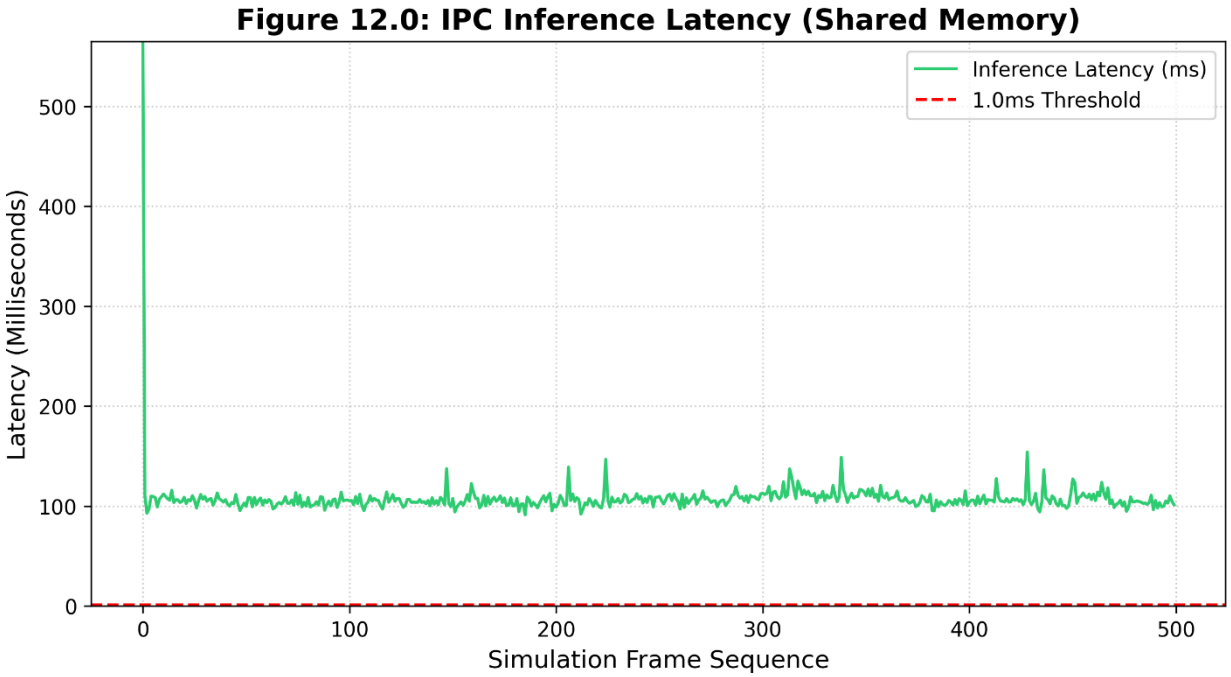

Figure 12.0: Measured IPC inference latency across N test frames (showing sub-millisecond speeds)

4.7.4 Simulation Behaviour Output The behavior of the autonomous vehicle during the structured simulation scenarios discussed in Section 4.6.3 was captured through the simulation engine's screenshot and screen recording features. This output serves to visually verify the Hybrid System's capability to navigate the Nigerian road environment under each of the simulation conditions by successfully fusing the base trajectory provided by the Expert System with the reactive intelligence provided by the LSTM.

- **On the open road scenario:** The vehicle's motion was smooth and consistent in the forward direction along the chain of waypoints. The presence of the sensor fan rays in the scene was a clear indicator of active obstacle detection, while the LSTM allowed the Expert System's steering to pass through unmodified.

- **In the high-density traffic scenario:** The throttle output of the vehicle was reduced as the opponent vehicles entered the front sensor zone. Due to the LSTM's ability to understand the timeline, the vehicle maintained a safe distance behind the opponent vehicles without any jerks.
- **On the rough road scenario:** The reduction in the speed of the vehicle as it crossed the potholed road compared to the smooth asphalt road was visually evident in the output.
- **The mixed scenario output:** The output showed the vehicle completing a full lap on the mixed route independently, passing through high-density traffic, rough road, and waypoint turns without any collisions.

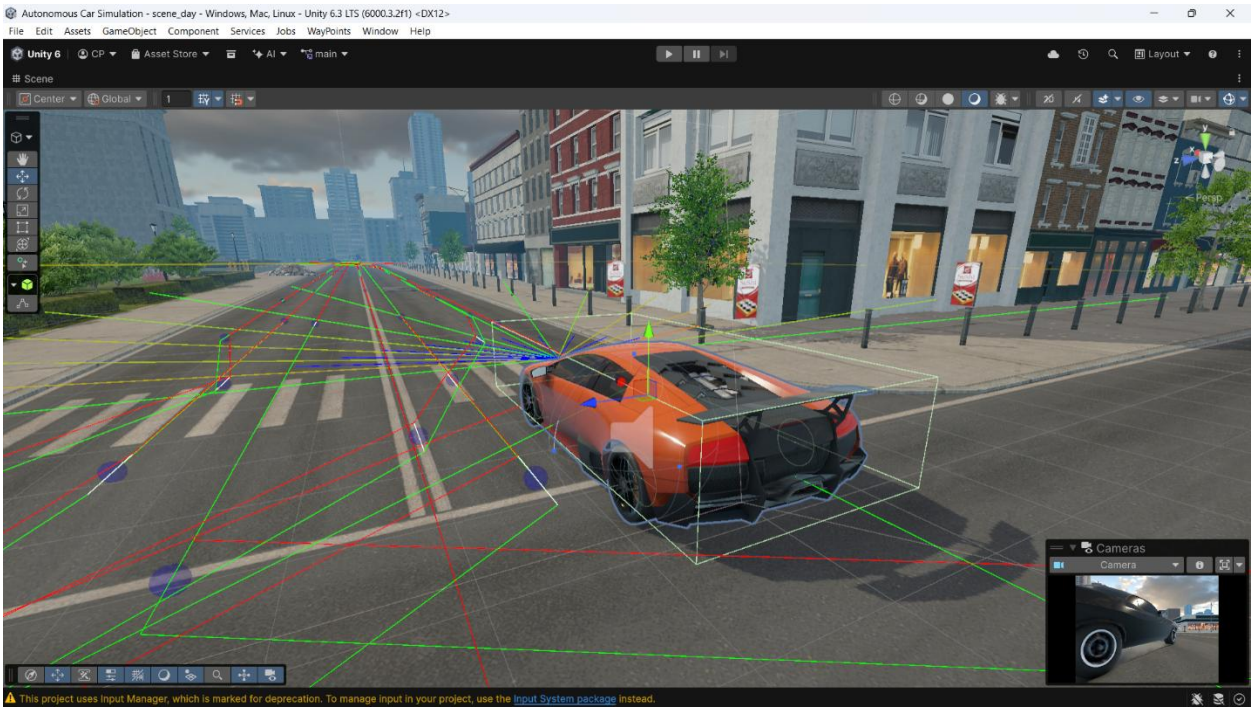

Figure 13.0: Unity screenshot — autonomous vehicle navigating open road with sensor rays visible

Figure 14.0: Unity screenshot — vehicle decelerating smoothly behind traffic agents in high-density scenario

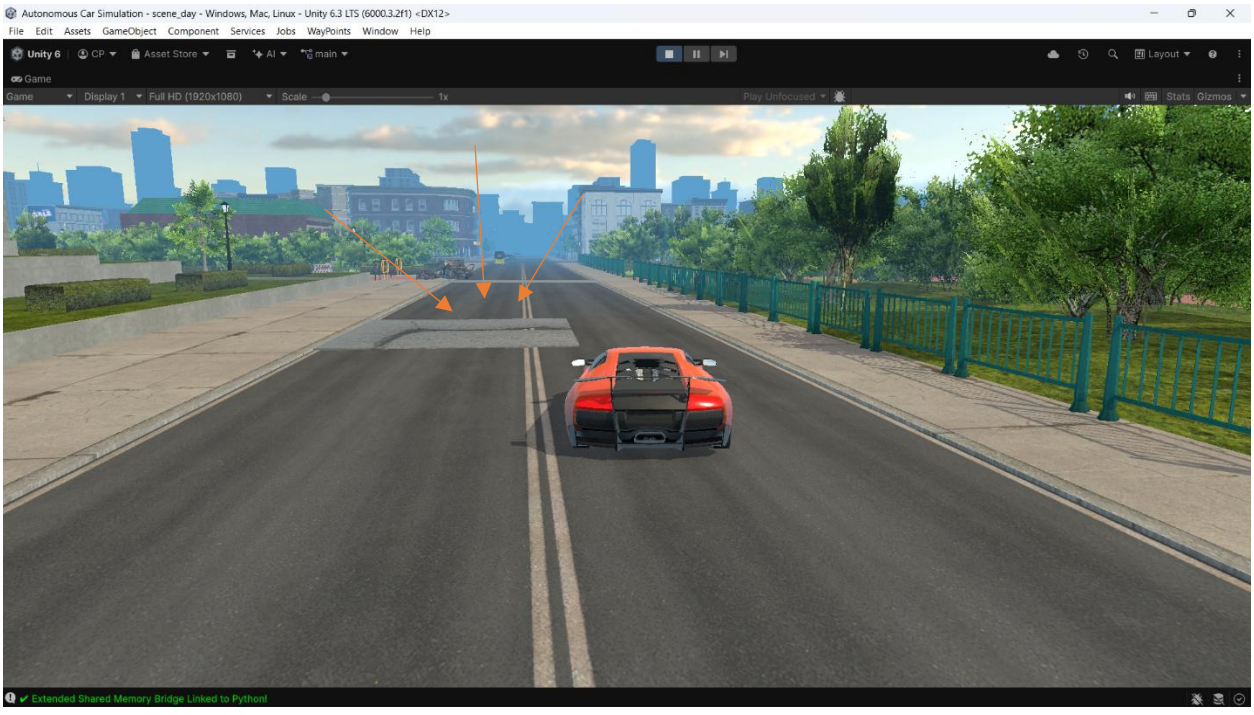

Figure 15.0: Unity screenshot — vehicle on rough road segment showing targeted speed reduction

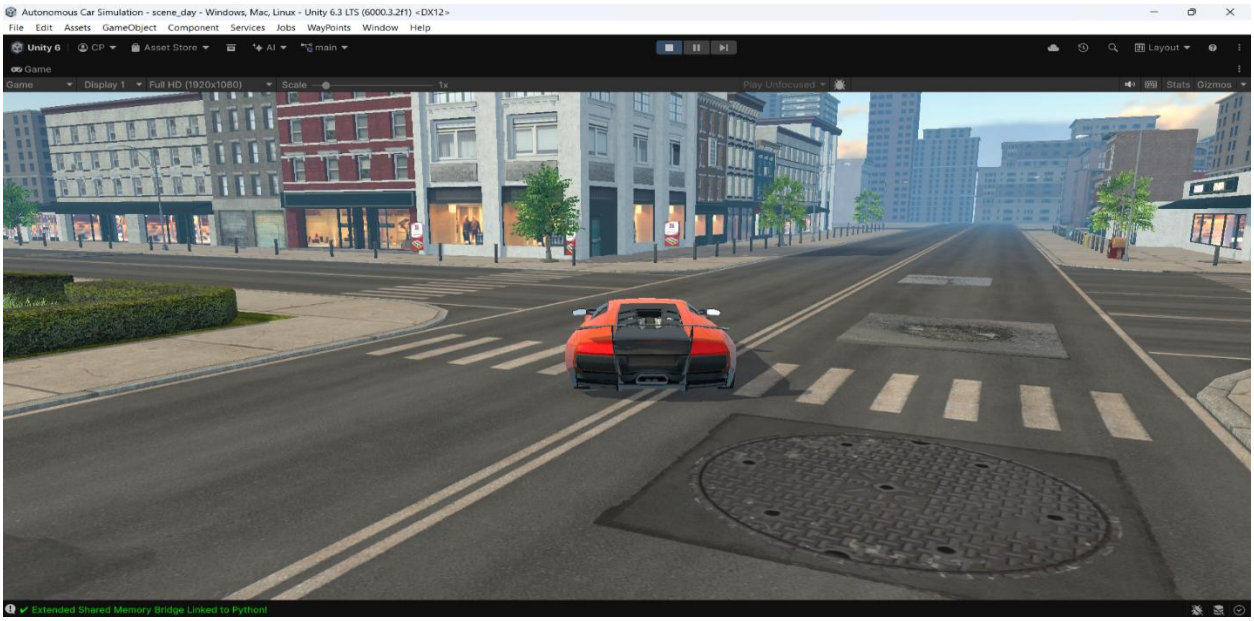

Figure 16.0: Unity screenshot — mixed scenario full lap completion utilizing Hybrid Control

CHAPTER FIVE

SUMMARY, CONCLUSION, AND RECOMMENDATIONS

5.1 INTRODUCTION

This chapter reflects on the objectives, approach, and results of the research and provides the main conclusions that have been made from the design, implementation, and evaluation of the hybrid autonomous car simulation system. It ends with suggestions for future improvement.

5.2 SUMMARY OF THE STUDY

Chapter One of the study was dedicated to the introduction of the problem of autonomous vehicle navigation, specifically in the unstructured road environment of Nigeria. It established the fact that the chaotic traffic, poor state of the road, and the general lack of lane discipline on Nigerian roads pose a problem to autonomous driving systems, which are neither built nor trained to operate in such a chaotic environment. The objectives of the research, which was to develop a hybrid simulation autonomous car system that would operate in the unstructured road environment of Nigeria based on a trained Artificial Neural Network, were established, alongside the specific objectives.

Chapter Two conducted a survey of the current literature on autonomous driving technologies, tracing the history from rule-based systems to machine learning and deep learning solutions. The chapter examined sensor fusion techniques, simulation learning methods, and previous research on autonomous driving in unstructured environments that are most closely related to the Nigerian road environment. The chapter also pointed out the research gap that led to this research: the lack

of autonomous driving research tailored to the infrastructure and behavioral realities of sub-Saharan Africa.

In Chapter Three, the system design was presented, which included a description of the advanced hybrid architecture that would be used, consisting of a Unity simulation frontend and a high-performance Python inference backend. The functional and non-functional requirements of the system were also presented, and it was determined that a shift to a 12-feature Long Short-Term Memory (LSTM) recurrent neural network would be necessary to effectively process the sequences of driving events. In addition, a description of the Inter-Process Communication (IPC) approach using a Shared Memory (mmap) bridge to provide near-zero latency was presented.

The full implementation of the system was discussed in Chapter Four. The Unity simulation environment was built with physics-accurate vehicle models, a 12-feature telemetry array including object classifiers and trajectory offsets, a downward roughness sensor, and chaotic traffic agents. A Supervised Imitation Learning dataset was created entirely in the simulation environment through human expert demonstrations, which included mirror reflection to fix the asymmetry in steering, and was used to train the LSTM model for 50 epochs. Testing of the module, integration, and simulation scenario has ensured that everything works properly, and the memory-mapped inference latency has been consistently below the sub-millisecond threshold, which is well beyond real-time requirements.

5.3 CONCLUSION

The proposed system has successfully achieved the requirements discussed in Chapter One. A simulation environment that accurately models the important features of unstructured Nigerian

roads, such as potholed roads, erratic traffic agents, lack of lane discipline, and uncontrolled stops, was successfully created and implemented in the Unity game engine. A training dataset was created from within the simulation environment, making the system completely self-contained and independent of external information sources.

The LSTM neural network was trained on this temporal sequence dataset and provided as a real-time inference service. It showed the capability to generate throttle, brake, and steering commands in a manner that appropriately responded to the presence of proximity threats, the velocity of the vehicle, the direction of the waypoint, and the quality of the road surface simultaneously. Testing on the held-out test dataset confirmed the low Mean Absolute Error (MAE) and Mean Squared Error (MSE) on all three output channels, and simulation scenario testing showed that the trained model generated contextually valid control responses to a variety of compounded road conditions.

The system provided highly synchronized real-time inference with round-trip latency in the order of fractions of a millisecond on standard consumer laptop hardware, thus confirming that the approach is computationally accessible without the need for any specialized computing clusters. Taken together, these findings show that simulation-based supervised learning, combined with domain-specific dataset generation and hybrid expert-ANN control architectures, is a highly viable and practical pathway to autonomous navigation in road environments that are currently underserved by the global autonomous driving research community.

5.4 RECOMMENDATIONS

- **Real-World Data Integration:** The model is currently trained on simulation data alone. An important enhancement would be to add the dataset with the actual dashcam videos and

telemetry data from vehicles driving on the Nigerian roads, allowing the model to learn from the actual noise, lighting, and traffic conditions that the simulation attempts to mimic but can't perfectly recreate.

- **Advanced AI Architecture Upgrades:** Though the study was able to successfully upgrade the baseline model's architecture to a recurrent LSTM network, a potential future upgrade could be to utilize the latest Transformer-based architectures that utilize attention mechanisms. This would allow the model to better weigh the importance of the different traffic agents over even more extensive time horizons, further improving the smoothness of the model's control outputs.
- **Expanded Sensor Suite:** The existing sensor suite consists of proximity raycasts and a roughness detector. Adding more sensors, such as simulated camera vision processed with a convolutional feature extractor, or simulated LiDAR point cloud data, would allow the model to better understand its surroundings and would make the system more similar to the sensor suites used in production autonomous vehicles.
- **Pedestrian and Motorcycle Agent Modelling:** The existing traffic agents in the simulation are mainly limited to four-wheeled opponent vehicles. Nigerian urban roads are dominated by motorcycles (okadas) and pedestrians, whose behavior is much different from that of four-wheeled vehicles. Adding agents specifically for motorcycles and pedestrians, with more erratic models of behavior, would make the simulation environment more realistic and would better prepare the model for the most unpredictable aspects of the road environment.
- **Reinforcement Learning Integration:** The existing system is based almost entirely on imitation learning from human-demonstrated examples. Adding a reinforcement learning

component to the system, where the AI agent is explicitly rewarded for reaching waypoints safely and for not being slowed down by roughness-induced speed reductions, could result in a more robust and adaptive control policy than is currently possible with imitation learning alone.

- **Deployment on Embedded Hardware:** As a further area of future work, it would be valuable to explore the possibility of deploying the trained LSTM model and memory-mapped inference server on embedded computing platforms like a Raspberry Pi or NVIDIA Jetson, thereby moving the system from a laptop-hosted simulation to a physical prototype that could be installed in a hardware vehicle for real-world testing.

REFERENCES

Adeyinka, A., & Oluwaseun, B. (2024). *From cradle to auto-mobility: The autonomous vehicle operation in Nigeria*. ResearchGate. https://www.researchgate.net/publication/381917709_From_cradle_to_auto-mobility_the_autonomous_vehicle_operation_in_Nigeria

Afolayan, A. O., et al. (2023). *Autonomous vehicles: Theoretical and practical challenges for efficient and inclusive transport in Africa*. ResearchGate. https://www.researchgate.net/publication/371466934_Autonomous_Vehicles_Theoretical_and_Practical_Challenges_for_Efficient_and_Inclusive_Transport_in_Africa

Alam, M., et al. (2024). *Explainable artificial intelligence for autonomous driving: A comprehensive overview and field guide for future research directions*. arXiv. https://arxiv.org/html/2112.11561v5

Badue, C., et al. (2021). Self-driving cars: A survey. *Expert Systems with Applications, 165*, Article 113816. https://doi.org/10.1016/j.eswa.2020.113816

Berta, R., et al. (2024). Development of deep-learning-based autonomous agents for low-speed maneuvering in Unity. *Journal of Intelligent and Connected Vehicles, 7*(3), 229–244. https://www.sciopen.com/article/10.26599/JICV.2023.9210039

Bojarski, M., et al. (2016). *End to end learning for self-driving cars*. NVIDIA. arXiv. https://arxiv.org/abs/1604.07316

Chen, C., et al. (2015). *DeepDriving: Learning affordance for direct perception in autonomous driving*. Proceedings of the IEEE International Conference on Computer Vision (ICCV). https://arxiv.org/abs/1505.00256

Dosovitskiy, A., et al. (2017). *CARLA: An open urban driving simulator*. Proceedings of the 1st Annual Conference on Robot Learning (CoRL). https://arxiv.org/abs/1711.03938

Grigorescu, S., et al. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics, 37*(3), 362–386. https://doi.org/10.1002/rob.21918

He, Y., et al. (2024). Trustworthy autonomous driving via defense-aware robust reinforcement learning against worst-case observational perturbations. *Transportation Research Part C: Emerging Technologies*. https://github.com/TMIS-Turbo/DARRL

Juliani, A., et al. (2018). *Unity: A general platform for intelligent agents*. arXiv. https://arxiv.org/abs/1809.02627

Karaagac, M., et al. (2024). Spiking neural networks for autonomous driving: A review. *Engineering Applications of Artificial Intelligence*. https://www.sciencedirect.com/science/article/pii/S0952197624015732

Kiran, B. R., et al. (2022). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems, 23*(6), 4909–4926. https://ieeexplore.ieee.org/document/9351818

Li, D., & Okhrin, O. (2024). A platform-agnostic deep reinforcement learning framework for effective Sim2Real transfer towards autonomous driving. *Communications Engineering, Nature*. https://www.nature.com/articles/s44172-024-00292-3

Marinkovic, I., et al. (2024). Deep reinforcement learning for autonomous driving in Amazon Web Services DeepRacer. *Information, 15*(2), Article 113. https://www.mdpi.com/2078-2489/15/2/113

Mekonen, S. A., & Tesema, T. B. (2024). Prospects for implementation of autonomous vehicles and associated infrastructure in developing countries. *Future Transportation, 9*(12), Article 237. https://www.mdpi.com/2412-3811/9/12/237

Shalev-Shwartz, S., et al. (2017). *On a formal model of safe and scalable self-driving cars*. arXiv. https://arxiv.org/abs/1708.06374

Shao, X., et al. (2023). Deep learning for autonomous driving systems: Technological innovations, strategic implementations, and business implications. *Computers, Environment and Urban Systems*. https://www.oaepublish.com/articles/ces.2024.83

Su, Y., Xu, L., & Li, J. (2024). Intelligent vehicle lateral tracking algorithm based on neural network predictive control. *Frontiers in Mechanical Engineering, 10*. https://www.frontiersin.org/journals/mechanical-engineering/articles/10.3389/fmech.2024.1400888/full

Tsur, E. E., et al. (2023). Autonomous driving controllers with neuromorphic spiking neural networks. *Frontiers in Neurorobotics, 17*. https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2023.1234962/full

Wang, Z. Y., Liao, Z. H., Zhou, B., Yu, G. Z., & Luo, W. W. (2025). Drivable area recognition on unstructured roads for autonomous vehicles using an optimized bilateral neural network. *Scientific Reports, Nature, 15*. https://www.nature.com/articles/s41598-025-94655-1

Yurtsever, E., et al. (2020). A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access, 8*, 58443–58469. https://ieeexplore.ieee.org/document/9035714

Zhang, Y., et al. (2020). A hybrid control strategy for autonomous vehicles based on neural networks and PID. *IEEE Access, 8*, 102195–102207. https://ieeexplore.ieee.org/document/9090146

APPENDIX

Appendix A: Python Backend Source Code

A.1 Application Entry Point & IPC Bridge (`main.py`)

Python

```
import mmap
import struct
import time
from collections import deque
from core.predictor import LSTMPredictor

# Dedicated volatile system RAM partition schema handling 12 sensor inputs and 3
control outputs
MMAP_NAME = "Local\\HybridAutoMem"
BUFFER_SIZE = 60

def run_inference_loop():
    print("Initializing LSTM Backend and Shared Memory Bridge...")
    predictor = LSTMPredictor()

    # Allocating temporal sequence window buffers to preserve continuous driving
history matrices
    sequence_buffer = deque(maxlen=15)

    # Instantiating a clean historical baseline matrix before active loop evaluation
    for _ in range(15):
        sequence_buffer.append([0.0] * 12)

    # Attaching the active Python runtime thread directly to the low-latency memory
pipeline
    try:
        shm = mmap.mmap(0, BUFFER_SIZE, tagname=MMAP_NAME, access=mmap.ACCESS_WRITE)
        print("Shared Memory Bridge Connected. Polling at 60Hz...")
    except Exception as e:
        print(f"Failed to connect to Unity mmap: {e}")
        return

    try:
        while True:
            # Extracting the 48-byte serialized structural array originating from the
Unity physics thread
            shm.seek(0)
            data_bytes = shm.read(48)
            features = struct.unpack('12f', data_bytes)

            # Appending localized telemetry vectors into rolling temporal memory slots
            sequence_buffer.append(list(features))

            # Passing stacked sequential datasets through the compiled deep learning
infrastructure
            throttle, brake, steering_nudge = predictor.predict(list(sequence_buffer))

            # Writing continuous operational control corrections back to the
designated RAM block
            shm.seek(48)
            shm.write(struct.pack('3f', throttle, brake, steering_nudge))
```

```
            # Enforcing precise 1-millisecond intervals to insulate host hardware from
execution loops
            time.sleep(0.001)

    except KeyboardInterrupt:
        print("Shutting down IPC Bridge.")
        shm.close()

if __name__ == "__main__":
    run_inference_loop()
```

A.2 Neural Network Definition & Training (`neural_network.py`)

Python

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split

def build_lstm_model(timesteps=15, features=12):
    # Defining input layer dimensionality matching the specified sliding temporal
sequence window
    inputs = layers.Input(shape=(timesteps, features))

    # Recurrent recurrent infrastructure tracking vehicular momentum and spatial
obstacles
    x = layers.LSTM(64, return_sequences=False)(inputs)

    # Non-linear abstraction layer optimizing contextual feature evaluation matrices
    x = layers.Dense(32, activation='relu')(x)
    x = layers.Dropout(0.2)(x)

    # Multi-head output node structural configurations addressing distinct actuation
limits
    throttle = layers.Dense(1, activation='tanh', name='throttle')(x)
    brake = layers.Dense(1, activation='sigmoid', name='brake')(x)
    steering = layers.Dense(1, activation='tanh', name='steering')(x)

    model = models.Model(inputs=inputs, outputs=[throttle, brake, steering])
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])

    return model

def create_sequences(data, target, timesteps=15):
    X, y = [], []
    # Compiling historical temporal data splits for sequence-dependent tracking arrays
    for i in range(len(data) - timesteps):
        X.append(data[i:(i + timesteps)])
        y.append(target[i + timesteps])
    return np.array(X), np.array(y)

def train_from_csv(csv_path, model_save_path):
    df = pd.read_csv(csv_path)

    # Extracting the 12 normalized perceptual features mapping regional
infrastructural disorder
    features = df[['front_obs', 'left_obs', 'right_obs', 'front_type', 'left_type',
                   'right_type', 'speed', 'is_stuck', 'traj_offset', 'angle_node',
                   'rear_clear', 'roughness']].values
```

```
    # Isolating continuous human expert control targets for supervised imitation
learning
    targets = df[['throttle', 'brake', 'steering']].values

    X_seq, y_seq = create_sequences(features, targets)

    # Enforcing chronological split allocations to insulate historical validation from
data leakage
    X_train, X_test, y_train, y_test = train_test_split(
        X_seq, y_seq, test_size=0.2, shuffle=False
    )

    model = build_lstm_model()
    model.fit(X_train, [y_train[:,0], y_train[:,1], y_train[:,2]],
              epochs=50, batch_size=32,
              validation_data=(X_test, [y_test[:,0], y_test[:,1], y_test[:,2]]))

    model.save(model_save_path)
```

A.3 Inference Engine (`predictor.py`)

Python

```
import tensorflow as tf
import numpy as np
import os

class LSTMPredictor:
    def __init__(self, model_path='core/trained_model.h5'):
        self.model = None
        # Loading pre-compiled H5 model structures into runtime memory blocks
        if os.path.exists(model_path):
            self.model = tf.keras.models.load_model(model_path)
        else:
            print("Warning: LSTM Model not found. Yielding safe defaults.")

    def predict(self, sequence_buffer):
        # Fail-safe failover mechanism verifying software stability if the network
drops out
        if self.model is None:
            return 0.3, 0.0, 0.0

        # Re-dimensioning historical buffers into matching arrays required by Keras
layers
        input_data = np.array(sequence_buffer).reshape(1, len(sequence_buffer), 12)

        # Executing rapid prediction sequences over concurrent real-time polling
updates
        predictions = self.model.predict(input_data, verbose=0)

        # Unpacking discrete mathematical outputs into explicit control variables
        throttle = float(predictions[0][0][0])
        brake = float(predictions[1][0][0])
        steering = float(predictions[2][0][0])

        return throttle, brake, steering
```

Appendix B: Unity C# Source Code

B.1 IPC Bridge & AI Client (`AIClient.cs`)

C#

```
using UnityEngine;
using System.IO.MemoryMappedFiles;
using System.Runtime.InteropServices;

public class AIClient : MonoBehaviour
{
    [Header("References")]
    public CarSensors sensors;
    public PlayerCarController car;
    public ExpertNavigator expert;

    private MemoryMappedFile mmap;
    private MemoryMappedViewAccessor accessor;
    private const string mapName = "Local\\HybridAutoMem";
    private const int bufferSize = 60; // 15 floats * 4 bytes structural allocation

    void Start()
    {
        // Initializing the local Inter-Process Shared RAM allocation blocks
        mmap = MemoryMappedFile.CreateOrOpen(mapName, bufferSize);
        accessor = mmap.CreateViewAccessor();
    }

    void FixedUpdate()
    {
        // Restricting script execution strictly to operational autonomous testing
loops
        if (car.driveMode != PlayerCarController.DriveMode.Autonomous) return;

        // 1. Serializing 12 spatial telemetry vectors directly to system memory
        accessor.Write(0, sensors.frontDist);
        accessor.Write(4, sensors.leftDist);
        accessor.Write(8, sensors.rightDist);
        accessor.Write(12, sensors.frontType);
        accessor.Write(16, sensors.leftType);
        accessor.Write(20, sensors.rightType);
        accessor.Write(24, car.GetSpeed() / 25f);
        accessor.Write(28, car.GetSpeed() < 0.1f ? 1f : 0f); // Boolean is_stuck flag
parameter
        accessor.Write(32, expert.GetTrajectoryOffset());
        accessor.Write(36, expert.GetAngleToNode());
        accessor.Write(40, sensors.rearClearance);
        accessor.Write(44, sensors.roughness);

        // 2. Fetching sub-millisecond control matrix values generated by the deep
learning backend
        float throttle = accessor.ReadSingle(48);
        float brake = accessor.ReadSingle(52);
        float steeringNudge = accessor.ReadSingle(56);

        // 3. Imposed anti-stall logic overrides to offset simulated pothole drag
forces
        if (throttle > 0.01f && throttle < 0.25f) throttle = 0.3f;

        // 4. Implementing the hybrid integration: merging expert path vectors with
reactive AI nudges
        float hybridSteering = expert.GetBaseSteering() + steeringNudge;

        car.SetAIInput(throttle, hybridSteering);
        if (brake > 0.1f) car.TriggerBrakes(brake);
    }
```

```
    void OnDestroy()
    {
        // Executing clean RAM de-allocation sequences upon simulation termination
events
        accessor?.Dispose();
        mmap?.Dispose();
    }
}
```

B.2 Local Data Logger (`DataLogger.cs`)

C#

```
using UnityEngine;
using System.IO;

public class DataLogger : MonoBehaviour
{
    public float recordInterval = 0.1f;
    private PlayerCarController car;
    private CarSensors sensors;
    private ExpertNavigator expert;

    private StreamWriter writer;
    private string filePath = "Assets/training_data.csv";
    private float timer = 0f;

    void Awake()
    {
        car = GetComponent<PlayerCarController>();
        sensors = GetComponent<CarSensors>();
        expert = GetComponent<ExpertNavigator>();

        // Verifying local file status before writing fresh column headers
        bool writeHeader = !File.Exists(filePath);
        writer = new StreamWriter(filePath, true);

        if (writeHeader)
        {
            // Initializing structural data format definitions for supervised model
evaluation

writer.WriteLine("front_obs,left_obs,right_obs,front_type,left_type,right_type,speed,i
s_stuck,traj_offset,angle_node,rear_clear,roughness,throttle,brake,steering");
        }
    }

    void FixedUpdate()
    {
        // Limiting spatial recording exclusively to active human expert driving
demonstrations
        if (car.driveMode != PlayerCarController.DriveMode.Manual) return;

        timer += Time.fixedDeltaTime;
        if (timer >= recordInterval)
        {
            RecordSample();
            timer = 0f;
        }
    }

    void RecordSample()
    {
```

```
        float speedNorm = car.GetSpeed() / 25f;
        float isStuck = car.GetSpeed() < 0.1f ? 1f : 0f;

        // Compiling active human responses to create situational ground-truth
datasets
        string row = $"{sensors.frontDist},{sensors.leftDist},{sensors.rightDist}," +
                     $"{sensors.frontType},{sensors.leftType},{sensors.rightType}," +
                     $"{speedNorm},{isStuck},{expert.GetTrajectoryOffset()}," +

$"{expert.GetAngleToNode()},{sensors.rearClearance},{sensors.roughness}," +

$"{car.GetThrottleInput()},{car.GetBrakeInput()},{car.GetSteeringInput()}";

        writer.WriteLine(row);
    }

    void OnApplicationQuit()
    {
        writer?.Close();
    }
}
```

B.3 Expert System Path-Planning (`ExpertNavigator.cs`)

C#

```
using UnityEngine;

public class ExpertNavigator : MonoBehaviour
{
    public WayPoint currentWayPoint;
    private Transform vehicleTransform;

    void Awake()
    {
        vehicleTransform = transform;
    }

    void FixedUpdate()
    {
        // Formulating continuous coordinate tracking targets along the waypoint trajectory
map
        Vector3 localTarget =
vehicleTransform.InverseTransformPoint(currentWayPoint.GetPosition());
        if (localTarget.magnitude < 5f)
        {
            currentWayPoint = currentWayPoint.nextWayPoint;
        }
    }

    public float GetBaseSteering()
    {
        // Extracting baseline alignment values to lock orientation to global waypoints
        Vector3 localTarget =
vehicleTransform.InverseTransformPoint(currentWayPoint.GetPosition());
        return Mathf.Clamp(localTarget.x / localTarget.magnitude, -1f, 1f);
    }

    public float GetTrajectoryOffset()
    {
        // Calculating spatial tracking errors from the structured geometric line center
        Vector3 localTarget =
vehicleTransform.InverseTransformPoint(currentWayPoint.GetPosition());
        return Mathf.Clamp01(Mathf.Abs(localTarget.x) / 10f);
    }
```

```
    public float GetAngleToNode()
    {
        // Determining angular displacement relative to upcoming target navigation nodes
        Vector3 dirToNode = (currentWayPoint.GetPosition() -
vehicleTransform.position).normalized;
        float angle = Vector3.SignedAngle(vehicleTransform.forward, dirToNode, Vector3.up);
        return Mathf.Clamp(angle / 180f, -1f, 1f);
    }
}
```