electronics

MDPI

Article

Deep Reinforcement Learning for a Self-Driving Vehicle Operating Solely on Visual Information

Robertas Audinys, Žygimantas Šlikas, Justas Radkevičius, Mantas Šutas and Armantas Ostreika *

Department of Multimedia Engineering, Faculty of Informatics, Kaunas University of Technology, LT-51368 Kaunas, Lithuania; robertas.audinys@ktu.edu (R.A.); zygimantas.slikas@ktu.edu (Ž.Š.); justas.radkevicius@ktu.edu (J.R.); mantas.sutas@ktu.edu (M.Š.)

* Correspondence: armantas.ostreika@ktu.lt; Tel.: +370-614-21-727

Abstract: This study investigates the application of Vision Transformers (ViTs) in deep reinforcement learning (DRL) for autonomous driving systems that rely solely on visual input. While convolutional neural networks (CNNs) are widely used for visual processing, they have limitations in capturing global patterns and handling complex driving scenarios. To address these challenges, we developed a ViT-based DRL model and evaluated its performance through extensive training in the MetaDrive simulator and testing in the high-fidelity AirSim simulator. Results show that the ViT-based model significantly outperformed CNN baselines in MetaDrive, achieving nearly seven times the average distance traveled and an 87% increase in average speed. In AirSim, the model exhibited superior adaptability to realistic conditions, maintaining stability and safety in visually complex environments. These findings highlight the potential of ViTs to enhance the robustness and reliability of vision-based autonomous systems, offering a transformative approach to safe exploration in diverse driving scenarios.

Keywords: autonomous driving; Vision Transformers (ViT); Deep Reinforcement Learning (DRL); policy learning; metadrive; airsim; safe exploration

check for updates

Academic Editor: Antoni Morell

Received: 17 January 2025
Revised: 13 February 2025
Accepted: 14 February 2025
Published: 20 February 2025

Citation: Audinys, R.; Šlikas, Ž.; Radkevičius, J.; Šutas, M.; Ostreika, A. Deep Reinforcement Learning for a Self-Driving Vehicle Operating Solely on Visual Information. *Electronics* **2025**, *14*, 825. https://doi.org/10.3390/electronics14050825

1. Introduction

Autonomous driving is one of the most important advancements in modern transportation, offering solutions to big problems like road safety, traffic jams, and pollution. Human error causes over 90% of road accidents, leading to a huge loss of life and property [1]. At the same time, cities are becoming more crowded, with more vehicles on the roads, making traffic worse and increasing pollution. Autonomous vehicles aim to solve these problems by removing human error, improving how traffic moves, and cutting fuel use with smart control systems. They also provide better mobility for people who have limited access to transportation, such as the elderly or disabled, improving their quality of life [2].

Even with major progress, building systems that can handle complex and constantly changing environments is still a big challenge. Traditional autonomous driving systems use separate modules for perception, planning, and control [3]. While these systems are easier to understand, they often suffer from issues like passing errors between modules and struggling to adapt to new situations. End-to-end approaches using deep reinforcement learning (DRL) aim to fix these problems by directly linking input data, like images, to driving actions [4,5]. However, most DRL systems rely on Convolutional Neural Networks (CNNs) for processing visual information. CNNs are good at tasks like object detection and recognizing scenes [6], but they struggle to understand large-scale patterns and long-distance relationships in images. This makes it hard for them to handle different driving conditions, such as heavy traffic, poor lighting, or complex road designs [7,8].

This study addresses these challenges by incorporating Vision Transformers (ViTs) into the reinforcement learning framework for autonomous driving. Unlike CNNs, which focus on small, localized image details, ViTs use self-attention to capture bigger patterns and relationships in images. This makes them better at understanding the complexity of visual environments [9–11]. The research tests ViTs for feature extraction in DRL systems to improve their ability to generalize, stay safe, and handle unexpected conditions. Training begins in the MetaDrive simulator, which creates many different driving scenarios [12], and testing is done in the AirSim simulator, which offers a more realistic driving environment to evaluate the system's ability to adapt to new settings [13].

In this research paper we test 2 hypotheses: firstly—whether patch-based information structure from ViTs is suitable for efficient reinforcement learning, secondly—whether ViTs pretrained in self-supervised fashion help to generalize RL to different visual environments.

The organization of the paper is as follows: Section 2 presents a review of related work. Section 3 describes the methodology, including the development of the ViT-based DRL model. Section 4 discusses the experimental setup and results. Finally, Section 5 concludes the paper and outlines future research directions.

2. Literature Review

Autonomous driving is an important innovation with the potential to change transportation by solving major problems like road safety, traffic jams, and pollution. A large part of research in this area focuses on teaching self-driving cars to see and navigate using visual data. Deep reinforcement learning (DRL) has become a promising method for this, as it maps input from sensors directly to driving actions. This approach avoids the issues of traditional systems, which often rely on separate modules that can pass errors between them [1].

Convolutional Neural Networks (CNNs) have been widely used in autonomous driving for processing images, as they are good at identifying patterns and features in visual data. However, CNNs have trouble understanding large-scale relationships in images and adapting to complex scenarios, like busy city traffic or unpredictable environments [1,3,4]. To address these problems, Vision Transformers (ViTs) have been introduced. ViTs use a different approach called self-attention, which helps them understand the big picture in images and handle complex visual scenes better than CNNs [3].

Simulation environments like MetaDrive and AirSim play a key role in developing and testing self-driving systems. These simulators allow researchers to create controlled conditions to evaluate models. MetaDrive provides a simpler environment with tools to quickly generate a variety of road layouts, while AirSim offers more detailed visuals, realistic physics, and challenging scenarios. This combination helps researchers study how well models work across different environments and prepare them for real-world applications [4].

This review looks at the use of CNNs and ViTs in self-driving systems, their role in DRL, and the challenges of using simulators for training. By examining past research, it builds a foundation for comparing performance in simpler and more complex simulation environments. This sets the stage for the current study, which aims to improve autonomous driving technology by combining better visual processing and reinforcement learning techniques.

2.1. Role of Vision in Autonomous Driving

Visual perception is a critical component of autonomous vehicles, providing the data needed for tasks like detecting objects, recognizing lanes, and making driving decisions. Compared to sensors like LiDAR or radar, cameras offer a detailed, high-resolution view of the environment at a lower cost, making them a practical and scalable choice for autonomous systems [1]. This focus on visual data mirrors how humans rely on sight for driving, emphasizing its importance for navigation and safety.

However, vision-based systems face challenges in dynamic environments. Changes in lighting, weather, or visibility, as well as unpredictable traffic and road conditions, can significantly reduce the accuracy of visual perception algorithms. In addition, visual data alone may lack the precise depth information required for tasks like obstacle detection and avoidance. These limitations can be mitigated by incorporating multimodal sensor data, such as LiDAR, radar, and GPS, which can provide complementary information to improve robustness and reliability. Multimodal sensor data integration, including UAV imagery and remote sensing, has been successfully used in precision agriculture for autonomous navigation, demonstrating its potential in improving the robustness of visual perception in dynamic environments [14].

However, while complex systems use multimodal sensors, the processing of visual information must be reliable and accurate, as it is a key part of the systems. On the other hand, cameras are cheaper and simpler than other sensors, so the focus of this work is on the processing of visual information.

Deep learning approaches, particularly convolutional neural networks (CNNs), have advanced the processing of visual data. CNNs are effective at identifying features such as edges, textures, and shapes, which are essential for tasks like object detection and scene segmentation [4]. However, CNNs have limitations in understanding long-range spatial relationships and global context, which are critical for navigating complex environments [9]. These shortcomings have led to growing interest in Vision Transformers (ViTs). Unlike CNNs, ViTs leverage self-attention mechanisms to capture global patterns and relationships in visual data, enabling them to handle interactions between objects and dynamic elements in the environment more effectively [9,13].

Visual data's role goes beyond perception, it is also central to decision-making. End-to-end learning systems, which directly connect visual inputs to driving actions, aim to simplify the process and reduce errors caused by multiple pipeline stages. These systems must combine visual perception with decision-making to drive safely and efficiently [15]. Training and testing such systems in real-world settings is challenging, so simulation platforms like MetaDrive and AirSim are essential. These tools allow models to experience a wide range of scenarios, testing their ability to adapt and generalize across environments [10,12]. Simulation also provides a controlled space to train models for reliable performance in real-world conditions [16].

2.2. CNNs in Autonomous Driving

Convolutional Neural Networks (CNNs) have played a pivotal role in advancing the capabilities of autonomous vehicles by enabling them to interpret complex visual data with high accuracy. As one of the earliest and most widely used architectures for visual perception, CNNs remain a benchmark for performance in many autonomous driving systems.

CNNs are characterized by a hierarchical structure consisting of convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply filters to detect spatial features such as edges, corners, and textures. Pooling layers reduce the spatial dimensions of data, preserving critical information while optimizing computational efficiency.

Fully connected layers combine learned features to generate predictions [6,17]. This design enables CNNs to process raw visual inputs and extract meaningful patterns, making them essential for tasks like object detection, lane detection, and traffic sign recognition:

1. Feature Extraction: CNNs excel at extracting local spatial features, enabling them to detect objects, identify lane markings, and recognize traffic signs with high accuracy [5,7].
2. Translation Invariance: The architecture of CNNs ensures robust feature detection regardless of object position within an image, which is vital for autonomous vehicles operating in dynamic environments [17,18].
3. Scalability: CNNs can handle high-resolution input data, making them suitable for detailed visual processing in urban and highway settings [6,8].

Despite their strengths, CNNs face notable challenges:

1. Limited Global Context Understanding: CNNs focus on local patterns, making it difficult to capture long-range dependencies and holistic spatial relationships [6,19].
2. Computational Intensity: Deep CNN architectures require significant computational resources, posing challenges for real-time processing in autonomous vehicles [7,17].
3. Vulnerability to Adversarial Attacks: CNNs are susceptible to small, intentional perturbations in input data, which can lead to erroneous predictions—a critical safety concern for autonomous driving [20].

Despite the emergence of Vision Transformers (ViTs) and other advanced architectures, CNNs remain a foundational component of autonomous driving research. Their established performance and extensive optimization make them an ideal baseline for evaluating newer models. Moreover, hybrid models combining CNNs and other architectures, such as attention mechanisms, have shown promise in enhancing visual perception for autonomous systems [1,8,19]. Benchmarks based on CNNs provide a robust reference point to assess advancements in performance, efficiency, and generalization capabilities [6,20].

CNNs have significantly contributed to the progress of vision-based autonomous driving systems. Their ability to extract features, process high-resolution data, and support end-to-end learning frameworks has been instrumental in enabling autonomous vehicles to navigate complex environments. While newer architectures like ViTs address some of CNNs' limitations, understanding and leveraging CNNs' strengths ensures they remain relevant for developing and evaluating autonomous systems [5,8,20].

2.3. Vision Transformers (ViTs) for Visual Perception

Vision Transformers (ViTs) represent a significant advancement in visual perception, offering substantial improvements over Convolutional Neural Networks (CNNs) in tasks critical to autonomous driving.

ViTs leverage the Transformer architecture, originally designed for natural language processing, to process visual data. Unlike CNNs, which rely on localized feature extraction, ViTs divide an image into fixed-size patches, embed these patches linearly, and add positional encodings to retain spatial information. These embeddings are then processed through multiple layers of self-attention mechanisms, allowing the model to analyze the entire image holistically. This structure enables ViTs to capture global dependencies and complex spatial relationships, overcoming the limitations of CNNs in handling long-range interactions [9–11].

ViTs have demonstrated their capabilities across a range of vision tasks:

1. Image Classification: ViTs have achieved state-of-the-art performance on large-scale datasets such as ImageNet, outperforming CNNs when sufficient training data is available [9,10,21].

2. Object Detection and Semantic Segmentation: With their global attention mechanism, ViTs improve object detection and segmentation tasks, enabling more accurate and coherent scene understanding [4,9].
3. 3D Perception for Autonomous Driving: ViTs have been successfully applied to 3D data, such as LiDAR point clouds, to enhance semantic segmentation and improve environmental understanding [19,22].

ViTs provide several benefits that address the limitations of CNNs in autonomous driving applications:

1. Global Context Modeling: By leveraging self-attention, ViTs capture long-range spatial dependencies, essential for interpreting complex driving environments [9,10,22].
2. Adaptability to Multimodal Inputs: ViTs can seamlessly integrate diverse input modalities, such as images, radar, and LiDAR data, making them ideal for sensor fusion in autonomous systems [10,22].
3. Scalability with Data: ViTs excel in performance when trained on large datasets, which aligns well with the extensive data typically available in autonomous driving applications [9,21].

ViTs address critical challenges in autonomous driving, such as understanding complex environments, processing multimodal sensor data, and generalizing across diverse driving scenarios. These features make them a strong complement to CNNs, which often serve as baselines for comparison. While CNNs are effective at local feature extraction, they struggle with modeling global spatial dependencies, which are critical for decision-making in autonomous driving. Recent studies [9–11] have demonstrated that Vision Transformers (ViTs) address this limitation by utilizing self-attention mechanisms to capture global context across an image. Our study extends these findings by applying ViTs in reinforcement learning for autonomous driving tasks, showing a significant improvement over CNN-based models.

The integration of Vision Transformers in visual perception for autonomous driving marks a transformative step in the field. By addressing the limitations of CNNs and enabling advanced capabilities such as multimodal integration and global context awareness, ViTs are well-positioned to play a pivotal role in the next generation of autonomous vehicle technologies. Future research will focus on improving their efficiency and adapting them for real-time applications [9,21,22].

2.4. Simulators in Autonomous Driving Research

Simulators are integral to autonomous driving research, providing safe, scalable, and cost-effective environments for testing and validation. They enable researchers to evaluate algorithms across diverse conditions without the risks associated with real-world deployment. MetaDrive and AirSim are two prominent simulators that cater to distinct aspects of autonomous driving research.

MetaDrive is an open-source simulator designed to generate procedurally diverse driving scenarios. Its primary strength lies in its ability to create infinite variations of road layouts, traffic conditions, and environmental complexities. This diversity supports reinforcement learning (RL) research by exposing models to varied conditions, fostering better generalization across unseen environments. MetaDrive is computationally efficient, running on standard hardware while supporting high-throughput experiments [12]. Its emphasis on scenario diversity makes it particularly well-suited for studying policy learning in dynamic and procedurally generated environments.

Developed by Microsoft, AirSim offers high-fidelity visual and physical simulations for both ground and aerial vehicles. Built on the Unreal Engine, it provides realistic physics, photorealistic environments, and extensive support for various sensors such as

LiDAR, radar, and cameras. AirSim's modular design allows for significant customization, making it a preferred choice for tasks that demand detailed simulations, such as perception, planning, and control in complex urban environments. Its realism facilitates the evaluation of sensor fusion strategies and simulation-to-reality transfer techniques, where simulated policies are deployed in real-world scenarios [13].

MetaDrive and AirSim serve different but complementary purposes in autonomous driving research:

- Scenario Diversity vs. Realism: MetaDrive excels at generating diverse scenarios efficiently, aiding generalization studies. AirSim's high-fidelity simulations are invaluable for testing algorithms under lifelike conditions [12,13].
- Hardware Requirements: MetaDrive is lightweight and runs smoothly on standard PCs, enabling large-scale RL experiments. In contrast, AirSim's detailed simulations are computationally intensive, necessitating high-performance hardware [13].
- Customization: While AirSim allows researchers to design specific sensors and vehicle configurations, MetaDrive focuses on procedurally generating diverse scenarios with minimal setup [12].

Generalization is a key challenge in autonomous driving, requiring models to perform reliably across diverse, unseen environments. MetaDrive addresses this challenge by exposing models to a wide range of procedurally generated scenarios during training. This process improves their adaptability and robustness [12]. Meanwhile, AirSim's high-fidelity simulations allow researchers to test the generalization of trained models under realistic environmental conditions, bridging the gap between simulation and real-world applications.

MetaDrive and AirSim are indispensable tools in autonomous driving research. MetaDrive's focus on efficiency and diversity makes it ideal for reinforcement learning and generalization studies, while AirSim's realism supports detailed perception and control tasks. Together, they enable comprehensive evaluation of autonomous driving systems, addressing both theoretical and practical challenges.

2.5. Comparative Studies and Research Gaps

Comparative studies of Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) have highlighted the unique advantages and limitations of each architecture in autonomous driving. These analyses also reveal critical research gaps in cross-environment generalization, where models must adapt to unseen scenarios and dynamic conditions.

CNNs have long been a cornerstone in computer vision due to their ability to extract hierarchical spatial features efficiently. However, their reliance on local receptive fields limits their capacity to model long-range dependencies and global context, which are crucial for understanding complex driving environments [9,10]. ViTs, in contrast, utilize self-attention mechanisms to capture global context across an image, enabling superior performance in scenarios requiring holistic scene understanding [23]. ViTs also demonstrate enhanced adaptability to diverse data modalities, such as LiDAR and radar, further extending their utility in autonomous systems [9,24].

Despite their strengths, ViTs require extensive training data and computational resources to achieve optimal performance. CNNs, on the other hand, remain efficient and effective for tasks with constrained data or simpler visual requirements. Studies comparing these architectures, such as those by Raghu et al. (2021) [23] and Leite et al. (2022) [25], emphasize that the choice between CNNs and ViTs often depends on the task and dataset size.

Generalization across environments remains a significant challenge for autonomous driving systems. Current research highlights the difficulty of training models that can reliably adapt to unseen scenarios involving variable lighting, weather, and traffic conditions.

While both CNNs and ViTs have shown promise, neither architecture fully addresses this gap [25].

Simulators like MetaDrive and AirSim have been instrumental in studying generalization by enabling the creation of diverse training scenarios [12]. However, there is a need for more robust evaluation frameworks that test models under extreme and unpredictable conditions. Moreover, recent studies, such as Alijani et al. (2022) [26] and Yue et al. (2021) [27], emphasize the importance of domain adaptation and randomization techniques for bridging the gap between simulation and reality. These methods enhance model robustness but require further exploration to achieve consistent results in real-world applications.

This study addresses these gaps by comparing CNNs and ViTs in the context of cross-environment generalization. By leveraging MetaDrive for diverse scenario generation and AirSim for high-fidelity testing, it evaluates the strengths and weaknesses of both architectures. The integration of reinforcement learning further examines how these models adapt to dynamic and procedurally generated environments. This work contributes to a deeper understanding of architectural trade-offs and provides a pathway for developing more generalizable autonomous driving systems.

2.6. Future Directions

The findings of this study point to several promising avenues for advancing autonomous driving systems. By comparing the strengths and limitations of CNNs and ViTs across various environments, key opportunities for further research have been identified.

One important direction is using multi-modal data, combining inputs like camera feeds, LiDAR, and radar with ViTs. This approach could make systems more adaptable and reliable by providing richer and more diverse information for decision-making [28]. Combining data from multiple sources could help overcome the current limitations of CNNs and ViTs, improving their ability to handle complex environments. The integration of self-driving systems with real-time data fusion, as demonstrated in precision agriculture [29], suggests promising future directions for combining vision perception models with sensor fusion techniques to enhance vehicle adaptability to diverse environments.

Although ViTs show strong potential, their high computational needs make real-time use a challenge. Research into more efficient ViT designs, such as those proposed by Pan et al. (2022), could help reduce delays and energy use, making them more suitable for use in self-driving cars [30].

Simulation platforms like MetaDrive and AirSim are great tools for testing in controlled environments, but the difference between simulation and real-world conditions remains an issue. Techniques like domain randomization and pyramid consistency could be further improved to help models perform just as well in real-world settings as they do in simulations [31].

The use of ViTs in autonomous systems has the potential to change how visual data is processed. Additionally, methods like masked autoencoders for unsupervised learning could reduce the need for large amounts of labelled data, making training more efficient [32]. By addressing these challenges, future research can help create safer and more flexible autonomous driving systems that can handle the complexities of real-world environments.

2.7. Conclusion of the Literature Review

The examination of Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) in autonomous driving highlights the unique strengths and limitations of each architecture. CNNs are effective at processing local spatial features, offering computational efficiency and reliability in controlled, well-structured scenarios. On the other hand, ViTs utilize self-attention mechanisms to understand global contextual information, making them better suited for handling complex and dynamic environments. However, ViTs often require significant computational resources and large datasets for effective training [9,10,30].

A key challenge identified is the difficulty these models face in generalizing across different environments. Both CNNs and ViTs tend to experience performance drops when exposed to unfamiliar conditions, such as changes in weather, lighting, or traffic. This limitation highlights the importance of robust training techniques and evaluation methods that include diverse and varied driving scenarios [12,25].

Simulation platforms like MetaDrive and AirSim play a crucial role in overcoming these challenges by providing controlled environments for training and testing. These tools enable researchers to create a wide range of scenarios, helping evaluate the adaptability and generalization of autonomous driving [12,31].

Building on these findings, our approach integrates CNN and ViT architectures within simulated environments to compare their performance under diverse driving conditions. By combining the strengths of these models and utilizing advanced simulation tools, this study aims to develop a more robust autonomous driving system capable of adapting to the complexities of real-world environments [9,10,31].

3. Materials and Methods

This section presents the methodologies and resources used to develop and evaluate the proposed reinforcement learning framework for autonomous driving based on Vision Transformers (ViTs). The approach substitutes CNNs with ViTs for visual feature extraction to enhance the robustness and generalization of the policy.

Training is conducted using the MetaDrive simulator in procedurally generated environments, while testing is carried out in realistic scenarios using the AirSim simulator. Key performance metrics, including success rate, collision rate, and average distance travelled, are used to measure the model's effectiveness compared to the CNN baseline.

The subsections provide detailed descriptions of the system architecture, hardware and software setup, training protocols, and evaluation criteria.

3.1. System Architecture Overview

The system architecture combines simulation tools, visual data processing, and decision-making into one streamlined reinforcement learning setup. At the input stage, the agent receives image data from a virtual car's front-facing camera, usually as a series of 4 RGB frames. These frames show the road ahead, nearby cars, and current weather or lighting conditions (see Figure 1 for an example of a road with cars).

Next, the image processing part converts these detailed raw images into simpler, meaningful features. In early tests, a convolutional neural network (CNN) was used to pull out spatial details from the images. Later, a Vision Transformer (ViT) was used to get even better, more comprehensive features. These features focus on important details and remove unnecessary ones, creating a compact summary of the scene. The complete system design is illustrated in Figure 2.

Once the visual features are prepared, they go to the decision-making system. This system uses fully connected networks to handle policy (choosing actions) and value functions (estimating the benefits of actions). The agent uses a reinforcement learning algorithm called Proximal Policy Optimization (PPO) to connect visual data to car control actions like steering, accelerating, and braking. These actions are sent back into the simulation, creating a feedback loop between perception and action.

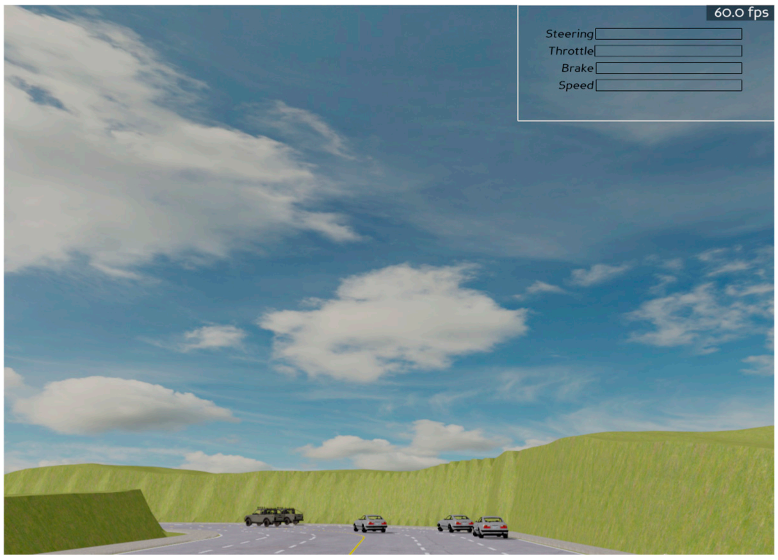


Figure 1. Agent's visual context.

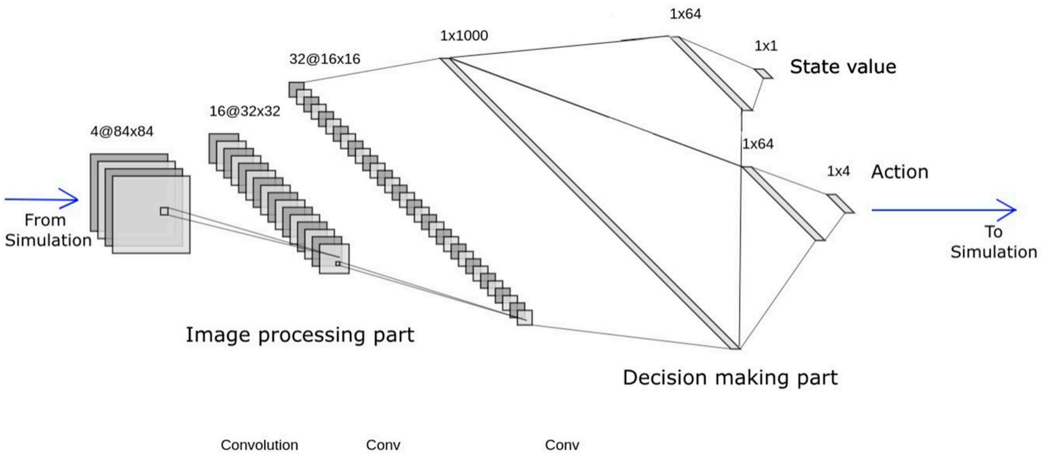


Figure 2. General model of commonly used neural network structure. Our experiments explore different alternatives for image processing.

Training happens step-by-step. The agent drives in the simulation, learns from experience, and updates its decision-making rules to get better over time. Different maps are used to make sure the agent doesn't just memorize one specific road but learns how to handle varied conditions. By testing the system multiple times with random variations, the agent's ability to adapt to new scenarios is confirmed. In short, the architecture combines simulated vision, advanced visual processing, smart decision-making, and repeated training to create adaptable and effective self-driving skills.

3.2. *Hardware Infrastructure*

All experiments were carried out on a workstation with an Nvidia RTX 4060 GPU that has 8 GB of video memory. This GPU provided enough power to train large neural network models, including both CNNs and Vision Transformers (ViTs). The system's 32 GB of main memory efficiently handled simulation data, model settings, and intermediate results, even when running multiple simulations at the same time. A 200 GB solid-state drive (SSD) was used for storing simulation environments, model checkpoints, training records, and evaluation outcomes.

3.3. Software Environment

The research used a carefully designed software setup to ensure the experiments were stable, repeatable, and compatible with the overall goals. This setup made it possible to develop, train, and test reinforcement learning methods for autonomous driving.

Operating System: All experiments were run on the Windows 11 operating system. Windows 11 was chosen because it works well with both the MetaDrive and AirSim simulators, making it easier to combine these tools with machine learning frameworks.

Programming Language and Machine Learning Frameworks: Python 3.11 was the main programming language because it has many useful libraries for machine learning and reinforcement learning. The training system used PyTorch 2.4.0 to build Vision Transformers (ViTs) and manage deep learning tasks like feature extraction and training. Stable Baselines3 2.3.2 was used to implement Proximal Policy Optimization (PPO), a reinforcement learning method that works well with continuous actions like steering or braking. To process data and handle numbers efficiently, the system used NumPy and pandas. OpenCV was included for image processing and to make sure the visual input data from the simulators was consistent and high-quality.

To keep track of all the hyperparameters, metrics, and results, the experiments used MLflow. This made it easier to manage and reproduce the experiments. All the training data, model versions, logs, and results were stored on a 2 TB NVMe SSD, allowing quick access for analysis.

Simulation Environments: For simulation environments, MetaDrive version 0.4.2.3 was used because it is lightweight and can quickly create different road layouts and traffic situations. Its low computational demands allowed for fast policy training and frequent testing. AirSim, on the other hand, is based on Unreal Engine and provides a highly realistic environment with accurate physics and detailed visuals. This made it ideal for testing how well the trained policies could adapt to real-world-like challenges.

Justifications for Software Choices: The combination of these tools and software was intentional. MetaDrive made it easy to quickly train policies in varied environments, while AirSim provided a realistic way to test their reliability. Similarly, PyTorch and Stable Baselines3 were chosen because they are reliable, widely used in the research community, and well-documented. Together, these tools helped create a balance between efficiency and realism, ensuring the research produced trustworthy and adaptable autonomous driving models.

3.4. Performance Metrics

Evaluating the proposed Vision Transformer (ViT)-based deep reinforcement learning (DRL) framework involves analyzing its performance through various metrics. These metrics are essential for comparing the effectiveness of vision processing components (CNN vs. ViT) and assessing performance across two simulators: a lightweight, procedurally generated simulator with simple visual contexts and a high-fidelity simulator with complex, realistic visual representations.

In this study, robustness refers to the model's ability to perform consistently across different environments and conditions, demonstrating adaptability to unseen road layouts and traffic densities. This is assessed through metrics such as average distance traveled, average episode length, and transferability from MetaDrive to AirSim. A more robust model should perform well across diverse environments without requiring extensive retraining.

Safety is defined as the ability of the autonomous system to minimize hazardous situations, particularly collisions, erratic lane departures, and abrupt braking. We quantify safety using collision rate per episode and lane adherence. A lower collision rate and smoother control inputs indicate a safer driving policy.

The following key metrics are used to evaluate the models.

The Success Rate quantifies the percentage of episodes where the agent successfully completes its task, such as reaching a target destination without collisions or violations. It is calculated as follows:

$$Success\ Rate = \frac{Num\ of\ successful\ episodes}{Total\ episodes} \tag{1}$$

Success Rate provides a high-level measure of the system's reliability. In a lightweight simulator, it indicates basic navigational robustness, while in a realistic simulator, it reflects the system's ability to handle more intricate and visually complex scenarios. This metric is especially useful in simulation environment comparisons, as a higher Success Rate in the realistic simulator would demonstrate ViT's superior ability to handle detailed visual inputs [33].

The Average Distance Travelled measures the total distance covered by the agent during an episode. It is computed as:

$$Avg\ dist\ travelled = \frac{\sum_{i=1}^{N} d_i}{N} \tag{2}$$

where d_i is the distance covered in episode i, and N is the total number of episodes.

This metric evaluates how effectively the agent navigates without collisions. It reflects overall progress and navigation efficiency and are useful for detecting early failures (in short distances).

The Collision Rate per Episode measures the frequency of collisions within an episode. It is calculated as follows:

$$Collision\ Rate = \frac{Num\ of\ collisions\ in\ an\ episode}{Total\ episodes} \tag{3}$$

This metric is critical for safety evaluation. In simulators, it tests basic obstacle avoidance or challenges the system's ability to manage more dynamic elements, such as other vehicles and pedestrians. The advantage of this metric is that it directly measures safety performance and helps to identify problematic scenarios, but this metric does not account for collision severity.

The Average Steps per Episode tracks the number of timesteps required to complete an episode.

$$Avg\ steps\ per\ episode = \frac{\sum_{i=1}^{N} S_i}{N} \tag{4}$$

where S_i is the number of steps in episode i. This metric reflects the time efficiency of the agent. It demonstrates the system's ability to adapt to dynamic and complex situations. A model with fewer steps per episode demonstrates more effective policy decisions [34,35]. The advantage of this metric is that it shows efficiency of navigation and helps identify unnecessary delays, but it does not directly account for safety or smoothness of driving.

These metrics were chosen as they align with standard practices in reinforcement learning and autonomous driving research, offering a comprehensive evaluation of safety, reliability, and efficiency [34].

3.5. Reinforcement Learning Framework

Our experiments use Proximal Policy Optimization algorithm to train simulation agents. This algorithm belongs to on-policy type and relies on multiple parallel actors for data collection and can be characterized by single loss function (5), where its terms are expressed in Equations (6) and (7).

$$L^{CLIP+VF+S}(\theta) = E\left[L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S(s_t|\pi_\theta)\right] \quad (5)$$

$$L^{VF}(\theta) = \left(V_\theta(s_t) - V^{targ}\right)^2 \quad (6)$$

$$L^{CLIP}(\theta) = E\left[min\left(\frac{\{\pi_\theta(a_t|s_t)\}}{\pi_{\theta_{old}}(a_t|s_t)}A_t,\ clip\left(\frac{\{\pi_\theta(a_t|s_t)\}}{\pi_{\theta_{old}}(a_t|s_t)},\ 1-\varepsilon,\ 1+\varepsilon\right)A_t\right)\right] \quad (7)$$

where V^{targ} and A are state value target and advantage values given by GAE (Generalized advantage estimator), c_1 and c_2 are hyperparameters, V and π are neural networks predicted state value and action probability values, S function is an entropy of policy distribution, which encourages exploration at the start of the training. The main goal of this function is to minimize agent's policy change in one step, which in turn avoids big changes in collected data and optimized target and saves from instability. The implementation of this algorithm was provided by StableBaselines3 library.

We use SafeMetaDriveEnv environment from MetaDrive as an agent's training environment with mostly default reward functions. Changed default values are shown in Table 1 and are taken into account when calculating when the final reward. The default reward function consist of three main parts [12].

$$R = c_1 R_{disp} + c_2 R_{speed} + R_{term} \quad (8)$$

Table 1. MetaDrive simulator environment's parameters.

Parameter	Value
Speed reward	0.0
Use lateral reward	False
Out of road penalty	30 score
Crash object done	False
Crash vehicle done	False
Out of road done	True
On continuous line done	False
Traffic density	0.05 ratio
Accident probability	0.8
Map	"CCCCCCCCCCCCCCC"

The driving reward (R_{disp}) encourages the agent to move forward towards the destination by considering the difference in longitudinal coordinates of the vehicle between two consecutive time steps. It is measured in meters (m). The speed reward (R_{speed}) incentivizes the agent to drive faster, using the vehicle's current velocity (v) relative to a maximum velocity of 80 km/h. The termination reward (R_{term}) is a sparse reward given at the end of an episode, accounting for events such as successful completion, going off-road, or crashing into objects or vehicles [12]. The coefficients c_1 and c_2 are weights that determine the relative importance of the driving and speed rewards.

The aim of changing these default environment parameters was to give more freedom for agent to learn and increase the training speed.

Both ViT and CNN vision processing architectures were combined with the same PPO reinforcement learning algorithm. The Convolutional Neural Network (ResNet34) and Vision Transformer (DinoV2) models were used as pretrained feature extractors, and their weights remained frozen during training. The only components trained were our additional neural network layers (as seen in images Figures 3 and 4), which learned the policy mapping from vision features to control actions.

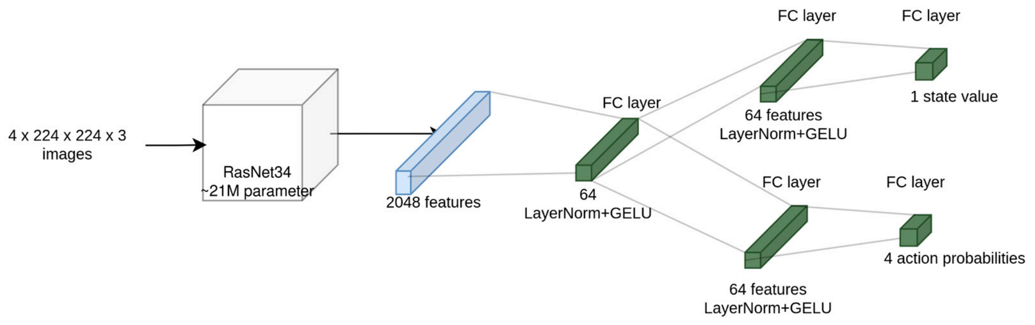


Figure 3. Our neural network with ResNet34 as feature extractor.

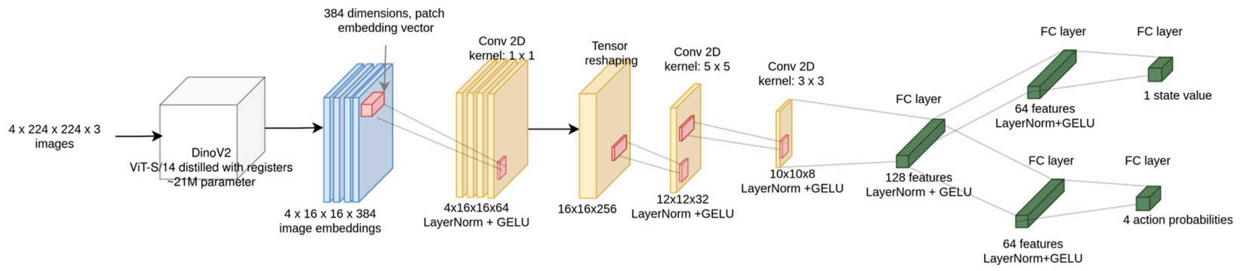


Figure 4. Our neural network with DinoV2 21M as visual feature extractor.

Simulation Environment:

- Frame Size: 224 $\times$ 224 pixels.
- Stacked Frames: 4 consecutive frames as input.
- Traffic Density: 0.1 (adjusted per training stage).
- Accident Probability: 0.8.
- Training Map: "CCCCCCCCCCCCCC" (Curved road segments).
- Number of Scenarios: 200.
- Random Seed: 42 (for reproducibility).

After multiple preliminary trials, we found that the hyperparameter values listed in Table 2 yield the most promising results. We chose a balance between obtained results ("episode reward mean"—last column), training stability and speed. If it's possible to ensure stable convergence—bigger learning rate should generally provide faster learning. For this reason, we prioritized hyperparameter configurations with value $4 \cdot 10^{-4}$. One experiment that used clipping values decay showed particularly promising results with episode reward mean twice the size of other trials, but some instabilities in learning lead it to diverge in just a couple final iterations. Despite this, we have chosen to conduct further experiments using plipping decays while keeping eye on potential instabilities. Learning rate also used linear decay to enable smaller adjustments and prevent radical changes in policy. The PPO clip range of 0.2 gives algorithm more freedom to change its policy, compared to smaller values. A higher value function clip (10) allows sufficient flexibility for value estimation. An entropy coefficient of 0.01 maintains exploration without excessive randomness, we have observed that this value gave better results. Gradient clipping at 1 prevents instability from exploding gradients.

The batch size of $1024 \cdot 3$ was based on memory consumption and RAM available and a minibatch size of 256 was picked to balance update count and gradient stability. Three training epochs per batch prevent overfitting while refining policy updates. A discount factor ($\gamma = 0.99$) promotes long-term reward optimization, and GAE ($\lambda = 0.95$) balances bias and variance in advantage estimation (default values from PPO, rarely changed). Results with different hyperparameters can be seen in Figure A1.

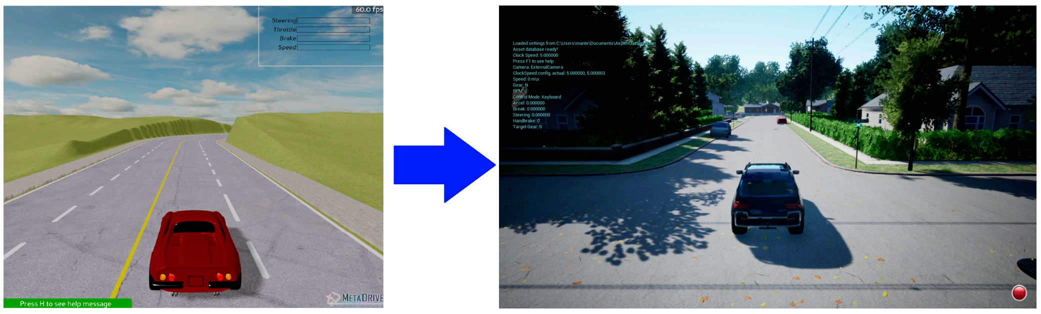
Table 2. Used PPO algorithm hyperparameters.

Parameter	Value
Simulations count	3
Learning rate	$4 \cdot 10^{-4}$
Clip range	0.2
Clip range value function	10
Entropy coefficient	0.01
Value function coefficient	1
Max gradient norm	1
Batch size	$1024 \cdot 3$
Minibatch size	256
N epochs	3
Discount γ	0.99
GAE λ	0.95

Learning rate, policy clip and value function clip parameters also use linear decay over training span.

3.6. Vision Processing: Transition from CNN to ViT

Vision transformers based neural net architectures already show better results than their convolutional counterparts partly because of lack of bias that comes with hardcoded convolution filter sizes, but ViTs aren't much explored in RL scene as image feature extractors. Our work explores possible advantages of using transformer-based image feature extraction networks instead of convolutional. For this comparison, we selected two similarly sized (~21M parameters) pretrained networks with the smallest possible number of parameters to optimize resource usage and training time. We then adapted their outputs by applying additional compression before producing state value and action probability outputs in the final layer. Images (see: Figures 3 and 4) show used neural net architectures. As CNN representation we chose ResNet34 because ResNet models are widely used in literature and one of the best for this task. For vision transformers model we chose to use Meta's DinoV2 21M parameter model with registers modification. It was pretrained using self-supervised learning on large dataset and shows good results on various styles of images with ability to extract depth and semantic information with only a few additional layers. Additionally, DinoV2's authors show that it is possible to find matching features between different style images, using principal component analysis. Knowing this, we assume that DinoV2 based model would be able to trivially adapt to different visual environments, for example: using different complexity simulators (see Figure 5) or sim2real.



```
Loaded settings from C:\Users\mante\Documents\AirSim\settings.json
Asset database ready!
Clock Speed: 5.000000
Press F1 to see help
Camera: ExternalCamera
ClockSpeed config, actual: 5.000000, 5.000003
Speed: 0 m\s
Gear: N
RPM: 0
Control Mode: Keyboard
Accel: 0.000000
Break: 0.000000
Steering: 0.000000
Handbrake: 0
Target Gear: 0
```

Figure 5. Example images from MetaDrive (**left**) and AirSim (**right**) simulators. We attempt to train neural network on MetaDrive simulator and apply it to control agent in AirSim.

3.7. Training and Evaluation Process

This research consists of 2 parts—training CNN based and ViT based PPO algorithms and evaluating trained ViT networks capabilities in different simulators.

For the first part, we trained a baseline neural-network based on most common methods found in literature, which is ResNet based visual feature extraction. After that, we aimed to isolate and directly compare convolutional and transformer network architectures for vision-based reinforcement learning, by training DivoV2 based model with as little additional modifications as possible. The training is done only in the Metadrive simulator using reinforcement learning with no external datasets or additional inputs. During training the observations received from the simulator after each action are passed to the agent which outputs the next action.

In the second part, the trained ViT-based model was tested in the AirSim simulator. With its more realistic physics and detailed visual environments, AirSim offered a better representation of real-world conditions. This phase assessed how well the ViT model could adapt, remain robust, and generalize to more complex and challenging scenarios.

3.7.1. Step 1: CNN and ViT Comparison Experiments

Environment Setup

In order to isolate all variables and fairly compare only network architectures, we used identical environments for both steps, CNN and ViT. Road is made from randomly assembled, curved highway sections with occasional other cars and road obstacles. Road example can be seen in (Figures 5 and 6) and hyperparameters used for map generation in (Table 1). This configuration represents driving and keeping lanes on the highway in the real world, and it can be considered the simplest case of autonomous driving. This research focuses on different vision models integration and their ability to perform in different environments, for this reason we have chosen to train agents in the simplest map type to minimize complexity that comes from broader reinforcement learning tasks. Adding more types of roads would require modifications of our neural networks—incorporating turning signals. In our case simulations start at the beginning of the curved road and in the correct direction. A vehicle's goal is to drive as far as possible or until the road ends while avoiding collisions with other vehicles and occasional obstacles and staying in its own lane. In all episodes the agent drives the same car.

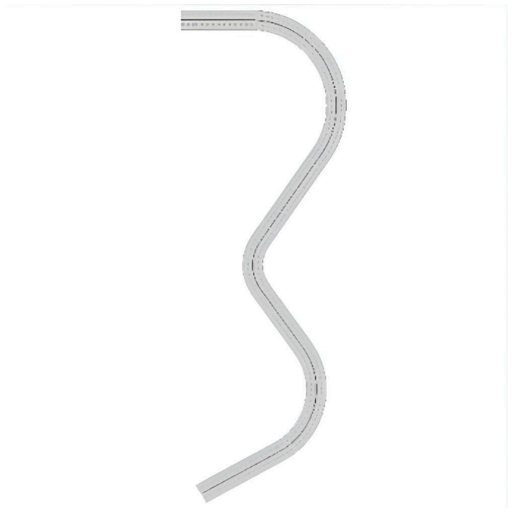


Figure 6. Examples of C road section combinations in MetaDrive simulator.

Policy Initialization and Feature Extraction

CNN neural network architecture is showed in Figure 3. For a ResNet34 part, we use pretrained and frozen weights from PyTorch library. Simulator return RGB images of size 224×244 pixels, to be able to infer current movement information from view, we save last 4 frames in a circular buffer and pass them together to neural net at each step. Also as required by ResNet, we normalize and scale each color channel by different factor.

ViT neural network architecture is shown in Figure 4. For a vision transformer we use pretrained DinoV2 20M parameter model with registers modification, weights are frozen and not trained. Similarly, as with CNN, we save last 4 frames and give them to the neural net at each iteration. From DinoV2 results we use just image patch embedding values (localized features). Patches represent 14×14 pixel squares which in our care translates to 16×16 patches for image, each having 384 length vector embeddings. Because of high dimensionality, we further compress embeddings using CNNs as shown in the diagram (Figure 4).

Training Process

The training involved randomly generated maps of combined C-shaped road sections, ensuring the agent encountered new layouts and traffic scenarios regularly. This variety prevented the agent from simply memorizing patterns, fostering adaptability and reducing overfitting. For a reward function, we used base evaluation provided by the MetaDrive SafeDrivingEnv environment with some hyperparameters changed as shown in Table 1. The agent earned rewards for staying in lanes and maintaining appropriate speeds while facing penalties for collisions and going off-road. Agents were trained for 200k simulation steps using 3 simulations running in parallel. PPO algorithms training process showed to be unstable and sensitive to selected hyperparameters. Depending on selected hyperparameters and reward scaling values, many times agents ended up learning suboptimal strategies, such as: staying in place and not moving at all, turning in circles or deliberately ending an episode by driving off the road. To make neural net convergence more stable we added normalization layers after each custom layer, which made convergence a little bit more stable and predictable. In addition, we removed most conditions related to simulation episode termination, such as driving on the opposite side of the road (penalties still present), this gave agent more freedom for various strategies and improved simulation speed performance, because MetaDrives 3D environments have extremely slow reset times. These changes allowed us to find hyperparameters that gives somewhat reasonable results, keeping in mind that 200k frames is comparatively small amount of data to achieve good results in RL. To improve results a little bit more, we added linear hyperparameter decay for: learning rate, clipping rate and value function clipping rate. This forces algorithm to make smaller and smaller changes which in turn allows it to find more precise, locally optimum strategy location and prevents it from making radical, potentially disruptive changes later in the training process.

Training Performance

Our experiments focus on using pretrained neural networks for visual information without further refinement (frozen weights), which means that image embeddings for the same frames always stay the same. Also, our chosen PPO algorithm in the training process reevaluates its predictions tenths of times, which usually also includes relatively expensive vision models. To avoid unnecessary repetitions of vision model calls, we combined our environment class with vision models and as a result from simulation returned just image embedding values. This massively reduces needed computations and RAM requirements, because we no longer need to save image batches. In our case training speed improved twice.

Considering environments simulation performance for future works, StableBaselines3 and MetaDrive combination showed to be slower than expected. Firstly, Metadrive simulations required multiple GB of RAM each despite being visually simple, and because of that system with 32 GB of RAM could only run 4 parallel simulations at maximum. Secondly, environment restarts take multiple (up to 10) seconds which substantially slows down data collection. Also, in StabeBaselines3, even parallel simulations steps are performed in sync including restarts which further increases restart wait time by multiplier equal to the number of parallel simulations. Some restarts can be avoided by increasing the map's road length (with negligible RAM increase), also using more advanced RL framework with asynchronous execution would remove costly multiplier.

Evaluation, Results and Discussion

The agent's progress was measured at training and later as an evaluation step, using metrics such as success rates, collision frequencies, total distances travelled, and episode durations. These evaluations were performed with different random seeds to test the robustness of the learned policies across varying initial conditions. At evaluation time we also tested algorithms performance on novel unseen types of maps (T intersection).

Despite serving as a benchmark, the CNN-based policy struggled to achieve satisfactory performance. Even after 200,000 training steps, the results fell short of expectations. Extending the training to 1 million steps did not significantly improve performance. Agent only managed to turn in circles. This could be attributed to very small amount of data used to train our networks. Also as seen in the network diagrams, our ResNet34 adaptation has a bottleneck where one layer has only 512 neurons which heavily compressed information and could lose necessary details, although further investigation into this architecture flaw is needed. However, the performance gap between ResNet34 and DinoV2 can primarily be attributed to differences in feature extraction rather than changes to the overall training setup. Since only hyperparameter adjustments were made to optimize each model, intermediate ablation studies were not necessary. The comparison was intended to evaluate the architectures under equivalent conditions rather than isolate individual components. The other possible explanation could be instability of the PPO algorithm, given the fact that the same situations occurred in the DinoV2-based model too. Experimenting with off-policy algorithms could solve this problem because of their optimum policy guarantees.

Meanwhile DinoV2 based model showed promising results with steady improvements when appropriate hyperparameters were used. Although algorithms learned to use steering signals very well, it failed to learn usefulness of breaks, agent was stuck at maximum acceleration at all experiments, despite various reward function and penalties variations. This behavior can be attributed to the reward structure, which emphasized forward progression by rewarding distance traveled toward the finish line. At the start of an episode, the agent received positive rewards for acceleration, as the car was initially well-aligned with the road, and boundaries were relatively far apart. This setup incentivized the agent to maximize immediate rewards by accelerating continuously, creating a reinforcement loop that made braking or cautious driving less favorable. Attempts to adjust the reward structure included testing speed-based incentives, experimenting with small positive rewards for deceleration trying maps that would require deceleration. However, these adjustments were only partially successful, as the agent still associated acceleration with higher rewards overall. Also in order to achieve good final policy—it is common to decrease policy clipping value during training, this allows model to perform small finetuning adjustments, but prohibits large strategy changes which are needed in order to unlearn bias for acceleration. This policy transition problem is related to PPO algorithm and could be solved by using off-policy algorithms like SAC or engineering more diverse environments from the training start where agent would have to brake immediately, but this was out of scope for our research. Also, PPO in this case showed to be quite unstable and sensitive to small hyperparameter changes, opposing to what was expected after literature review.

Despite our efforts to make both neural networks as similar as possible, some differences such as layer count remain. Further research could focus on experimenting with different vision models' outputs adaptations, possibly modifying base models to extract different features.

3.7.2. Step 2: Testing and Refinement in AirSim

This was the testing of the policies trained in MetaDrive in the AirSim simulator. The transition emphasized the challenges in handling more detailed visuals, like dynamic shadows and complex textures, as well as adapting to a realistic physics engine simulating wheel slip and friction. Adjustments needed to be made in the areas of steering, braking, and throttle to maintain stability and improve control in this more demanding simulation environment. This called for revising the manner in which actions were taken and performance measured. Although no fine-tuning was made on the model, domain adaptation techniques and refined mechanisms of control were set up to sustain high performance for the policy. A few key improvements addressed a couple of major challenges in AirSim's complex environment, where refined steering adjustments helped damp excessive wobbling of the vehicle that resulted from minor adjustments, hence handling sharp turns in the path with much ease giving smooth trajectories. Throttle and brake refinements balance speed with stability in high-speed maneuvers and emergency scenarios, such as sudden stops. These targeted changes enhanced the model's adaptability to AirSim's realistic physics and its unpredictable dynamics for improved overall performance and safety in navigation.

Differences Between MetaDrive and AirSim

The transition from MetaDrive to AirSim involved several significant differences that impacted the model's performance. AirSim's visuals included complex textures, dynamic shadows, and more realistic lighting conditions, whereas MetaDrive's visuals were simpler and less detailed. These changes made it harder for the model to interpret the environment without adaptation. Additionally, AirSim had a more advanced physics engine, with realistic vehicle dynamics such as wheel slip, friction, and weight distribution, while MetaDrive used simplified physics. This required more precise control adjustments for actions like steering and braking.

AirSim featured larger and more intricate maps with multiple intersections, tight curves, and varying elevations. In contrast, MetaDrive's maps were procedurally generated but less challenging, with simpler layouts. Vehicle controls, such as steering and throttle, were also more sensitive in AirSim due to its detailed physics engine, demanding fine-tuned adjustments for smooth navigation. Furthermore, AirSim included variable environmental conditions, such as wind and uneven terrain, which were absent in MetaDrive. The combination of realistic visuals and dynamic elements in AirSim required the model to make more context-aware decisions, unlike the relatively predictable scenarios in MetaDrive.

Steering Adjustments

Steering adjustments were made to address two key issues: unnecessary small steering changes that caused the car to wobble, and insufficient steering during sharp maneuvers. Small steering adjustments were reduced to stabilize the vehicle's trajectory, while larger steering outputs were locked and amplified to ensure the car could handle challenging turns effectively. These changes improved overall stability and control, especially during dynamic movements. When the vehicle moved in reverse, steering angles were inverted to ensure accurate corrections. To further smooth the vehicle's trajectory, recent steering outputs were averaged, reducing sudden directional changes and promoting steady control.

Throttle Adjustments

Throttle adjustments were introduced to balance speed and control, particularly during turns and high-speed driving. The throttle was scaled inversely to the magnitude of the steering angle, reducing speed during sharp turns to prevent skidding or overshooting. For smaller steering angles, throttle output was kept near maximum to maintain efficient travel on straight paths. To avoid unsafe speeds, the throttle was proportionally reduced when the vehicle exceeded a predefined speed limit, ensuring consistent and controlled navigation.

Braking Mechanisms

Braking mechanisms were also refined to improve safety. Braking intensity was calculated based on the vehicle's deceleration. When deceleration was detected, braking force was applied proportionally to maintain stability. Emergency braking was added for situations involving extreme steering angles or abrupt speed changes, preserving control in challenging conditions and preventing accidents.

Gear Selection and Reversing Logic

Gear selection and reversing logic depended on the vehicle's speed and acceleration. Positive acceleration automatically engaged the forward gear, while negative acceleration below a specific threshold triggered reverse gear. During reversing, throttle and steering outputs were carefully coordinated to ensure smooth and stable repositioning without sudden directional changes, improving maneuverability in reverse.

Action Synchronization

A sequential execution mechanism prevented conflicting actions (e.g., simultaneous braking and throttle application). This ensured the agent's actions were logically coherent and physically realistic.

Domain Adaptation

Domain adaptation techniques were essential for adapting the model from MetaDrive to AirSim's more complex environment. Domain adaptation techniques were applied to address the visual differences between MetaDrive and AirSim. These included Adaptive Instance Normalization (AdaIN) and a lightweight style transfer mechanism. A representative style image from MetaDrive was resized to 224×224 pixels, normalized using ImageNet statistics, and processed through DinoV2's forward features to extract normalized patch tokens (x_norm_patchtokens). These tokens were used to align the embeddings of MetaDrive frames with AirSim's higher-fidelity visuals in an efficient manner. However, the overall impact of these adjustments on model performance was minimal, likely due to DinoV2's robust pretraining, which enabled generalization across visual domains without requiring significant alignment. Temporal embedding stacking further improved the model's ability to recognize changes over time, such as moving vehicles and shifts in lighting, by using a rolling buffer of four consecutive image embeddings. These techniques addressed challenges like handling AirSim's detailed visuals and dynamic elements, enabling smoother navigation and better decision-making.

Evaluation Metrics

Policy performance in AirSim was assessed using several key metrics to evaluate navigation efficiency and safety. The average episode length, which measures how long the agent could navigate, decreased in AirSim due to its more complex maps and challenging road layouts, making it harder to maintain stable navigation. Collision rates, indicating how often the agent hit obstacles and reflecting safety, were higher because of AirSim's realistic physics and detailed environments. The average speed, showing how smoothly the agent drove, was lower as vehicles had to slow down frequently for sharp turns and intersections. Success rates, which track how often the agent completed its goals without errors, also dropped in the more demanding AirSim environment. Finally, the average distance traveled per episode, which shows how far the agent could go without unnecessary stops, was shorter because of the increased difficulty in AirSim. Together, these metrics highlighted the greater challenges posed by AirSim and the model's struggles to adapt to a more realistic and demanding simulation environment.

Scenario Design and Execution

Scenario design in AirSim ensured consistent and fair testing conditions. At the start of each episode, the environment was re-initialized, resetting both the vehicle's state and environmental parameters to provide comparable conditions across trials. Episodes ran for a fixed number of simulation steps, ending when the vehicle reached its goal or experienced a collision. Throughout each episode, the agent processed real-time observations, including speed, position, and collision data, to guide its actions. After each run, metrics such as distance travelled, collision count, episode length, and average speed were recorded. These metrics were aggregated across multiple trials to produce a statistically reliable assessment of the policy's performance and ability to generalize to different scenarios.

4. Results

We start by looking at how the models were trained in the MetaDrive simulator, comparing the DinoV2 ViT-S/14 and ResNet34 models over different training periods. Next, we review how the trained models performed in MetaDrive and their key results. Finally, we discuss testing the models in the AirSim simulator and the adjustments made to improve their performance in this more complex environment. These results show the challenges of moving models trained in simpler simulators like MetaDrive to more realistic ones like AirSim. They highlight the need to fine-tune important settings like steering control, lane correction, and sensor adjustments to achieve better performance in these challenging conditions.

4.1. Model Training Evaluation

This section evaluates model performance during training in the MetaDrive simulator. Models were trained for 200,000 and 1,000,000 steps, with the backbone feature extractors (DinoV2 ViT-S/14 and ResNet34) frozen during training. Key metrics, including mean episode reward and mean episode length, were compared to assess performance in Figure 7.

As shown in the chart, the DinoV2 model trained for 1,000,000 steps achieved the best performance, with the longest mean episode length (1394.8) and the highest mean reward (676.49). The DinoV2 model trained for 200,000 steps performed moderately, producing shorter episodes and lower rewards. ResNet34 showed consistently weaker performance across both training durations, with the 200,000-step version performing the worst. These results demonstrate DinoV2's superior capability, particularly when trained for a longer duration.

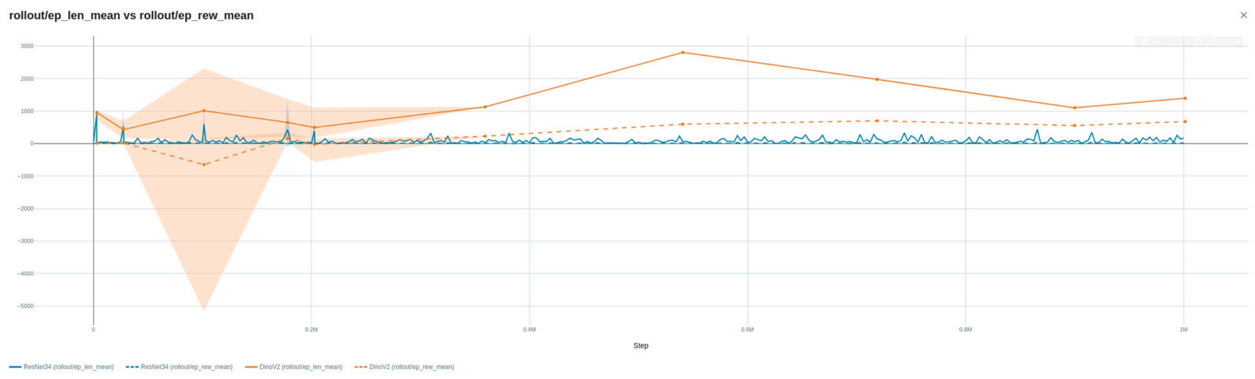


Figure 7. Model performance comparison across training steps in the MetaDrive simulator. Here yellow represents model with DinoV2 ViT-S/14 backbone trained for 1 million steps, blue represents model with ResNet34 backbone trained for 1 million steps.

From multiple hyperparameter tuning experiments, the averaged graphs (Figures 8 and 9) show key performance trends. Figure 8 illustrates how, as training progresses, the model learns to approximate the expected return from the environment, reducing errors and improving its ability to generalize from experience. Similarly, Figure 9 shows the mean episode reward increasing over training steps because the model's improved generalization enables it to learn better driving strategies, leading to higher rewards. All models in these graphs were trained for 200,000 frames on a map with curved road segments. The blue line represents DinoV2-based models, and the yellow line shows ResNet34-based models. On average, the ViT-based models achieved higher rewards, and the difference in performance grew with training. The PPO algorithm also seemed to evaluate the agent's state more effectively and faster when using the Vision Transformer, further proving its advantage over CNN-based models.

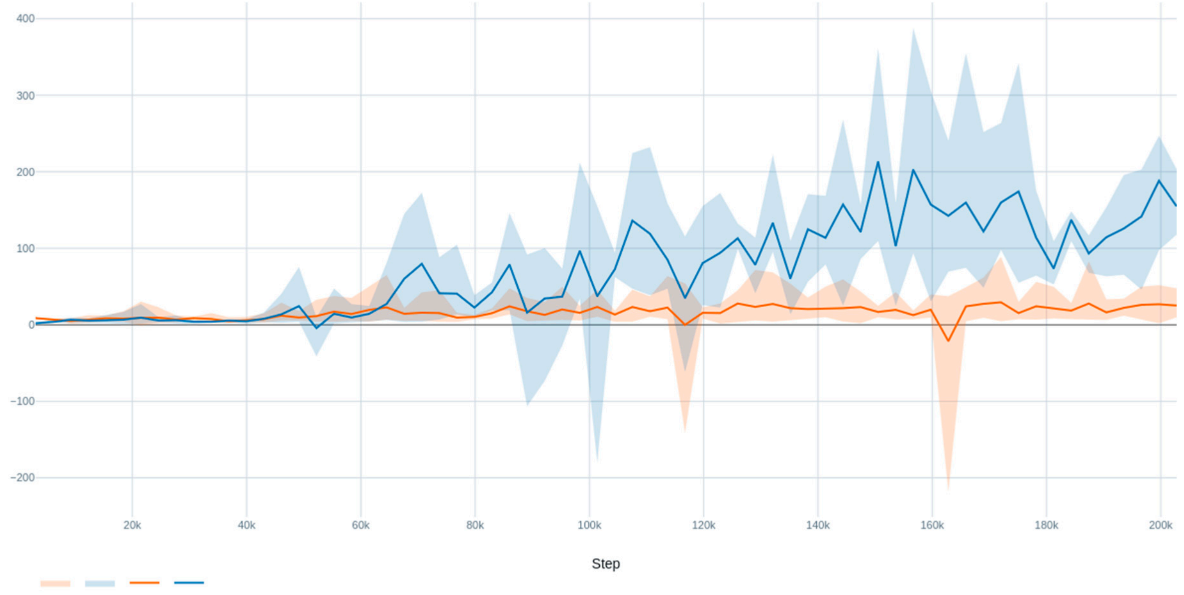


Figure 8. The mean episode reward throughout model training. The blue line shows model with DinoV2 backbone while the orange line shows model with ResNet34 model backbone.

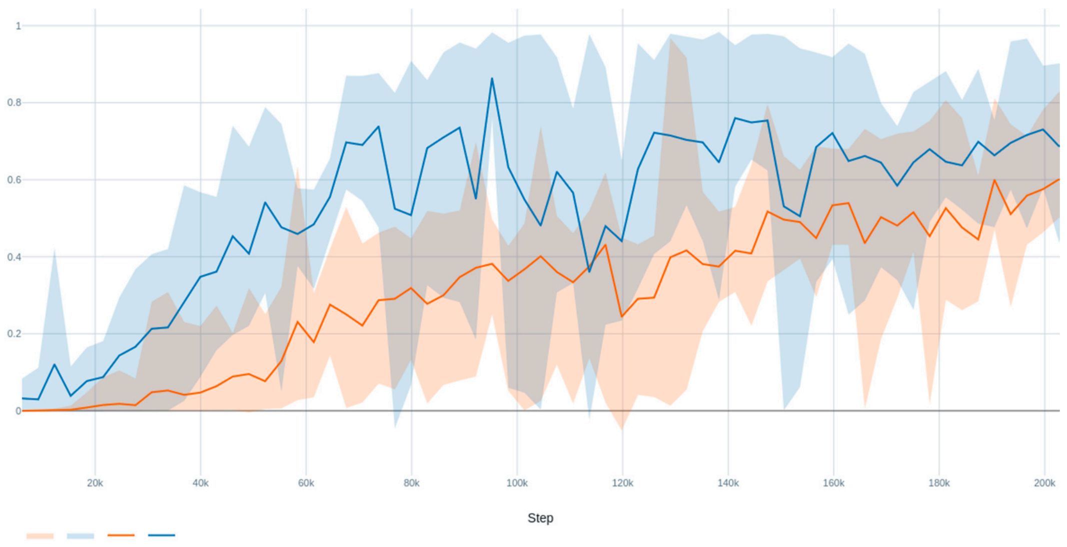


Figure 9. Value function's explained variance difference throughout model training. The blue line shows model with DinoV2 backbone while the orange line shows model with ResNet34 backbone model.

4.2. Trained Model Results in Metadrive

Before evaluating the trained models in the MetaDrive simulator, several adjustments were made to the default settings. The vehicle's maximum speed was reduced, and the termination rule for entering oncoming traffic was disabled. This decision allowed the agent to explore navigation strategies more freely, particularly during early training phases, without being prematurely penalized for minor deviations. However, this adjustment also led to the emergence of suboptimal behaviors, as the agent occasionally prioritized smoother paths in opposing lanes over staying within its designated lane. To counteract this, additional vehicles were introduced in opposing lanes, creating a higher risk of collisions and incentivizing the agent to adhere to correct lane usage. These adjustments balanced exploration and realistic behavior development, ensuring that the agent could generalize better across diverse driving scenarios.

To ensure a fair comparison between MetaDrive and AirSim, the way distances were measured was adjusted to account for differences in how each simulator calculates them. MetaDrive measures distances in meters, while AirSim uses the cumulative Euclidean distance between consecutive coordinates. AirSim's method often exaggerates path lengths due to its finer granularity and the inclusion of curves.

To align MetaDrive's distance measurements with AirSim's format, the average step distance in MetaDrive (0.222 m, calculated from the distance and episode length) was scaled by a factor of 4. This scaling factor was derived empirically by generating identical paths in both simulators and recording the total distance traveled by the vehicle. The results showed that AirSim's Euclidean distance calculation, which accounts for finer granularity and curved trajectories, inflated distance measurements by approximately four times compared to MetaDrive's straight-line segment calculations. This validation step supports the appropriateness of the $\times 4$ scaling factor, ensuring that the reported results are directly comparable across the two simulators.

The evaluation results (see Table 3) show that the DinoV2 ViT-S/14 model significantly outperformed ResNet34 as the backbone model in the MetaDrive simulator. After 1 million training steps, DinoV2 achieved the best overall performance, with an average distance traveled of 265.91 units, minimal collisions (0.2), the longest episode length (299.2 steps), and a higher average speed (7.82 units). At 200,000 steps, its performance was lower, with an average distance of 67.19 units, a collision rate of 1, but a competitive speed of 6.21 units.

Table 3. Trained model evaluation results in Metadrive.

Backbone Model	Training Steps	Avg. Distance Travelled (m)	Avg. Collisions	Avg. Episode Length (s)	Avg. Speed (km/h)
DinoV2 ViT-S/14 distilled with registers	1,000,000	265.91	0.2	299.2	7.82
DinoV2 ViT-S/14 distilled with registers	200,000	67.19	1	75.6	6.21
ResNet34	1,000,000	39.1	0	44	4.18
ResNet34	200,000	51.54	0	58	3.01

In contrast, ResNet34 showed only slight improvement even with extended training. After 1 million steps, it managed an average distance of just 39.1 units, no collisions, an episode length of 44 steps, and a speed of 4.18 units. At 200,000 steps, its performance was similarly limited, achieving an average distance of 51.54 units with a lower speed of 3.01 units.

These findings emphasize the superior ability of DinoV2 ViT-S/14 to process visual information for autonomous driving tasks, especially when given sufficient training. ResNet34's limited performance highlights its difficulty in handling the complexity of visual navigation in MetaDrive.

While DinoV2 performed better in numbers, test drives showed some problems. The model sometimes went outside lane boundaries and had trouble keeping a steady speed, meaning it needs better control and to follow driving rules more closely. ResNet34, on the other hand, often went off the road completely. This explains its low collision rates but shows that it couldn't drive steadily, making it unreliable for autonomous driving in this environment.

4.3. Results from Trained Model Testing in AirSim

Testing a model trained in a simpler simulator like MetaDrive in a more complex environment like AirSim posed significant challenges due to the differences in visuals, physics, and map complexity. The Vision Transformer (ViT)-based model showed better adaptability than the CNN model but still required several adjustments. Changes to steering control, lane detection, camera placement, and braking logic were necessary to align the model's actions with AirSim's realistic physics and detailed visuals. These adjustments were guided by the unique demands of AirSim's environment, including more intricate road layouts and dynamic conditions. Together, these refinements helped the model navigate the complex environment effectively and prepared it for performance evaluations.

The results (Table 4) highlight that steering dynamics are the most critical aspect of successful navigation. Adjustments to steering logic play a vital role in autonomous driving performance. When additional steering logic was removed, the model's performance dropped dramatically, with an 82.9% decrease in distance traveled and nearly halved average speed compared to the baseline. Without proper steering adjustments, the model could not control the vehicle effectively, resulting in erratic or stalled behavior. This highlights the importance of fine-tuning steering logic to align with AirSim's more sensitive and realistic physics system. Steering adjustments ensured smoother trajectories, even in the absence of other advanced features like lane detection or style transfer.

Table 4. AirSim simulator adjustments results.

Scenario	Average Distance Travelled (m)	Average Collisions	Average Episode Length (s)	Average Speed (km/h)
Fully implemented system	103.47	1	9.4	6.82
No forward camera adjustment	112.01	1	11.6	8.12
No lane correction logic (removing line detection)	128.62	1	14.8	7.98
No style transfer, no forward camera adjustment, no lane correction	102.56	1	9.8	6.33
No brake logic, no forward camera adjustment, no lane correction	98.14	1	12.2	7.36
No speed logic, no forward camera adjustment, no lane correction	114.38	1	11.2	10.88
No steering logic, no forward camera adjustment, no lane correction	17.7	1	6	3.45

Lane detection corrections, initially implemented using a Hough Transform-based method, aimed to align the vehicle's trajectory by detecting lane boundaries. However, in AirSim's high-fidelity environment, this approach often misidentified non-lane elements, such as shadows, curbs, and complex textures, as lane markings. These noisy inputs frequently conflicted with the reinforcement learning (RL) policy, which already utilized visual embeddings to make context-aware navigation decisions. This conflict resulted in oscillatory or unstable steering behavior, particularly on curved roads or intersections with ambiguous or missing lane markings.

Disabling the lane detection corrections resolved these conflicts by allowing the RL policy to operate independently, relying solely on its learned understanding of the environment. This simplification led to the best performance overall, with a 24.3% increase in distance traveled compared to the baseline. This result underscores that in high-variability environments like AirSim, simpler systems relying on robust learned policies can outperform pipelines that introduce noise through redundant or conflicting inputs.

Removing forward camera adjustments resulted in an 8.54% improvement in distance traveled, indicating that the default sensor alignment may already be optimal. Proper placement of sensors is critical, and any adjustments should be carefully tested to ensure they improve rather than disrupt the perception system.

Disabling the speed control logic resulted in the highest average speed and a notable improvement in distance travelled, with a 10.6% increase over the baseline. However, this approach comes with risks, as faster speeds can reduce control and increase the chances of instability or collisions. Striking a balance between aggressive speed strategies and stability is crucial. Adaptive speed control, which adjusts speed based on real-time conditions, could provide better results than fixed-speed strategies.

When braking logic was removed, the distance travelled decreased slightly by 5.2%, but the average speed increased by 7.9%. Braking logic helps maintain stability by smoothing sudden changes in speed or direction, but it can also limit forward momentum. To optimize performance, braking logic should be adjusted to allow for better momentum while still ensuring stability during challenging maneuvers.

Removing style transfer caused only a minor impact on performance, with a 0.9% reduction in distance travelled and a 7.2% drop in speed. This suggests that style transfer is not essential for navigation in this scenario. While it helps align visual embeddings with training conditions, such as images from MetaDrive, the DinoV2 features proved robust enough to perform effectively without it. Style transfer remains useful in environments with significant visual differences or domain shifts, such as transitioning from synthetic data to real-world images, but it is not critical in all cases.

Despite its strong performance in MetaDrive (Table 3). the model with DinoV2 ViT-S/14 model faced difficulties in the more realistic AirSim environment. Lane correction logic had to be disabled due to conflicts with dynamic conditions, and a speed limit was necessary to maintain control. Even with these adjustments the average distance travelled dropped from 265.91 to 128.62 units. Collisions increased to an average of 1 per episode, and although the average speed remained comparable at 7.98 units, the model's episode length was reduced to 14.8, indicating challenges in maintaining stable navigation.

4.4. Reasons for Environment Transition Difficulties

As shown by results DinoV2 did allow algorithm to generalize to novel environments. It demonstrates that DinoV2 enabled successful model transfer from the lightweight MetaDrive simulator to the more complex AirSim, proving that reinforcement learning models trained in simplified environments can adapt to more realistic settings. However, full real-world deployment remains challenging due to visual, physics, sensor, and control differences.

One major limitation was that some real-world dynamics in AirSim had to be refined for our model to function correctly. Features like advanced physics steering control, lane detection, camera placement, and braking logic caused instability, forcing us to align these control mechanics. This highlight critical gaps in the model's adaptability, particularly in handling complex physics, environmental variations, and real-time perception.

Agents suffered from big action mistiming and strength. From this we can notice a major unaddressed part in our study—physics and car parameter differences. Even in the same simulator different car models can have different mass, acceleration, speed and turning radius parameters, which wasn't accounted for when creating the primary neural networks. In addition to control signals, algorithms received images can be recorded from different relative positions and angles, which can be mistakenly identified by neural network as different distances.

To truly enable transfer to real-world scenarios, further investigation is needed in domain adaptation (to handle visual differences like lighting and reflections), vehicle physics modeling (to improve braking, acceleration, and steering response), sensor fusion (to integrate IMU, and radar for better perception), and adaptive control strategies (to dynamically adjust to varying road and vehicle conditions). Addressing these areas will be essential for ensuring a smooth transition from simulation to real-world autonomous driving.

5. Conclusions and Future Research

In this study we show that it's possible to use vision transformers and patch-based information for autonomous reinforcement learning tasks. With a relatively limited about of training data (200k frames) and PPO algorithm we achieved 87% better performance using ViT type DinoV2 backbone over CNN type ResNet34. Secondly, we show that vison backbones pretrained in self-supervised fashion offers big advantages over recognition networks, as DinoV2's abilities to adapt to different visual domains were successfully utilized in RL tasks. This allowed our algorithm to adapt to entirely different visuals in different simulator with no additional modifications.

Despite success in vision, adaptability to different physical properties of the vehicle and the road still remains a challenge. Future work should focus on giving an agent the ability to evaluate its actions results from visual information, in turn allowing it to plan in accordance with its limitations.

While convolutional neural networks (CNNs) have been a reliable choice for vision-based tasks, our experiments demonstrated their limitations in capturing long-range dependencies and understanding global contextual cues, particularly when using a relatively small dataset (approximately 200,000 frames). CNNs rely on local receptive fields, which restrict their ability to process spatial relationships across an entire image, making it harder to generalize to complex driving scenarios. In contrast, Vision Transformers (ViTs) utilize self-attention mechanisms that analyze global patterns and relationships in visual data, enabling a more comprehensive understanding of spatial structures. This led to faster training and more robust policies under the same conditions. Based on this performance, it is reasonable to infer that training with a larger dataset could further enhance the robustness of ViTs, enabling them to handle an even more visually complex environment.

Our trained ViT-based policy performed better in handling complex road layouts, varying traffic patterns, and challenging road designs such as T-intersections and C-shaped curves compared to CNN's. This ability to handle visually complex environments highlights ViTs' potential for enhancing safety and reliability in real-world autonomous vehicle systems. Furthermore, their demonstrated adaptability across simulated environments suggests they could reduce retraining costs and accelerate deployment in diverse geographic regions.

Using universal DinoV2 features allowed the system to adapt to different visual environments when other settings were carefully aligned. This was evident in the partial success of transferring the trained agent to a more realistic simulator (AirSim). However, additional training is needed to better address differences in simulator physics and camera angles. These findings show the potential of ViTs to reduce the gap between controlled simulations and more realistic testing conditions.

Lightweight embedding-level style transfer techniques showed limited impact in addressing the domain gap between MetaDrive and AirSim, likely due to the strong generalization capabilities of DinoV2. Additionally, refining perception and control strategies will remain critical for bridging the gap between simulation and real-world implementation. Steering control has been identified as a key factor in navigation, suggesting that more adaptive steering strategies or advanced reinforcement learning algorithms could enhance performance. Simplifying perception systems by relying on robust, general-purpose feature extraction instead of complex, rule-based methods could streamline system design and improve reliability.

Improving sensor placement and camera angles could also enhance perception quality without adding unnecessary computational load. Additionally, control mechanisms for speed and braking should be fine-tuned to strike a balance between safety and efficiency. More advanced RL algorithms could be better at learning appropriate speed strategy. The use of style transfer techniques to adapt to different operating conditions should be considered based on the specific differences between training and deployment environments. Finally, future work should explore deploying ViT-based reinforcement learning systems in real-world scenarios or in more realistic, visually detailed simulation environments. This would help validate their ability to handle larger images and operate effectively in real-time settings, taking the research a step closer to practical applications. For the full implementation details, code, and trained models, refer to the public GitHub repository at https://github.com/ktu-need-for-speed-team/self-driving-research (accessed on 5 January 2025).

This repository provides resources to save time and enable reproducibility by offering pretrained models, configuration files and scripts. The models, stored in the models folder, include DinoV2 backbones trained for 200k and 1M steps, as well as ResNet34 backbones trained for equivalent durations. Configuration files in the configs folder define setups for both training and evaluation. Scripts such as main.py facilitate flexible training and testing with parameters that enable automated metrics collection.

Two simulators were employed in this research, MetaDrive and AirSim, each serving distinct roles. MetaDrive, version 0.4.2.3, offers lightweight, procedurally generated scenarios and was configured with custom settings such as *CCCC* road maps, adjusted penalties, and low traffic density. AirSim, on the other hand, provides high-fidelity simulations with realistic visuals and physics, including detailed textures, dynamic shadows, and vehicle dynamics. Custom scenarios in AirSim featured variable environmental conditions, enhancing the complexity and realism of the tests. Both simulator settings and road layouts are documented to ensure experimental replication.

The methodology utilized the Proximal Policy Optimization (PPO) algorithm with key hyperparameters, including a learning rate of 4×10^{-4}, batch size of 1024×3, and a discount factor of 0.99. Linear decay was applied to stabilize training. Vision Transformers (ViTs), specifically the DinoV2 model, were integrated as replacements for CNNs such as ResNet34. DinoV2 embeddings were compressed to optimize reinforcement learning tasks, resulting in improved efficiency and performance.

The hardware setup included an Nvidia RTX 4060 GPU with 8 GB of VRAM, 32 GB of RAM, and a 200 GB SSD for data storage. The software stack featured Python 3.11 and PyTorch 2.4.0 for model development, Stable Baselines3 2.3.2 for reinforcement learning, and additional tools like NumPy, Pandas, OpenCV, and MLflow for data processing and tracking. This configuration ensured the computational and developmental needs of the project were fully supported.

Evaluation protocols focused on metrics such as success rate, collision rate, average distance traveled, and episode length. Automated scripts streamlined the collection of these metrics, while the use of random seeds ensured consistent results. Logs of training and testing processes were detailed to allow for performance benchmarking.

Comprehensive documentation is provided to guide setup and usage, with detailed instructions in the README file for installing dependencies, configuring environments, and running the system. These include commands for setting up MetaDrive, installing PyTorch, and verifying installations. Examples of training and testing processes are included, along with results from MetaDrive and AirSim. The availability of pretrained models, documented configurations, and usage examples ensures that researchers can easily replicate the findings. Additionally, this work lays a foundation for future research, such as extending the simulation results to real-world implementations, improving performance in the visual processing components, or exploring other domains like adaptive control systems, multi-agent learning, or task-specific optimizations. This flexibility highlights the potential for advancing the field of autonomous driving and related applications.

Author Contributions: Conceptualization, A.O.; Methodology, R.A., J.R., M.Š. and Ž.Š.; Software, Ž.Š., R.A. and J.R.; Validation, R.A., J.R., M.Š. and Ž.Š.; Formal Analysis, R.A. and J.R.; Investigation, Ž.Š., R.A., J.R. and M.Š.; Resources, A.O.; Data Curation, R.A.; Writing—Original Draft, R.A.; Writing—Review & Editing, R.A., J.R., M.Š., Ž.Š. and A.O.; Visualization, R.A.; Supervision, A.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The use of artificial intelligence techniques for language enhancement and grammar checking is acknowledged by the writers.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

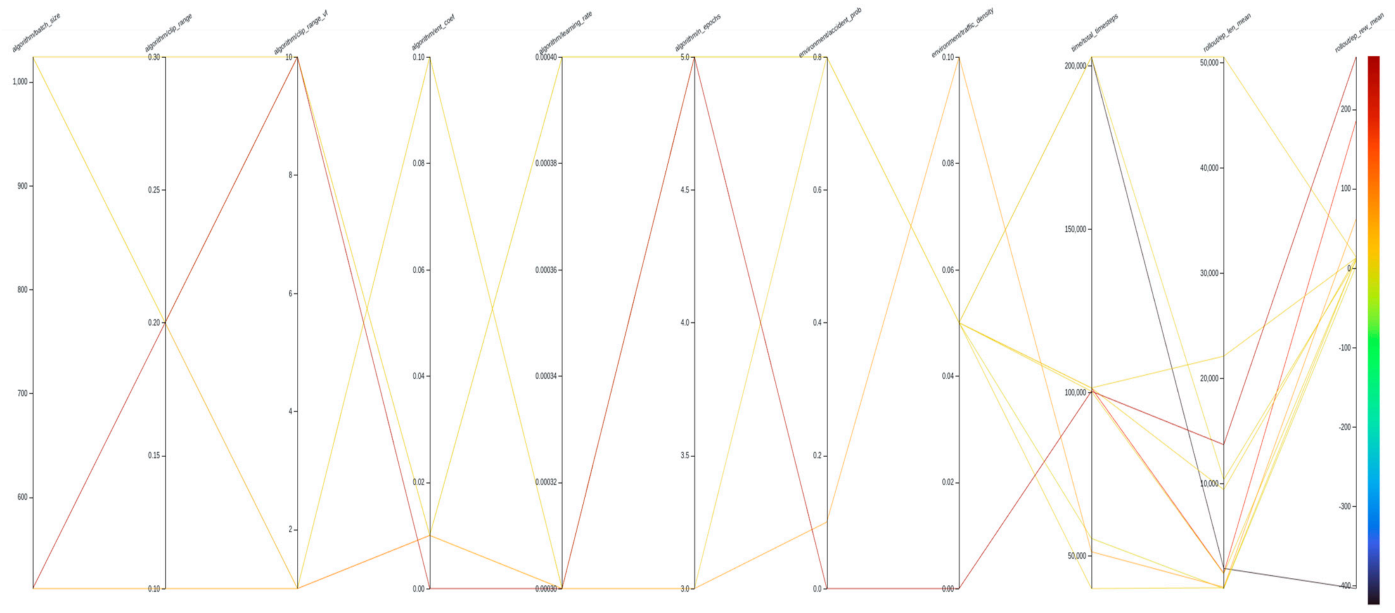


Figure A1. Parallel coordinates plot illustrating results from various hyperparameter combinations. Each line represents a different experiment, with individual axes corresponding to specific hyperparameters. The final column represents the median episode reward, which serves as the primary criterion for selecting optimal hyperparameters. Additionally, training curve stability, training time, and consistency across multiple trials are considered when evaluating performance. The color gradient indicates variations in performance, helping to identify the most effective hyperparameter settings.

References

1. Yurtsever, E.; Lambert, J.; Carballo, A.; Takeda, K. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access* **2020**, *8*, 58443–58469. Available online: https://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=9046805 (accessed on 24 November 2024). [CrossRef]
2. Crayton, T.; Meier, B.M. Autonomous Vehicles: Developing a Public Health Research Agenda to Frame the Future of Transportation Policy. 2017. Available online: https://papers.ssrn.com/abstract=2966084 (accessed on 24 November 2024).
3. Cultrera, L.; Becattini, F.; Seidenari, L.; Pala, P.; Bimbo, A.D. Addressing Limitations of State-Aware Imitation Learning for Autonomous Driving. *IEEE Trans. Intell. Veh.* **2024**, *9*, 2946–2955. [CrossRef]
4. Tampuu, A.; Tambet, M.; Semikin, M.; Fishman, D.; Muhmmad, N. A Survey of End-to-End Driving: Architectures and Training Methods. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 1364–1384. Available online: http://arxiv.org/abs/2003.06404 (accessed on 8 January 2025). [CrossRef] [PubMed]
5. El Sallab, A.; Abdou, M.; Perot, E.; Yogamani, S. Deep Reinforcement Learning framework for Autonomous Driving. *Electron. Imaging* **2017**, *29*, 70–76. Available online: http://arxiv.org/abs/1704.02532 (accessed on 5 January 2025). [CrossRef]
6. Chen, C.; Seff, A.; Kornhauser, A.; Xiao, J. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 1–13 December 2015; pp. 2722–2730. Available online: https://ieeexplore.ieee.org/document/7410669 (accessed on 5 January 2025).
7. Huval, B.; Wang, T.; Tandon, S.; Kiske, J.; Song, W.; Pazhayampallil, J.; Andriluka, M.; Rajpurkar, P.; Migimatsu, T.; Cheng-Yue, R.; et al. An Empirical Evaluation of Deep Learning on Highway Driving. 2015. Available online: http://arxiv.org/abs/1504.01716 (accessed on 8 January 2025).
8. Zhang, Z.; Liniger, A.; Dai, D.; Yu, F.; Van Gool, L. End-To-End Urban Driving by Imitating a Reinforcement Learning Coach. 2021. Available online: http://arxiv.org/abs/2108.08265 (accessed on 5 January 2025).

9. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An Image Is Worth 16 × 16 Words: Transformers for Image Recognition at Scale. 2021. Available online: http://arxiv.org/abs/2010.11929 (accessed on 5 January 2025).
10. Lai-Dang, Q.-V. A Survey of Vision Transformers in Autonomous Driving: Current Trends and Future Directions. 2024. Available online: http://arxiv.org/abs/2403.07542 (accessed on 5 January 2025).
11. Khan, S.; Naseer, M.; Hayat, M.; Zamir, S.W.; Khan, F.S.; Shah, M. Transformers in Vision: A Survey. *ACM Comput. Surv.* **2022**, *54*, 1–41. Available online: http://arxiv.org/abs/2101.01169 (accessed on 5 January 2025). [CrossRef]
12. Li, Q.; Peng, Z.; Feng, L.; Zhang, Q.; Xue, Z.; Zhou, B. MetaDrive: Composing Diverse Driving Scenarios for Generalizable Reinforcement Learning. 2022. Available online: http://arxiv.org/abs/2109.12674 (accessed on 24 November 2024).
13. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*; Springer International Publishing: Cham, Germany, 2018; pp. 621–635. Available online: https://link.springer.com/chapter/10.1007/978-3-319-67361-5_40 (accessed on 5 January 2025).
14. Barrile, V.; Simonetti, S.; Citroni, R.; Fotia, A.; Bilotta, G. Experimenting Agriculture 4.0 with Sensors: A Data Fusion Approach Between Remote Sensing, UAVs and Self-Driving Tractors. *Sensors* **2022**, *22*, 7910. Available online: https://www.mdpi.com/1424-8220/22/20/7910 (accessed on 2 February 2025). [CrossRef] [PubMed]
15. Codevilla, F.; Müller, M.; López, A.; Koltun, V.; Dosovitskiy, A. End-To-End Driving via Conditional Imitation Learning. 2018. Available online: http://arxiv.org/abs/1710.02410 (accessed on 5 January 2025).
16. Chen, D.; Zhou, B.; Koltun, V.; Krähenbühl, P. Learning by Cheating. *Proc. Conf. Robot. Learn.* **2019**, *100*, 66–75. Available online: https://www.semanticscholar.org/paper/Learning-by-Cheating-Chen-Zhou/ca5045c9d9e0bf2e95f6694dff657e28ffcd4f07 (accessed on 5 January 2025).
17. Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L.D.; Monfort, M.; Muller, U.; Zhang, X.; et al. End to End Learning for Self-Driving Cars. 2016. Available online: http://arxiv.org/abs/1604.07316 (accessed on 5 January 2025).
18. Xu, H.; Gao, Y.; Yu, F.; Darrell, T. End-to-End Learning of Driving Models from Large-Scale Video Datasets. 2017. Available online: http://arxiv.org/abs/1612.01079 (accessed on 5 January 2025).
19. Chen, X.; Ma, H.; Wan, J.; Li, B.; Xia, T. Multi-View 3D Object Detection Network for Autonomous Driving. 2017. Available online: http://arxiv.org/abs/1611.07759 (accessed on 5 January 2025).
20. Zeng, W.; Luo, W.; Suo, S.; Sadat, A.; Yang, B.; Casas, S.; Urtasun, R. End-to-End Interpretable Neural Motion Planner. 2021. Available online: http://arxiv.org/abs/2101.06679 (accessed on 5 January 2025).
21. Touvron, H.; Cord, M.; Douze, M.; Massa, F.; Sablayrolles, A.; Jegou, H. Training Data-Efficient Image Transformers & Distillation Through Attention. 2021. Available online: http://arxiv.org/abs/2012.12877 (accessed on 5 January 2025).
22. Ando, A.; Gidaris, S.; Bursuc, A.; Puy, G.; Boulch, A.; Marlet, R. RangeViT: Towards Vision Transformers for 3D Semantic Segmentation in Autonomous Driving. 2023. Available online: http://arxiv.org/abs/2301.10222 (accessed on 5 January 2025).
23. Raghu, M.; Unterthiner, T.; Kornblith, S.; Zhang, C.; Dosovitskiy, A. Do Vision Transformers See Like Convolutional Neural Networks? 2022. Available online: http://arxiv.org/abs/2108.08810 (accessed on 7 January 2025).
24. Tolstikhin, I.; Houlsby, N.; Kolesnikov, A.; Beyer, L.; Zhai, X.; Unterthiner, T.; Yung, J.; Steiner, A.; Keysers, D.; Uszkoreit, J.; et al. MLP-Mixer: An All-MLP Architecture for Vision. 2021. Available online: http://arxiv.org/abs/2105.01601 (accessed on 7 January 2025).
25. Leite, D.; Teixeira, I.; Morais, R.; Sousa, J.J.; Cunha, A. Comparative Analysis of CNNs and Vision Transformers for Automatic Classification of Abandonment in Douro's Vineyard Parcels. *Remote Sens.* **2024**, *16*, 4581. Available online: https://www.mdpi.com/2072-4292/16/23/4581 (accessed on 8 January 2025). [CrossRef]
26. Alijani, S.; Fayyad, J.; Najjaran, H. Vision Transformers in Domain Adaptation and Domain Generalization: A Study of Robustness. 2024. Available online: http://arxiv.org/abs/2404.04452 (accessed on 7 January 2025).
27. Yue, X.; Zhang, Y.; Zhao, S.; Sangiovanni-Vincentelli, A.; Keutzer, K.; Gong, B. Domain Randomization and Pyramid Consistency: Simulation-To-Real Generalization Without Accessing Target Domain Data. 2022. Available online: http://arxiv.org/abs/1909.00889 (accessed on 7 January 2025).
28. Xu, P.; Zhu, X.; Clifton, D.A. Multimodal Learning with Transformers: A Survey. 2023. Available online: http://arxiv.org/abs/2206.06488 (accessed on 7 January 2025).
29. Bilotta, G.; Genovese, E.; Citroni, R.; Cotroneo, F.; Meduri, G.M.; Barille, V. Integration of an Innovative Atmospheric Forecasting Simulator and Remote Sensing Data into a Geographical Information System in the Frame of Agriculture 4.0 Concept. *AgriEngineering* **2023**, *5*, 1280–1301. Available online: https://www.mdpi.com/2624-7402/5/3/81 (accessed on 2 February 2025). [CrossRef]
30. Diffusion Models in Vision: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence. Available online: https://dl.acm.org/doi/10.1109/TPAMI.2023.3261988?utm_source=chatgpt.com (accessed on 7 January 2025).

31. Katiyar, N.; Shukla, A.; Chawla, N.; Singh, R.; Singh, S.K.; Husain, F. AI in Autonomous Vehicles: Opportunities, Challenges, and Regulatory Implications. *Educ. Adm. Theory Pract.* **2024**, *30*, 6255–6264. Available online: https://kuey.net/index.php/kuey/article/view/2373 (accessed on 8 January 2025). [CrossRef]
32. He, K.; Chen, X.; Xie, S.; Li, Y.; Dollár, P.; Girshick, R. Masked Autoencoders Are Scalable Vision Learners. 2021. Available online: http://arxiv.org/abs/2111.06377 (accessed on 7 January 2025).
33. Gutiérrez-Moreno, R.; Barea, R.; López-Guillén, E.; Araluce, J.; Bergasa, L.M. Reinforcement Learning-Based Autonomous Driving at Intersections in CARLA Simulator. *Sensors* **2022**, *22*, 8373. Available online: https://www.mdpi.com/1424-8220/22/21/8373 (accessed on 7 January 2025). [CrossRef] [PubMed]
34. Voogd, K.; Allamaa, J.P.; Alonso-Mora, J.; Son, T.D. Reinforcement Learning from Simulation to Real World Autonomous Driving Using Digital Twin. 2022. Available online: http://arxiv.org/abs/2211.14874 (accessed on 7 January 2025).
35. Zhang, T.; Liu, H.; Wang, W.; Wang, X. Virtual Tools for Testing Autonomous Driving: A Survey and Benchmark of Simulators, Datasets, and Competitions. *Electronics* **2024**, *13*, 3486. Available online: https://www.mdpi.com/2079-9292/13/17/3486 (accessed on 7 January 2025). [CrossRef]