OPEN ACCESS Freely available online

Journal of Research and Development

Review Article

A Comparative Analysis of CARLA and AirSim Simulators: Investigating Implementation Challenges in Autonomous Driving

Manav Khambhayata*

Department of Computer Science, Chandigarh University, Chandigarh, India

ABSTRACT

The advancement of autonomous driving technologies relies heavily on effective training methodologies for self-driving car AI. Reinforcement learning has emerged as a promising approach in this domain. In this paper, we present a comparative analysis of training strategies for self-driving cars using two popular simulators: CARLA and AirSim. We focus solely on the comparison between the two simulators by implementing them using current technology, analyzing their ease of implementation, and identifying the associated challenges. CARLA offers ease of setup and a realistic environment, while AirSim provides excellent overall performance despite its challenging setup process. However, integrating CARLA with TensorFlow poses certain difficulties. To conduct the comparative analysis, we implemented reinforcement learning algorithms on both simulators and evaluated their performance metrics, including training time, learning efficiency, and generalization to real-world scenarios. Our findings indicate that CARLA, despite its ease of setup, encountered challenges when integrating with TensorFlow due to compatibility issues. However, once resolved, CARLA demonstrated promising results in terms of learning efficiency and generalization to real- world scenarios, outperforming conventional methods. On the other hand, AirSim showcased superior overall performance but required substantial effort in setting up the simulator and configuring the environment. We provide insights into the strengths and weaknesses of each simulator and offer recommendations for choosing the most suitable training platform based on specific research requirements.

Keywords: Reinforcement learning; CARLA; AirSim; TensorFlow

INTRODUCTION

No the development of autonomous driving technologies has experienced remarkable progress, largely driven by advancements in Artificial Intelligence (AI) and machine learning. Reinforcement Learning (RL), a subfield of machine learning, has emerged as a promising approach for training AI systems in self-driving cars. RL enables autonomous vehicles to learn decision-making models by interacting with their environment, making it an ideal technique for addressing complex driving scenarios.

Simulators play a pivotal role in the development and evaluation of AI algorithms for self- driving cars. They provide a controlled and safe environment for training and testing, enabling researchers and developers to iterate rapidly without the risks associated with real- world experiments. Among the leading simulators used in autonomous driving research, CARLA and AirSim have gained prominence.

CARLA, an open-source simulator developed by the Computer Vision center at the Universitãt Autònoma de Barcelona, offers a highly realistic platform for autonomous driving. It encompasses features such as a rich urban environment, sensor models, and a variety of traffic scenarios. CARLA boasts a user-friendly setup process, making it an attractive option for researchers and practitioners. However, integrating CARLA with popular deep learning frameworks like TensorFlow can present challenges due to compatibility issues and additional configuration requirements.

Correspondence to: Manav Khambhayata, Department of Computer Science, Chandigarh University, Chandigarh, India; E-mail: mumairhassanqais11@gmail.com

Received: 20-Jul-2023, Manuscript No. JRD-23-25779; **Editor assigned:** 24-Jul-2023, PreQC No. JRD-23-25779 (PQ); **Reviewed:** 07-Aug-2023, QC No. JRD-23-25779; **Revised:** 16-Dec-2024, Manuscript No. JRD-23-25779 (R); **Published:** 23-Dec -2024, DOI: 10.35248/2311-3278.24.10.281

Citation: Khambhayata M (2024) A Comparative Analysis of CARLA and AirSim Simulators: Investigating Implementation Challenges in Autonomous Driving. J Res Dev. 12:281.

On the other hand, AirSim, an open-source simulator developed by microsoft research, focuses on high-fidelity physics-based simulation and provides realistic environments spanning urban, rural, and off-road scenarios. AirSim offers flexibility in terms of customization and sensor modeling. However, its setup process is more complex, which may pose a steep learning curve for users.

In this paper, our primary objective is to perform a comprehensive comparative analysis of CARLA and AirSim simulators by implementing them using current technology, analyzing their ease of implementation, and identifying the associated challenges. We concentrate solely on comparing these two simulators to gain valuable insights into their respective strengths and weaknesses. By focusing on the implementation aspect, we aim to provide practical guidance to researchers and practitioners in selecting the most suitable training platform based on their specific needs.

To achieve our objective, we implemented and evaluated various reinforcement learning algorithms on both CARLA and AirSim simulators. However, the primary emphasis of our analysis lies in identifying and addressing the challenges encountered during the implementation process. We sought to gain a deep understanding of the difficulties and complexities associated with setting up and integrating CARLA with TensorFlow, as well as the challenges involved in configuring the AirSim simulator. By prioritizing these challenges, we provide valuable insights into the practical aspects of implementing these simulators using current technology.

The outcomes of this research contribute significantly to the understanding of implementing CARLA and AirSim simulators using current technology. By conducting a thorough comparison, we provide insights into the ease of implementation and associated challenges of each simulator. This study serves as a valuable resource for researchers and developers in the autonomous driving domain, assisting them in making informed decisions and optimizing their training methodologies for autonomous vehicles. Ultimately, our findings aim to drive further advancements in this exciting field.

LITERATURE REVIEW

In this section, the literature review investigates and analyzes two influential papers that have contributed significantly to the field. The selected papers, "Deep reinforcement learning for autonomous driving" by Intel Labs and "reinforcement learning-based control for autonomous driving" by Stanford University, are examined in detail to understand their methodologies, key findings, and contributions to the advancement of autonomous driving [1,2]. By critically analyzing these papers, this literature review aims to provide a comprehensive understanding of the current state of research in the application of reinforcement learning techniques for autonomous driving and to identify any gaps or areas for further investigation.

“Deep reinforcement learning for autonomous driving" by Intel labs

The paper "Deep reinforcement learning for autonomous driving" by Intel labs explores the application of Deep Reinforcement Learning (DRL) techniques in the field of autonomous driving [1]. The authors acknowledge the limitations of traditional approaches that often rely on handcrafted rules and heuristics, which may not capture the complex dynamics of real-world driving scenarios. In response to these challenges, the paper proposes a novel framework that leverages DRL algorithms to enable autonomous vehicles to learn driving behaviors directly from raw sensor data.

The paper begins by providing a comprehensive overview of reinforcement learning and its potential in the context of autonomous driving. It discusses the fundamental concepts of state, action, reward, and the Markov Decision Process (MDP) formulation. The authors emphasize the advantages of deep reinforcement learning, which allows the agents to learn complex representations and make decisions based on raw sensor inputs.

To demonstrate the efficacy of their approach, the authors present an experimental setup in which the DRL agent is trained in a simulated driving environment. They describe the key components of their framework, including the perception module, action space representation, reward function design, and the training process itself. The training process involves iterative updates using state-of-the-art DRL algorithms like Deep Q-Networks (DQN) or Proximal Policy Optimization (PPO).

The results of their experiments reveal promising performance in terms of decision-making and control in various driving scenarios, including urban, highway, and off-road conditions. The DRL agent exhibits the ability to navigate complex environments, handle dynamic obstacles, and make informed driving decisions. The paper also discusses the challenges and limitations of their approach, such as the need for significant computational resources and the difficulty of scaling these techniques to real-world applications.

One of the key applications of this research paper is the improvement of driving behaviors. Traditional approaches in autonomous driving often rely on handcrafted rules and heuristics, which may not fully capture the complex dynamics of real-world driving scenarios. By utilizing DRL algorithms, the paper aims to enable autonomous vehicles to learn driving behaviors directly from raw sensor data. This approach has the potential to enhance the adaptability and decision-making capabilities of autonomous vehicles, allowing them to navigate various driving scenarios such as urban, highway, and off-road conditions.

Another important application of this research paper is its contribution to real-world deployment of autonomous driving systems. Deploying autonomous vehicles in real-world environments poses numerous challenges, and the paper recognizes the limitations of traditional approaches in addressing these challenges.

By leveraging DRL algorithms, the authors aim to overcome these limitations and provide a more effective solution. The ability of DRL algorithms to learn from raw sensor data and make decisions based on complex representations offers promising potential for improving the performance and safety of autonomous driving systems.

Overall, the paper "deep reinforcement learning for autonomous driving" provides valuable insights into the potential of using DRL algorithms to enhance autonomous driving systems. It highlights the advantages of learning driving behaviors directly from raw sensor data and presents encouraging results in simulated environments. While there are challenges to overcome for real-world deployment, the paper serves as a foundation for further research and development in the exciting intersection of deep reinforcement learning and autonomous driving.

"Reinforcement learning-based control for autonomous driving" by Stanford University

The paper "Reinforcement learning-based control for autonomous driving" by Stanford University explores the use of Reinforcement Learning (RL) techniques in the context of autonomous driving [2]. The authors address the limitations of traditional rule-based approaches, which often struggle to handle the complexity and variability of real-world driving scenarios. To overcome these challenges, the paper proposes an end-to-end learning framework that integrates perception, planning, and control using RL algorithms.

The paper begins by providing a comprehensive overview of the principles of RL and its application to autonomous driving. It explains the key concepts of state, action, reward, and the Markov Decision Process (MDP) formulation. The authors emphasize the potential of RL to learn driving policies directly from data and highlight the importance of learning from human demonstrations to enhance the training process.

The proposed framework consists of several components, including perception modules, a high-level planner, and a low-level controller. RL algorithms such as Deep Deterministic Policy Gradient (DDPG) or Trust Region Policy Optimization (TRPO) are employed to train the driving policy. The authors emphasize the significance of incorporating expert knowledge and human demonstrations to improve the safety and performance of the learned policy.

To evaluate the effectiveness of their approach, the authors conduct extensive experiments in various driving scenarios, including urban and highway environments. The results demonstrate improved driving accuracy, robustness, and generalization capabilities compared to traditional rule-based methods. The paper also discusses the challenges associated with real-world deployment, including safety concerns, scalability, and interpretability of RL-based control systems.

One of the key applications of this research paper is the improvement of driving accuracy and robustness. Traditional rule-based approaches often struggle to handle the complexity and variability of real-world driving scenarios. By leveraging RL techniques, the authors propose an end-to-end learning framework that integrates perception, planning, and control. This framework enables the autonomous vehicle to learn driving policies directly from data, resulting in improved accuracy and robustness in various driving environments, including urban and highway scenarios.

Another important application of this research paper is the inclusion of human demonstrations and expert knowledge to enhance the safety and performance of the learned policy. RL algorithms typically require extensive training, which can be time-consuming and potentially unsafe in real-world settings. By incorporating human demonstrations and expert knowledge into the training process, the authors aim to accelerate the learning process and improve the reliability and safety of the autonomous driving system. This application highlights the practicality and effectiveness of RL-based control strategies in autonomous driving.

Furthermore, the research paper contributes to the advancement of RL-based control systems for autonomous driving by discussing the challenges associated with real-world deployment. Safety concerns, scalability, and interpretability are crucial factors that need to be addressed when deploying RL-based control systems in autonomous vehicles. By acknowledging these challenges, the authors provide valuable insights into the practical considerations and limitations of RL techniques in real-world autonomous driving applications.

Overall, the research paper "Reinforcement learning-based control for autonomous driving" by Stanford University presents applications of RL techniques in the context of autonomous driving. The focus on improving driving accuracy and robustness, incorporating human demonstrations and expert knowledge, and addressing the challenges of real-world deployment contributes to the development of more effective and reliable autonomous driving systems. This research paper serves as a foundation for further exploration and advancement of RL-based control strategies in the field of autonomous driving.

Methodology

The methodology for building a reinforcement learning AI for autonomous driving involves a series of well-defined steps. We recognized that most of the processes, including data collection, preprocessing, reinforcement learning algorithm selection, agent design, training, evaluation, and fine-tuning, were similar for both CARLA and AirSim simulators. These common steps provided a solid foundation for implementing the AI models in both environments. However, our main focus in this research was on the implementation process, as it presented unique challenges and considerations (Figure 1).

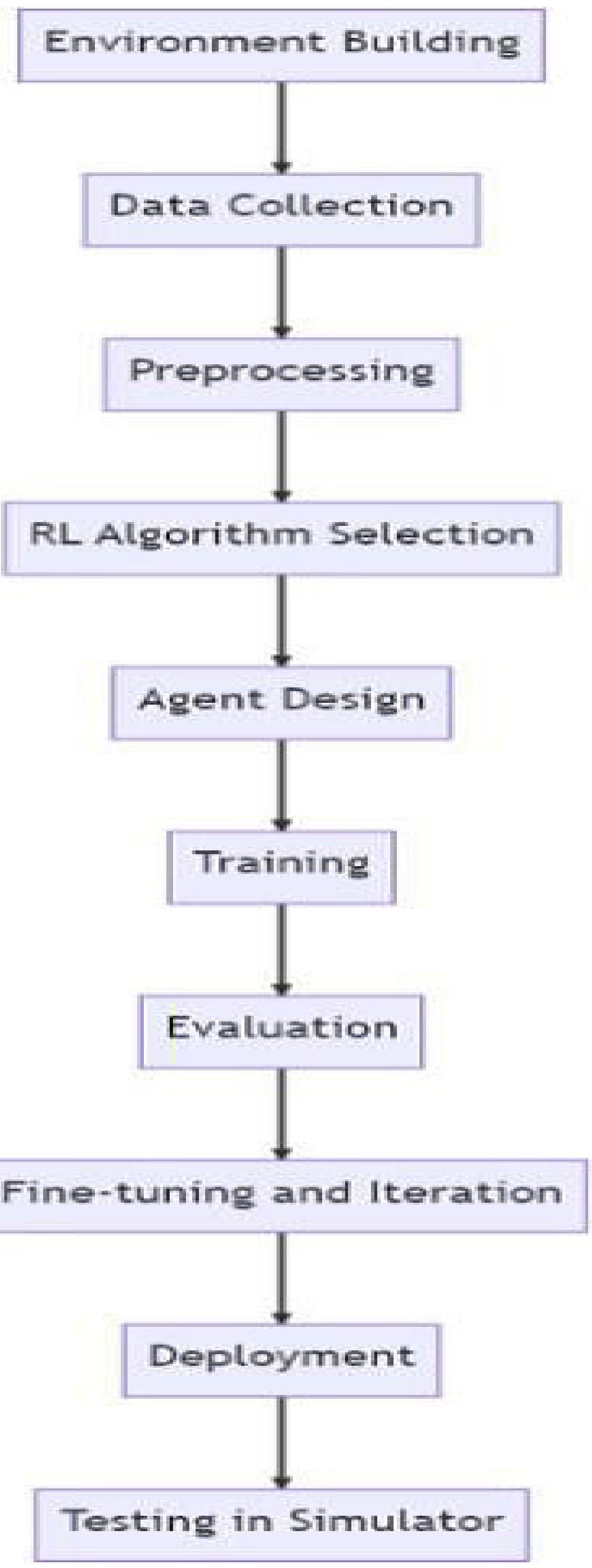


Figure 1: Sequential steps in the implementation of a Reinforcement Learning (RL) system.

During the implementation process, particularly in environment building, we carefully selected between CARLA and AirSim as the simulation environments for training the AI models. This decision played a crucial role in laying the groundwork for subsequent development stages and facilitated the comparative analysis between the two simulators. Throughout our research, we encountered various challenges and obstacles that greatly contributed to our understanding of the strengths and limitations of each simulator.

In the following sections, we have thoroughly explained the details of the methodology employed in building a reinforcement learning AI for autonomous driving. This included selecting a suitable reinforcement learning algorithm, designing neural network models, conducting the training process, and evaluating the performance of the trained AI agent. Through our research, we aimed to provide a comprehensive understanding of the training strategies implemented. It is important to note that we addressed the specific implementation details of CARLA and AirSim simulators separately in subsequent sections, where we highlighted the challenges we faced, and the approaches we took to overcome them.

Algorithmic framework

In our study, we have implemented a neural network architecture that adheres to a Convolutional Neural Network (CNN) structure commonly employed in reinforcement learning tasks. The design of this network plays a crucial role in enabling our AI agent to learn and make informed decisions in the context of autonomous driving.

The input shape of our neural network is concatenated with the specific input shape of the data. This allows us to handle variable-length sequences of input data and effectively capture temporal dependencies.

The network consists of several layers, starting with a permutation layer. The permutation layer rearranges the input dimensions to (2, 3, 1), ensuring compatibility between the input data and the subsequent convolutional layers.

Following the permutation layer, we incorporate three convolutional layers with progressively decreasing filter sizes (32, 64, 64). Each convolutional layer is accompanied by a Rectified Linear Unit (ReLU) activation function. The ReLU activation function introduces non-linearity to the network, allowing it to capture complex patterns and learn hierarchical representations from the input data.

The output of the final convolutional layer is flattened into a one-dimensional vector, transforming the spatial information into a format that can be fed into a fully connected layer. This fully connected layer comprises 512 units and is followed by another ReLU activation, further enabling the network to capture high-level features and relationships in the data.

Lastly, we employ a dense layer with a number of units equivalent to the available action space. This dense layer uses a linear activation function, allowing the network to directly output Q-values for each possible action. By mapping inputs to Q-values, our agent can evaluate the potential outcomes of different actions and make informed decisions in the autonomous driving context.

By implementing this neural network architecture, we enable our AI agent to learn and adapt based on the input data, ultimately empowering it to navigate and make optimal decisions in autonomous driving scenarios (Figure 2) [3].

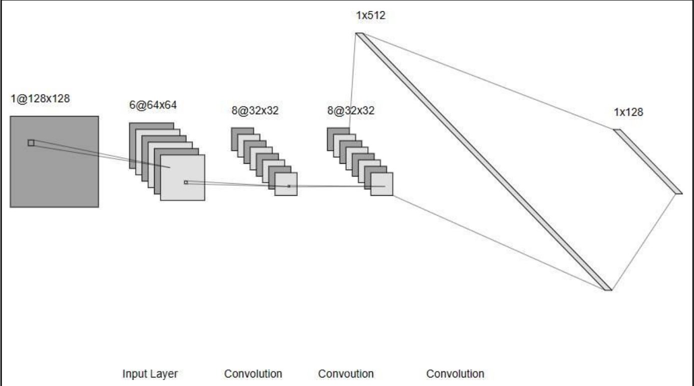


Figure 2: Illustration of the neural network model used in the Study (Note that the last layer 1 × 128 is variable and depends on the available action space of the agent).

We have employed the same neural network architecture and training methods for both the CARLA and AirSim simulators.

Reward engineering for navigation

In our study, we have designed a similar reward function for the environment. The calculation of the reward is based on the car's position relative to the center of the track. We have implemented a method to estimate the perpendicular distance to the line connecting the two closest waypoints, which serves as a proxy for the distance to the center. By utilizing this distance and considering the track width, we have established a reward scheme that incentivizes the agent to stay on the track and penalizes deviations.

Furthermore, we have incorporated termination conditions in the reward function. If the reward falls below zero, indicating a significant deviation from the desired behavior, we terminate the episode. Additionally, if the car approaches the destination within a close proximity (within 5 meters), we consider the episode as complete.

By devising this reward function, we have aimed to guide the agent's learning process towards successful navigation. The formulation aligns with our specific objectives of encouraging the agent to follow the desired path, maintain track adherence, and reach the destination. We believe that these design choices effectively shape the agent's behavior and contribute to improved performance.

It is important to note that the reward function can be further customized and refined based on project requirements or domain expertise. However, the approach we have implemented in the provided code serves as a solid foundation for evaluating the agent's performance and driving capabilities in the context of our study.

AirSim

In our research, we have extensively performed the environment building process for training a reinforcement learning AI model for autonomous driving using AirSim.

This step involved a comprehensive procedure, encompassing the download of the epic games launcher, installation of unreal engine, and meticulous setup of AirSim within the unreal engine environment [4].

However, it is worth noting that setting up AirSim presented certain challenges and required additional effort compared to other simulators. Allow us to provide a more detailed account of the difficulties encountered during the setup process.

Upon downloading the epic games launcher and installing unreal engine, we anticipated a smooth transition into integrating AirSim. However, we soon discovered that configuring AirSim within the unreal engine environment involved a more intricate setup compared to other simulators. The initial challenge we encountered was identifying the appropriate version of unreal engine compatible with the AirSim plugin [5]. Due to the rapid evolution of both unreal engine and AirSim, ensuring compatibility between the two became a crucial consideration (Figure 3).

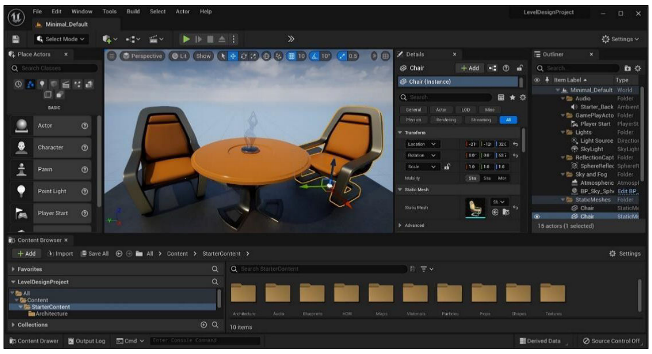


Figure 3: Unreal engine.

Once the compatible version of unreal engine was selected, we proceeded to add the AirSim plugin to our project through the unreal marketplace. While the plugin itself was readily available, we encountered difficulties in navigating the vast array of options within the marketplace and identifying the most suitable plugin version for our research objectives. This process required careful evaluation and consideration to ensure seamless integration and optimal performance.

Furthermore, configuring the AirSim environment to align with our research requirements posed additional challenges. Customizing the simulated driving scenarios, defining the complexity of the environment, and adjusting various simulation settings demanded a deep understanding of both AirSim and unreal engine functionalities. Meticulous attention to detail was required to accurately replicate real-world driving conditions and scenarios, ensuring the authenticity and effectiveness of our AI model's training experience.

The complexity of the AirSim setup process also extended to the importation of assets for visual fidelity and accuracy. Integrating 3D models of buildings, roads, traffic signs, and other objects into our unreal engine project required thorough selection, placement, and alignment to create a realistic and immersive environment. This task demanded both technical expertise and a keen eye for detail to accurately capture the intricacies of real-world driving environments.

Despite the challenges encountered during the setup process of AirSim, we made a strategic decision to focus our training efforts on a simplified racetrack environment. Given that this research primarily aims to compare and analyze the implementation difficulties of different simulators, we opted for a controlled and manageable environment to ensure a fair and accurate evaluation of AirSim. By using a simplified racetrack, we were able to isolate and assess the specific challenges and complexities associated with AirSim setup, enabling us to provide a comprehensive analysis of its implementation process.

While our choice of a simplified racetrack may limit the diversity and complexity of real-world driving scenarios, it allowed us to effectively evaluate the fundamental aspects of AirSim's setup and training procedures. By streamlining the environment, we could focus on the core functionalities of AirSim, such as sensor integration, vehicle dynamics, and reinforcement learning algorithms, without being overwhelmed by extraneous factors. This approach enabled us to conduct a detailed comparison of the challenges and performance characteristics specific to AirSim, offering valuable insights into the simulator's capabilities and limitations.

By leveraging the simplicity of the racetrack environment, we were able to assess the efficiency and effectiveness of AirSim's setup process in a controlled setting. This approach facilitated a targeted analysis of the difficulties encountered during integration, while still providing sufficient context for evaluating the simulator's performance. Our findings and observations from this research can serve as a valuable resource for researchers and practitioners seeking to understand the intricacies of implementing AirSim for autonomous driving AI training (Figure 4).

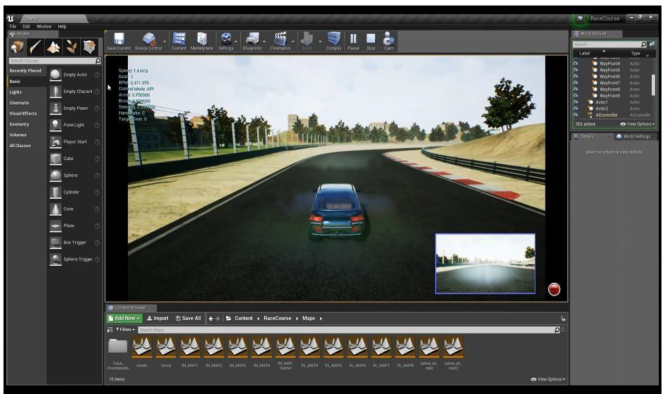


Figure 4: Unreal engine AirSim simulator.

DISCUSSION

CARLA

To implement the CARLA simulator for our autonomous driving research, we followed a stepwise approach. Firstly, we downloaded the CARLA simulator package from the official website [6] to ensure we had the most recent version compatible with our operating system. Next, we thoroughly studied CARLA's extensive documentation, which included installation guides, API references, and tutorials [7]. This allowed us to gain a comprehensive understanding of CARLA's architecture, features, and system requirements.

During the initial setup of the CARLA simulator, we encountered various challenges that required careful attention and troubleshooting. One of the challenges we faced was related to dependencies and library installations. Due to differences in our system environment and dependencies required by CARLA, we encountered missing or incompatible packages during the installation process. This required us to carefully analyze the error messages, search for relevant solutions, and ensure that all the necessary dependencies were installed correctly.

Another challenge we encountered was configuring CARLA's environment variables. CARLA relies on specific paths and configurations to function properly, and any misconfiguration could lead to errors or inconsistencies. We had to carefully set up the environment variables, ensuring that the necessary paths were correctly specified and that the configurations matched our system setup.

As we made progress, we began exploring the diverse features offered by CARLA. This involved experimenting with different maps, vehicle models, and traffic scenarios to create realistic driving environments for our research. We utilized CARLA's extensive APIs to interact with the simulator, enabling us to control vehicles, gather sensor data, and simulate various driving scenarios. This hands-on experimentation allowed us to gain a deeper understanding of CARLA's capabilities and tailor our research to specific use cases.

Moving on to the code implementation phase, we encountered a significant challenge related to compatibility issues with TensorFlow and Keras, which are popular deep learning frameworks. Specifically, we faced problems with CUDA and cuDNN, which are essential components for GPU acceleration. These compatibility issues resulted in errors during installation or while executing our code. To address this, we meticulously verified and ensured that we had the correct versions of TensorFlow, Keras, CUDA, and cuDNN installed, aligning them with CARLA's documentation and our specific algorithm requirements. This involved carefully checking version compatibility, reinstalling libraries, and adjusting configurations as necessary.

Errors

Moving on to the code implementation phase, we encountered a significant challenge related to compatibility issues with TensorFlow and Keras, which are popular deep learning frameworks. Specifically, we faced problems with CUDA and cuDNN, which are essential components for GPU acceleration. These compatibility issues resulted in errors during installation or while executing our code.

One of the errors we encountered was a "ValueError: Calling Model.fit in graph mode is not supported when the Model instance was constructed with eager mode enabled" [8]. This error occurred when trying to train our neural network model using the Model.fit method in TensorFlow. It indicated a conflict between graph mode and eager mode, which are different execution modes in TensorFlow.

Another error we faced was "Failed precondition: Could not find variable Variable. This could mean that the variable has been deleted. In TF1, it can also mean the variable is uninitialized." This error message typically indicates an issue with variable initialization or usage in TensorFlow. It can occur when attempting to access a variable that has not been properly initialized or when the variable has been deleted.

There were more random errors while running the code like "Tensorflow-FailedPreconditionError: Could not find variable dense_24/bias. This could mean that the variable has been deleted" but the main reason for these errors to generate were compatibility issues, such as version mismatches between TensorFlow, Keras, CUDA, and cuDNN. These version discrepancies could result in a lot of various errors and failures during the execution of our code. To overcome these challenges, we meticulously cross-checked the required versions specified by CARLA's documentation, verified the compatibility of all dependencies, and made necessary updates, including reinstalling libraries and adjusting configurations.

By addressing these compatibility issues and troubleshooting the encountered errors, we were able to successfully navigate through the code implementation phase and make progress in our research. These challenges taught us the importance of meticulous version management and compatibility verification when working with deep learning frameworks like TensorFlow and Keras in conjunction with CARLA.

Proposed solution

Based on our experience, we recommend using the following versions for a more seamless implementation process:

CUDA version: cuda_10.0.130_411.31_win10

cuDNN version: cudnn-10.0-windows10-x64-v7.6.0.64

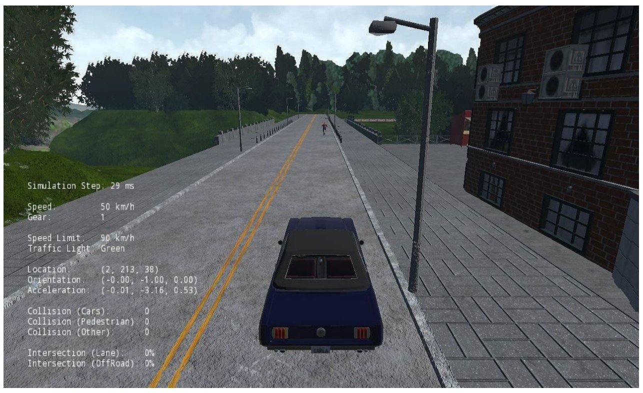
TensorFlow GPU version: tensorflow_gpu-1.1.5

Python version: 3.6

By using these specific versions, researchers and developers can minimize the potential for version mismatch and related errors. These versions have been found to work well together, providing a stable and compatible environment for building autonomous driving projects using CARLA and TensorFlow. Following this recommended version setup can help researchers and developers save time and effort by avoiding unnecessary troubleshooting and compatibility challenges, allowing them to focus more on the core aspects of their projects.

Throughout the process, we relied heavily on CARLA's comprehensive documentation, which provided detailed guidance and troubleshooting steps for various scenarios. Additionally, we actively sought assistance from online forums and developer communities to seek advice from experienced CARLA users who had encountered similar compatibility issues.

By meticulously addressing these challenges and persistently troubleshooting compatibility issues, we were able to successfully overcome initial hurdles and progress in implementing CARLA for our autonomous driving research (Figures 5 and 6).



```
Simulation Step: 29 ms

Speed:           50 km/h
Gear:            1

Speed Limit:     90 km/h
Traffic Light:   Green

Location:        (2, 213, 38)
Orientation:     (-0.00, -1.00, 0.00)
Acceleration:    (-0.01, -3.16, 0.53)

Collision (Cars):          0
Collision (Pedestrian):    0
Collision (Other):         0

Intersection (Lane):       0%
Intersection (OffRoad):    0%
```

Figure 5: CARLA simulator.

Working and validation

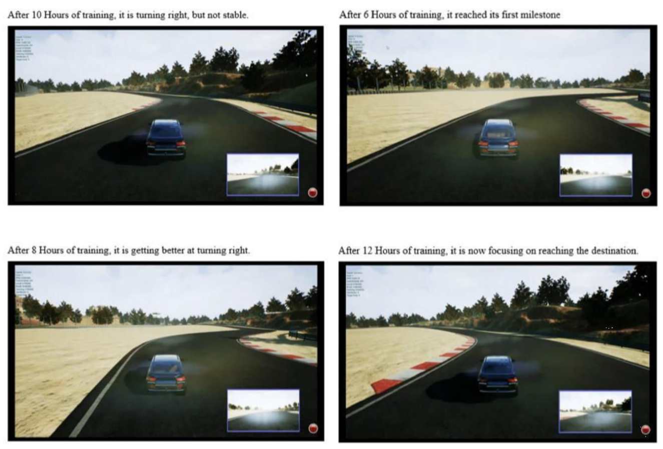

After 10 Hours of training, it is turning right, but not stable.
After 6 Hours of training, it reached its first milestone
After 8 Hours of training, it is getting better at turning right.
After 12 Hours of training, it is now focusing on reaching the destination.

Figure 6: Visual progression of the AI car's training journey captured through a captivating collage of real-time screenshots, reflecting the evolving capabilities and adaptive nature of the learning process.

During the working and validation phase, our focus shifted towards rigorous testing, fine-tuning, and validation of the autonomous driving models implemented using CARLA and AirSim. By subjecting the models to extensive testing scenarios, benchmark evaluations, and meticulous analysis of the outputs, our aim was to ascertain their accuracy, reliability, and ethical behavior. Through this study, we strive to contribute a valuable resource that empowers researchers and practitioners in advancing the field of autonomous driving. Our comprehensive approach ensures that the insights gained from this phase can guide future developments and pave the way for safer and more efficient autonomous driving systems (Figure 6).

CONCLUSION

In this research, we have successfully achieved our primary objective of performing a comprehensive comparative analysis of CARLA and AirSim simulators, focusing on their implementation aspects and identifying associated challenges. Through the implementation and evaluation of various reinforcement learning algorithms on both simulators, we gained valuable insights into their strengths, weaknesses, and the practical considerations involved in their integration.

Throughout our study, we encountered numerous challenges and obstacles, ranging from compatibility issues to configuration complexities. However, we actively addressed these challenges by providing proper solutions and recommendations to overcome them. Notably, we emphasized the significance of using specific versions which have been found to work harmoniously and minimize version mismatch errors.

In conclusion, this research provides a valuable resource for advancing the implementation of CARLA and AirSim simulators using current technology. By highlighting the ease of implementation, challenges faced, and offering practical solutions, we aim to contribute to the development of more robust and efficient autonomous driving systems. Our findings aim to drive further advancements in this exciting field and inspire future research endeavors.

REFERENCES

1. Kiran BR, Sobh I, Talpaert V, Mannion P, Al Sallab AA, Yogamani S, et al. Deep reinforcement learning for autonomous driving: A survey. IEEE Trans Intell Transp Syst. 2021;23(6): 4909-4926.
2. Yu A, Palefsky-Smith R, Bedi R. Deep reinforcement learning for simulated autonomous vehicle control. Course Project Reports: Winter. 2016;2016:1-7.
3. Store. Epic Games. 2023.
4. Github. Build AirSim on Windows. 2023.
5. Carla. CARLA: Open-source simulator for autonomous driving research. CARLA Team. 2024.
6. Github. tensorflow/tensorflow. 2024.
7. NVIDIA Developer. CUDA Toolkit Archive. NVIDIA Corporation, California, United States. 2007.
8. NVIDIA Developer. cuDNN Archive. NVIDIA Corporation, California, United States. 2007.