

UNIVERSITY OF TÜBINGEN

Faculty of Science
Wilhelm-Schickard-Institute for Computer Science

MASTER THESIS IN MACHINE LEARNING

Enhancing Model-based Reinforcement Learning for Autonomous Driving

Jens Beißwenger

May 02, 2025

Reviewers

Prof. Dr. Andreas Geiger
Department of
Computer Science
University of Tübingen

Prof. Dr. Georg Martius
Department of
Computer Science
University of Tübingen

Supervisors

Kashyap Chitta & Bernhard Jaeger
Department of Computer Science
University of Tübingen

Author: Beißwenger, Jens (6198682)

Title: *Enhancing Model-based Reinforcement Learning for Autonomous Driving*

Degree: Master of Science in Machine Learning

Institution: University of Tübingen

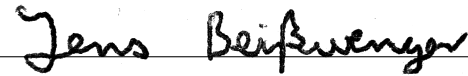
Period: November 01, 2024 – April 30, 2025

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind.

Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Tübingen, May 2, 2025

A handwritten signature in black ink, reading "Jens Beißwenger", written over a horizontal line.

Jens Beißwenger (Matrikelnummer: 6198682)

Abstract

We present CaRL-Lite, a model-based reinforcement learning method for autonomous driving in the CARLA simulator. Current reinforcement learning (RL) methods for autonomous driving require significant computational resources, including hundreds of CPU cores and multiple high-end GPUs, due to slow driving simulators and the large number of samples needed for RL training. This presents a major obstacle when developing reinforcement learning policies. State-of-the-art approaches in this domain are primarily rule-based; however, these methods fail to scale effectively to address uncommon driving scenarios. To overcome this limitation, we implement CaRL-Lite, a modified DreamerV3 architecture that learns a world model of CARLA, which allows efficient policy training through latent space simulation instead of direct interaction with the slow simulator. Our approach uses Birds-Eye View (BEV) images and incorporates a dual-modal stochastic latent space for visual and scalar inputs alongside a multiplicative reward function. On the Longest6 v2 benchmark, our model surpasses the previous best approach with a Driving Score of 79.2 without scenarios and maintains a Driving Score of 47.6 with scenarios. Significantly, these results are achieved with considerably reduced computational requirements: only 1 A100 GPU compared to the model-free approach CaRL's 8 GPUs, 12.5 million simulation frames versus 300 million, and 16 CPUs versus 256 CPUs, within just 3 days of training instead of 7.

Acknowledgments

I thank my supervisors, Kashyap Chitta and Bernhard Jaeger, for their help and weekly discussions throughout this thesis. Thanks to Katrin Renz for introducing me to CARLA at the beginning, which made this thesis even possible in the first place. I thank Fabrice von der Lehr and Daniel Kerezsy for proofreading this thesis. Also, thanks to Professor Andreas Geiger for allowing me to be part of the autonomous vision group (AVG) for almost two years as a student assistant (Hiwi), conducting my research project and this master's thesis.

On a personal note, I would like to thank my parents for their support throughout my studies. Lastly, I thank Vanessa for her love, patience, and constant support.

Contents

1	Introduction	11
2	Related Work	13
2.1	Reinforcement Learning	13
2.1.1	Model-free reinforcement learning	13
2.1.2	Model-based reinforcement learning	14
2.2	Data collection	16
2.2.1	Real-world data collection	16
2.2.2	Autonomous Driving Simulators	16
2.2.3	Expert planners in CARLA	16
3	Methods	19
3.1	DreamerV3	19
3.1.1	Overview	19
3.1.2	World Model	20
3.1.3	Actor-Critic	23
3.2	CaRL-Lite	24
3.2.1	Coupling CARLA and DreamerV3	25
3.2.2	Route Processing and Precalculation	26
3.2.3	Data Collection	26
3.2.4	Input Representation	26
3.2.5	Action commands	30
3.2.6	Architectural Enhancements	30
3.2.7	Reward Function	32
3.2.8	Stop-gradient Operator	37
3.2.9	Training	37
4	Experiments	39
4.1	Routes	39
4.2	Evaluation Metrics	40
4.3	Baselines	41
4.4	Model Checkpoint Selection	44
4.5	Ablation Studies	45
4.6	Scenarios	48

Contents

5	Discussion	53
5.1	World Model Evaluation	53
5.2	Future Research Directions	54
5.3	Conclusion	56

1 Introduction

The traditional approach to autonomous driving systems has relied on complex modular pipelines with hand-crafted components for perception, prediction, planning, and control. Although these modular pipelines provide interpretability, they suffer from error propagation between modules and require extensive engineering to achieve satisfactory performance. This problem has encouraged manufacturers to adopt imitation learning (IL) with end-to-end trainable models. Unlike modular systems, end-to-end trainable methods prevent the accumulation of errors between components. However, IL models face two significant limitations in closed-loop inference: they require large labeled datasets for training and tend to compound errors as small errors accumulate until the system encounters unfamiliar situations [CWC⁺24].

Reinforcement learning (RL) [SB⁺98, JG⁺24] is a promising alternative that trains end-to-end policies through trial and error while generating their own training data. Training in closed-loop simulations [ZGZ⁺22] enables these policies to recover from mistakes and perturbations, thus avoiding the compounding error problem. Since RL policies use neural networks that can run in parallel on hardware accelerators such as GPUs, inference requires only milliseconds. Recent advances in model-based reinforcement learning [HLBN19, HLN20, HPBL23, HSW23, SAH⁺20] made RL an attractive solution for autonomous driving without requiring enormous effort to accelerate simulators for RL training.

This thesis introduces a model-based reinforcement learning method for autonomous driving for CARLA Leaderboard 2.0 [DRC⁺17]. We build on the DreamerV3 [HPBL23] architecture with several modifications. For example, we provide 2.6 times greater encoding capacity than the largest DreamerV3 model with 400 million parameters, while using only 133 million parameters but instead increasing the size of the latent space. Our experiments show that these modifications are crucial to perform well in scenarios that involve multiple objects, which is a limitation of the original architecture [ZWS⁺23]. In addition, we develop a multiplicative reward function to enable efficient and quick training. This reward function outperforms other reward functions that often lead the models to local minima. We employ a BEV renderer that can achieve up to 8000 frames per second (FPS) and produce multi-class object segmentation masks. We focus on scenarios where the model previously failed and employ an image gradient stopping approach that was previously not considered in this domain.

Our model performs well on the longest6 v2 routes with and without scenarios, requiring significantly fewer computational resources than other approaches. We achieve a new state-of-the-art DS of 79.2 on longest6 v2 without scenarios, surpassing the current best method PDM-Lite [SRC⁺24]. On longest6 v2 with scenarios, we achieve a DS of 47.6 while training for only three days with a single A100 GPU and 16 logical CPU cores, which is 8 times fewer GPUs and 16 times fewer CPU cores than CaRL [JDB⁺25], the current leading RL algorithm on longest6 v2 with scenarios. We also reproduced Think2Drive [LJWY24b], which claims to solve all CARLA Leaderboard 2.0 scenarios. Think2Drive’s code has not been released. CaRL-Lite outperformed our reimplementation of Think2Drive by 40.5 DS on longest6 v2. Through comprehensive ablation studies, we demonstrate the importance and effect of each architectural and training improvement. These results indicate that model-based reinforcement learning can address autonomous driving without requiring massive computing resources. To support research on CARLA Leaderboard 2.0 with RL, we make our code publicly available on GitHub.

In Chapter 2, we discuss related research that focuses on model-based reinforcement learning for autonomous driving. Chapter 3 describes the original DreamerV3 architecture and CaRL-Lite. We describe the BEV renderer, input representation, model changes, reward function, and training. Next, Chapter 4 ablates all added components and the reward function and compares CaRL-Lite with CaRL, Think2Drive, and PDM-Lite. Chapter 5 concludes this thesis by discussing the data generation capabilities of the world model and suggesting future research directions.

2 Related Work

This chapter discusses the existing literature on reinforcement learning (RL) and focuses on model-based RL. We begin by describing the two dominant paradigms of RL in Section 2.1. Next, Section 2.2 discusses different types of data collection, especially expert planners designed for the CARLA simulator.

2.1 Reinforcement Learning

Reinforcement learning has become a powerful method for developing agents in various fields, including games [HPBL25, VBC⁺19, SAH⁺20], robotics[RSG⁺25, SKS22], and autonomous driving [LJWY24b, KST⁺21, CTHH⁺25, JDB⁺25, ZLD⁺21b, GK24]. RL agents learn by interacting with their environment and aim to maximize the discounted sum of rewards (Equation (2.1)).

$$\text{return} = \sum_t \gamma^t r_t \quad (2.1)$$

The RL paradigm is formally represented as a Markov Decision Process (MDP), defined by the tuple (S, A, P, R, γ) [SB⁺98]. Here, S represents the state space, A denotes the action space, P is the transition probability function, R is the reward function and γ is the discount factor. The reward function provides a single number that indicates how well the agent performs in solving a task. This reward function is not bound by any constraints and enables RL algorithms to achieve superhuman performance in various domains.

RL algorithms can be broadly categorized into two main paradigms: model-free reinforcement learning and model-based reinforcement learning. In the following, we focus exclusively on deep reinforcement learning, which utilizes neural networks for function approximation.

2.1.1 Model-free reinforcement learning

Model-free RL algorithms learn policies directly from experience without estimating the dynamics of the environment. These approaches have dominated most of the breakthroughs in RL.

Mnih et al. [MKS⁺15] were the first to use deep neural networks with Q-learning by introducing **Deep Q-Networks** (DQN) to address high-dimensional control problems. Their research showed impressive performance in various Atari 2600 games using only raw pixel input as observations. Prior to this, RL algorithms were mainly limited to low-dimensional domains. DQN introduced several improvements, such as the replay buffer and target networks, which stabilized the training process and laid the foundation for modern deep reinforcement learning.

Based on DQN, Hessel et al. [HMHV⁺18] developed **Rainbow**, which combined multiple improvements, including Double Q-Learning, a replay buffer with prioritization, and distributional RL. With these changes, Rainbow achieved state-of-the-art performance on the Atari benchmark.

For problems requiring continuous control, Schulman et al. [SWD⁺17] proposed **Proximal Policy Optimization** (PPO), which uses a clipped objective to ensure stable policy updates. Similarly, Haarnoja et al. [HZH⁺18] proposed **Soft Actor-Critic** (SAC), an off-policy algorithm that maximizes expected return and policy entropy.

Despite these advancements, model-free RL is limited by sample inefficiency and often requires millions or billions of interactions with the environment to achieve good performance. Therefore, it needs a fast simulator, which is often the bottleneck in RL for autonomous driving.

2.1.2 Model-based reinforcement learning

Model-based RL extends model-free approaches by also training a world model that learns to simulate the environment. This world model encodes and compresses data for the actual RL agent and allows it to plan by simulating actions. In addition, world models are used to generate much more experience, which addresses the sample inefficiency of model-free RL and reduces the required number of environment interactions.

Early Model-based RL development The first model-based RL algorithm, as we know it today, was developed by Ha et al. [HS18]. They built a probabilistic generative model for OpenAI Gym environments to encode the high-dimensional input into a more expressive low-dimensional space. The main intention behind their approach was to reduce the number of parameters trained with RL. Therefore, they developed a world-model that contains most of the parameters and encodes the input temporally and spatially.

Hafner et al. [HLF⁺19] extended this work and developed **PlaNet** (Planning Network), which learned to simulate the environment in the latent space using pixels. Instead of only using a deterministic state, PlaNet also employed a stochastic state, resulting in the Recurrent State Space Model (RSSM). To select actions, it used the Cross-Entropy Method (CEM). PlaNet achieved great performance on visual control

tasks while requiring orders of magnitude fewer environment interactions than model-free approaches.

Dreamer family The Dreamer family represents a significant advancement in model-based RL in terms of performance, sample efficiency, and model scaling. The first Dreamer version **DreamerV1**, [HLBN19] extended PlaNet by adding a policy and value model that were only trained on simulated trajectories. This step made explicit planning redundant and made action selection more efficient while improving performance.

DreamerV2 [HLNB20] improved the architecture by replacing Gaussian latent variables with Categorical latent variables. Categorical latent variables can capture multimodal distributions and learn the complex dynamics of environments. It enabled Dreamer to surpass many model-free approaches, including Rainbow on Atari and DeepMindSuite.

The latest version **DreamerV3** [HPBL23] made the world-model more robust and applicable to large scales of input data and rewards. Without changing its hyperparameters, DreamerV3 achieved great performance in 150 diverse tasks, surpassing even specialized models.

Other model-based approaches In addition to Dreamer, several other algorithms have achieved state-of-the-art performance using different model architectures. DeepMind’s **AlphaZero** [SHS⁺17] used explicit tree-based planning for action selection but still relied on a hand-programmed world model. Its successor, **MuZero** [SAH⁺20] extended it by adding a trainable world model and achieved state-of-the-art performance on the Atari benchmark at that time. However, it was still inefficient and required billions of frames. That changed with the advent of **EfficientZeroV1** [YLK⁺21] and **EfficientZeroV2** [WLY⁺24]. They incorporated self-supervised objectives into MuZero, which greatly improved sample efficiency. EfficientZeroV2 achieves state-of-the-art performance on the Atari100K benchmark, which limits the total environment frames to 2 hours of real-time gameplay. These improvements require increased computing resources, and have not been evaluated much beyond the limited data regime of around 400,000 frames.

Recently, **TD-MPC2** [HSW23] was published, which achieved great results in continuous control tasks. Unlike Dreamer, which relies heavily on image reconstruction, TD-MPC2 uses an objective that combines reward prediction, value estimation, and latent consistency losses. For action selection, TD-MPC2 uses the CEM similar to PlaNet. However, it has not been evaluated on complex benchmarks such as the Atari benchmark and does not work on discrete action spaces.

2.2 Data collection

Data collection is a fundamental step in the development of autonomous driving systems. However, collecting data from the real world requires an expensive amount of manual annotation, making it expensive. In research, simulators such as CARLA [DRC⁺17] have emerged as cheap alternatives that allow automated data collection with precise ground-truth labels without requiring manual annotation.

2.2.1 Real-world data collection

An example of a real-world data collection is nuScenes [CBL⁺19], which comprises 15 hours of data from 1,000 scenes in Boston and Singapore, labeled by human experts. To employ autonomous systems, the Sim2Real problem requires real-world data to ensure that the algorithm works as intended. Due to the vast data requirements for RL real-world data is not an option for this thesis, which is why we stick to simulators.

2.2.2 Autonomous Driving Simulators

There exist multiple simulators for the research of autonomous driving [KPC⁺24, DRC⁺17, GFL⁺23, LYZ⁺24]. We selected **CARLA** for this work due to its realistic physics engine and diverse scenarios, enabling training with safety-critical scenarios, and support for closed-loop simulation of long routes. Alternative simulators such as **nuPlan** [HC21] use scenarios initialized from real-world recordings but are constrained to only 15 seconds of simulation. Similarly, **Waymax** also uses scenarios initialized from real-world recordings and accelerates computation with JAX and hardware accelerators such as GPUs and TPUs, but is still limited to short simulation durations. Due to the mentioned downsides and since we do not need a fast simulator because we are using model-based RL, we decided on CARLA Leaderboard 2.0.

2.2.3 Expert planners in CARLA

PDM-Lite Rule-based expert planners are the predominant approach to collect data in CARLA, with **PDM-Lite** [SRC⁺24] being state-of-the-art in CARLA Leaderboard 2.0. PDM-Lite employs the kinematic bicycle model and the Intelligent Driver Model (IDM) to forecast vehicles, and based on these forecasts, it selects actions. For some scenarios, it accesses their specifications and follows a pre-coded procedure to solve them. However, this rule-based approach forces PDM-Lite to categorize every situation as route following or a predefined scenario, which prevents PDM-Lite from solving everyday traffic situations. When encountering newly implemented scenarios or facing unfamiliar situations and scenario configurations, PDM-Lite may fail, resulting in fatal accidents. Similarly to other rule-based expert planners, PDM-Lite cannot deviate from the planned route unless it is in a predetermined scenario.

This results in dangerous situations where it cannot change lanes due to obstruction of vehicles, which sometimes requires stopping on highways. Furthermore, it always assumes access to ground-truth data and cannot cope with occluded objects. These limitations motivated us to develop an RL-based expert planner that only needs training to solve all these situations.

Think2Drive Think2Drive [LJWY24b] is a similar approach to CaRL-Lite that uses DreamerV3 to solve CARLA Leaderboard 2.0. However, unlike our method, it uses DreamerV3 out of the box without making any architectural modifications. Think2Drive similarly trains on automatically generated routes with scenarios and applied prioritized sampling of observations close to the end of failed routes. The training protocol consists of two steps: (1) they use easy routes without scenarios, requiring only route following; (2) afterward, they switch to more complicated routes with scenarios. Excluding the steering cost, their reward function is very similar to Roach’s, which we find does not work well in certain situations, as described in Section 4.5. Think2Drive collects 2 million CARLA frames in 3 days for training and requires 128 logical CPU cores. Our model only uses 16 logical CPU cores, which is 8 times less, and collects 12.5 million frames within the same training time. Since its authors did not release the source code, the training routes, many required hyperparameters, and their evaluation routes, we reproduce Think2Drive, evaluate it on longest6 v2, and compare it to our method.

CaRL The state-of-the-art performance of RL agents on the CARLA Leaderboard 2.0 is held by **CaRL** [JDB⁺25]. This PPO-based agent builds on Roach [ZLD⁺21b], which was previously the leading open source PPO method for CARLA Leaderboard 1.0. CaRL’s authors significantly optimized the Leaderboard 2.0 code and parallelized training, allowing them to collect 300 million samples within one week of training. This optimization enabled their agent to train solely using CARLA frames. Instead of rewarding the agent for waiting, such as at red traffic lights, they mainly reward route completion. They scale route completion by soft penalties, which are bounded between 0 and 1, including comfort, route deviation, route adherence, speeding, and time to collision. We improve on their approach by using model-based RL and developing a denser and more informative reward function to make it more efficient. This reduces the number of required logical CPU cores from 256 to only 16 logical CPU cores, and reduces the required 8 GPUs to only 1 GPU. Furthermore, we use CaRL’s optimized Leaderboard 2.0 implementation and their automatically generated routes with scenarios for training.

Other expert planners Kazemkhani et al. and Cusumano-Towner et al. published **GPUDrive** [KPC⁺24] and **GIGAFLow** [CTHH⁺25], respectively, which are two works that accelerate autonomous driving simulators using GPUs. While GPUDrive

builds on Waymax [GFL⁺23] and the Waymo Open dataset, GIGAFLOW approximates the CARLA simulator and uses its maps for training. This allowed them to achieve approximately 1 million frames per second. GPU Drive achieved a good policy, but was never tested it on other simulators like Waymax, which makes it difficult to assess its capabilities on simulators that do not approximate agents' behaviors. Although GIGAFLOW reaches good performance, it requires a trillion training samples, making it extremely inefficient. In contrast, our RL agent learned to drive safely using only 12.5 million frames from the CARLA simulator and 6 billion samples generated by the world model. Additionally, our agent also used only 1 GPU while GIGAFLOW requires 8.

3 Methods

The following chapter describes our model-based reinforcement learning (RL) algorithm, CaRL-Lite, developed for CARLA Leaderboard 2.0. We begin by introducing the problem setting and explain the task within the simulation. Next, we elaborate on the model-based RL algorithm DreamerV3 [HPBL25] in Section 3.1, which serves as the foundation of our work. We provide a detailed examination of DreamerV3’s world model architecture, training methodology, and actor-critic framework in detail to provide context for our modifications. Finally, in Section 3.2, we present CaRL-Lite, including architectural improvements, data collection, input representation, and reward function design.

Problem setting The objective is to navigate a pre-defined route in the CARLA simulator. Each route consists of a sequence of checkpoints spaced approximately 1 meter apart. Routes with scenarios include seven different scenario types, which do not require the agent to deviate from the route. Although occasional lane changes may necessitate temporary route deviations, the primary task involves following the designated path while avoiding collisions and adhering to traffic regulations.

3.1 DreamerV3

This section discusses the DreamerV3 architecture that forms the the basis for our model. DreamerV3 [HPBL23] represents the latest iteration in the Dreamer family [HLBN19, HNLNB20, HPBL23]. We chose DreamerV3 for its computational efficiency, requiring only hours to a few days of training on a single A100 GPU, and its effectiveness across various benchmarks, making it suitable for autonomous driving.

3.1.1 Overview

DreamerV3 consists of two main components: a world model and an actor-critic. The world model is trained through self-supervised learning with collected experiences. It encodes high-dimensional data into compact latent representations. This compression allows for efficient parallel forecasting, enabling the actor-critic to learn from many simulated experiences, which reduces environment interactions.

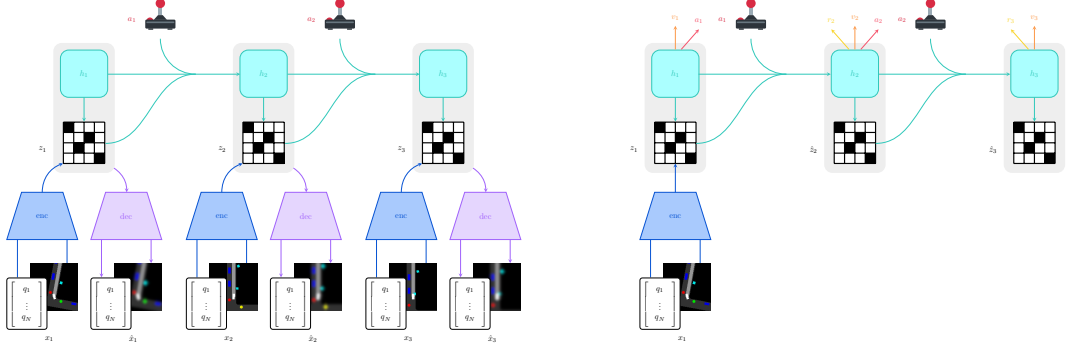


Figure 3.1: Left: The DreamerV3 world model unrolled for three timesteps, illustrating the data generation in latent space. Right: The actor-critic operating in latent space for reinforcement learning through generated trajectories. Images taken from [HPBL23].

DreamerV3 has achieved great results across 150 diverse tasks, including benchmarks like Atari, Atari100K, DMLab, and Minecraft. It has reached state-of-the-art performance in multiple domains while using fixed hyperparameters. This versatility applies to environments with both dense and sparse rewards, high-dimensional visual input, and low-dimensional state representations.

An advantage of the DreamerV3 architecture, in contrast to methods that rely solely on rewards, estimated returns, or model-free approaches, is its image decoder. This decoder enables the visualization of the agent’s internal representations. During our experiments, this feature proved to be essential for identifying problems and developing solutions based on observed failure cases.

3.1.2 World Model

The DreamerV3 [HPBL25] world model employs a recurrent state-space model (RSSM) that generates latent space trajectories conditioned by observations and actions. This model consists of six integrated components, formalized as:

$$\text{RSSM} \begin{cases} \text{Sequence model:} & h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\ \text{Encoder:} & z_t \sim q_\phi(z_t | h_t, x_t) \\ \text{Dynamics predictor:} & \hat{z}_t \sim p_\phi(\hat{z}_t | h_t) \\ \text{Reward predictor:} & \hat{r}_t \sim p_\phi(\hat{r}_t | h_t, z_t) \\ \text{Continue predictor:} & \hat{c}_t \sim p_\phi(\hat{c}_t | h_t, z_t) \\ \text{Decoder:} & \hat{x}_t \sim p_\phi(\hat{x}_t | h_t, z_t) \end{cases} \quad (3.1)$$

World Model Components The **Encoder** transforms an observation x_t and a recurrent state h_t into a stochastic latent state z_t . While h_t captures the deterministic dynamics, z_t represents the stochastic elements and information missing in h_t . The Encoder transforms images and scalar observations, merges them, and produces logits for z_t .

Since DreamerV2 [HLNB20], this stochastic representation uses a vector of softmax distributions that consists of 32 one-hot vectors, each with, e.g., 48 classes. After computing the probabilities using the softmax function, Dreamer samples from these vectors and applies a straight-through estimator to ensure that gradient flow is maintained through the discrete sampling operation.

$$z_t = \text{probs} + \text{sg}(\text{sample}(\text{probs}) - \text{probs}) \quad (3.2)$$

where $\text{sg}(\cdot)$ denotes the stop-gradient operator. Image inputs are normalized to the range $[0, 1]$, and scalar observations are transformed using the symmetric logarithm $\text{symlog}(x) = \text{sign}(x) \log(|x| + 1)$, which scales large inputs to a more appropriate range.

The **Sequence model** uses a modified Gated Recurrent Unit (GRU) to transition between timesteps. It processes the previous action along with the latent representation, which includes h_{t-1} and z_{t-1} , to generate the next recurrent state h_t . This component acts as the model’s memory mechanism storing information across multiple timesteps.

The **Dynamics Predictor** estimates the stochastic state \hat{z}_t using only the recurrent state h_t . It’s purpose is to approximate the distribution that the encoder would generate based on the corresponding observation. This generative ability allows the model to simulate possible scenarios without requiring actual observations. Together, the dynamics predictor, the sequence model, and the encoder form what is known as the RSSM.

The **Reward predictor** estimates expected rewards based on the latent representation by processing the combined vectors h_t and z_t .

The **Continue predictor** outputs a binary value indicating whether the current timestep ends the episode. DreamerV3 differentiates between episode termination (ending due to success or failure) and truncation (ending due to reaching the maximum time limit).

The **Decoder** reconstructs the original observation from the latent representations h_t and z_t .

The encoder and decoder utilize convolutional neural networks (CNNs) and multi-layer perceptrons (MLPs). The dynamics, reward, and continuation predictors use MLPs. The sequence model is built with a slightly modified GRU. In the official implementation, the architecture is implemented in JAX [BFH⁺18].

World Model Training The world model is trained in a self-supervised manner using trajectories of length T sampled from the replay buffer and employs a multi-task loss function:

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi} \left[\sum_{t=1}^T \left(\beta_{\text{pred}} \mathcal{L}_{\text{pred}}(\phi) + \beta_{\text{dyn}} \mathcal{L}_{\text{dyn}}(\phi) + \beta_{\text{rep}} \mathcal{L}_{\text{rep}}(\phi) \right) \right] \quad (3.3)$$

with the scaling coefficients being $\beta_{\text{pred}} = 1$, $\beta_{\text{dyn}} = 1$, and $\beta_{\text{rep}} = 0.1$.

Prediction loss The prediction loss $\mathcal{L}_{\text{pred}}$ includes several loss functions for the reconstruction of observations, rewards, and continuation flags:

$$\mathcal{L}_{\text{pred}}(\phi) = -\log p_\phi(x_t | z_t, h_t) - \log p_\phi(r_t | z_t, h_t) - \log p_\phi(c_t | z_t, h_t) \quad (3.4)$$

For image reconstruction, the model applies the mean squared error (MSE) loss. For scalar values, it calculates the MSE loss after transforming the target values using the symmetric logarithm:

$$\mathcal{L}(\theta) = \frac{1}{2} (f(x, \theta) - \text{symlog}(y))^2 \quad (3.5)$$

This transformation scales values close to zero linearly and larger values logarithmically, allowing the model to use large input values. This scaling is required to make RL applicable to multiple RL problems with different I/O scales.

Two-Hot Encoding for Value Regression To decouple gradient magnitudes from underlying value scales, the reward predictor and the critic use the cross-entropy loss with two-hot encoding, allowing regression of continuous values within the range of $\text{symexp}(-20)$ to $\text{symexp}(20)$ without gradient instabilities:

$$\mathcal{L}(\theta) = \text{twohot}(y)^T \log \text{softmax}(f(x, \theta)) \quad (3.6)$$

The predicted value is computed as:

$$\hat{y} = \text{softmax}(f(x))^T B \quad (3.7)$$

where $B = \text{symexp}([-20, \dots, +20])$ represents the exponentially spaced bin centers, and $\text{symexp}(x) = \text{sign}(x)(\exp(|x|) - 1)$ the symmetric exponential function.

Dynamics and Representation Losses To match the predicted stochastic states with those derived from actual observations, DreamerV3 uses the Kullback-Leibler (KL) divergence between the output of the encoder and the dynamics predictor. In addition, it scales the gradients for both components differently.

$$\mathcal{L}_{\text{dyn}}(\phi) = \max\left(1, \text{KL}\left[\text{sg}(q_\phi(z_t | h_t, x_t)) \parallel p_\phi(z_t | h_t)\right]\right) \quad (3.8)$$

$$\mathcal{L}_{\text{rep}}(\phi) = \max\left(1, \text{KL}\left[q_\phi(z_t | h_t, x_t) \parallel \text{sg}(p_\phi(z_t | h_t))\right]\right) \quad (3.9)$$

This approach encourages the dynamics predictor to align with the encoder’s distribution while encouraging predictable representations. To avoid trivial or degenerate solutions, "free bits" [KSJ⁺16] are implemented by capping the dynamics and representation losses to below 1 nat (approximately 1.44 bits).

Additionally, the model applies distribution smoothing by mixing neural network outputs with a uniform distribution (1% uniform and 99% network output), ensuring that the KL-divergence remains well behaved.

3.1.3 Actor-Critic

The actor-critic operates entirely within the latent space generated by the world model, learning from trajectories simulated by the world model. They utilize both the deterministic recurrent state h_t and the stochastic latent state z_t as inputs:

$$\text{Critic: } v_\psi(R_t | h_t, z_t) \quad \text{Actor: } a_t \sim \pi_\theta(a_t | h_t, z_t) \quad (3.10)$$

Critic Network The critic v_ψ approximates the value function by estimating the expected discounted return $R_t = \sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau}$ for any given latent state (h_t, z_t) . To improve learning stability, DreamerV3 implements several mechanisms:

- **λ -returns:** The critic employs bootstrapped λ -returns [ARS18], calculated recursively: $R_t = \sum_{\tau=0}^{\infty} \lambda^\tau r_{t+\tau}$ where c_t represents the continuation flag (0 for terminal states, 1 otherwise), and $\lambda \in [0, 1]$ the discount factor balances between Monte Carlo returns and one-step TD estimates.
- **Exponential Moving Average Regularization:** The critic’s parameters are regularized to an exponential moving average of its historical parameters to reduce oscillations during training.
- **Two-Hot Encoding:** The critic employs the two-hot encoding described earlier with exponentially spaced bins, allowing for value estimation across multiple orders of magnitude. This is necessary to operate on different environments with different reward scales.

Actor The actor network π_θ learns a policy that maximizes the expected discounted return by transforming latent states into probability distributions of all action classes. In our implementation, we utilize the discrete action variant, although DreamerV3 supports both continuous and discrete action spaces.

The actor employs the REINFORCE estimator [Wil92] with several changes:

- **Return Normalization:** To stabilize learning with a fixed entropy scale, returns are normalized by $\max(1, S)$, where S represents a scale factor:

$$S = \text{EMA}(\text{Per}(R_t^\lambda, 95) - \text{Per}(R_t^\lambda, 5), 0.99) \quad (3.11)$$

This normalization uses the exponential moving average of the difference between the 95th and 5th percentiles of returns, making it robust to outliers. To prevent noise amplification due to small values and reward approximations, the denominator is limited to a minimum of 1.

- **Entropy Regularization:** To sustain exploration throughout training, the actor’s objective incorporates an entropy term to avoid one-hot-like action distributions:

$$\mathcal{L}(\theta) = - \sum_{t=1}^T \text{sg} \left(\frac{R_t^\lambda - v_\psi(h_t, z_t)}{\max(1, S)} \right) \log \pi_\theta(a_t | h_t, z_t) + \eta H[\pi_\theta(a_t | h_t, z_t)] \quad (3.12)$$

where η controls the strength of the entropy regularization.

Inference During inference, DreamerV3 processes the current observation x_t with its encoders to obtain the posterior representation z_t . Afterwards, the RSSM updates h_t based on this encoded observation and the applied action. The updated hidden state h_t and the sampled stochastic state z_t are used as input for the actor, which outputs a probability distribution on the actions. For both evaluation and training, the final action is sampled to ensure exploration while training and to avoid a distribution shift during evaluation. Unlike planning-based approaches that require intensive rollouts, DreamerV3’s inference is computationally efficient and requires only a single forward pass of the model.

3.2 CaRL-Lite

This section describes CaRL-Lite with the modifications of the original DreamerV3 architecture to enhance its performance for autonomous driving on the CARLA Leaderboard 2.0. Figure 3.2 illustrates our modified architecture. CaRL-Lite employs bird’s-eye-view (BEV) images with object masks and scalar observations as input representation, adapted encoders, separate stochastic latent distributions for visual

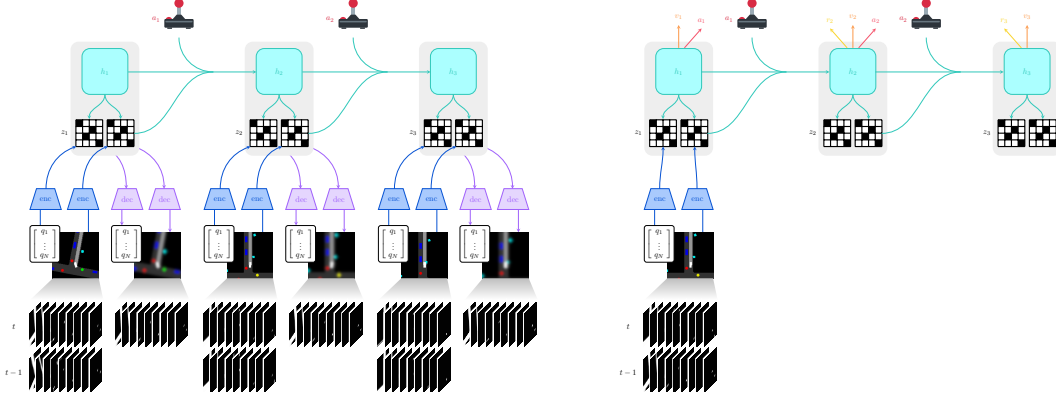


Figure 3.2: The modified DreamerV3 architecture, optimized for the CARLA Leaderboard 2.0. Left: The world model utilizing separate stochastic latents for visual and scalar inputs, with dedicated encoders for each modality. The visual input has 18 channels, which include object masks for the current and previous timesteps; however, only the masks for the current timestep are decoded. Right: The actor-critic training on generated trajectories in latent space.

and scalar data, our multiplicative reward function, and changes in the training methodologies.

3.2.1 Coupling CARLA and DreamerV3

We encountered a major challenge when integrating DreamerV3, which requires Python 3.11, with CARLA, which is compatible only with Python 3.8. To address this compatibility issue while ensuring stable training despite simulator crashes, we developed a dual-environment solution using separate Anaconda environments, each with its required Python version.

Our implementation includes a custom agent class for the DreamerV3-side and another agent for the CARLA simulator. These components communicate information through inter-process communication (IPC) to exchange action commands, reset signals, and observations. These observations comprise BEV images, scalar measurements, rewards, and termination flags, which are processed on the CPU within the CARLA agent before sending them to the Dreamer agent.

For later analysis, we record episode termination causes whenever an agent commits an infraction. Furthermore, we store terminated episodes as videos categorized by their termination cause.

3.2.2 Route Processing and Precalculation

The standard route representation in CARLA Leaderboard 2.0 consists of irregularly spaced waypoints, sometimes varying by up to 6 meters. This inconsistency makes reward calculation more complicated. To resolve this issue, we preprocess the route using SciPy’s [VGO⁺20] interpolation methods to generate uniformly spaced waypoints at 10-centimeter intervals. These refined checkpoints allow computing accurate route deviation and the driven distance without simulator access.

Furthermore, we pre-calculate the lane orientations at each checkpoint, which are necessary to render the BEV images described in Section 3.2.4.

3.2.3 Data Collection

Our data collection leverages DreamerV3’s parallel agent implementation by running six concurrent processes, each with its own CARLA simulator instance one for towns 1 to 6. This reduces the frequency of loading different towns, which can be computationally expensive. As a result, we achieve an aggregate throughput of approximately 50 frames per second, which results in about 8 frames per environment per second.

The coupling between training and data collection ensures a consistent train ratio, which is the average number of times each sample is used during training. This can lead to frame rate fluctuations between 20 and 90 FPS, depending on training dynamics and simulator stability. With this setup, a 20-hour training session collects approximately 3.5 million CARLA frames, while a 3-day training session accumulates approximately 12.5 million (25 million with action repeat 2) total frames. For comparison, the official DreamerV3 model achieves 50 million frames (200 million with action repeat 4) in 7.7 days but uses 16 environments, while we are only using 6 environments.

Episodes shorter than 10 frames, those terminated due to faulty routes, or episodes ending because of simulator crashes are discarded.

3.2.4 Input Representation

Our approach utilizes bird’s eye view (BEV) images as visual input, supplemented by scalar measurements. We intentionally reduced the number of total image channels to focus on the most relevant ones. As a result, we omit some information such as road lines.

Visual Input The visual input consists of 18-channel BEV images with a resolution of 128×128 pixels. The first nine channels represent the current simulator state, while the remaining are the object masks from the previous timestep. This provides

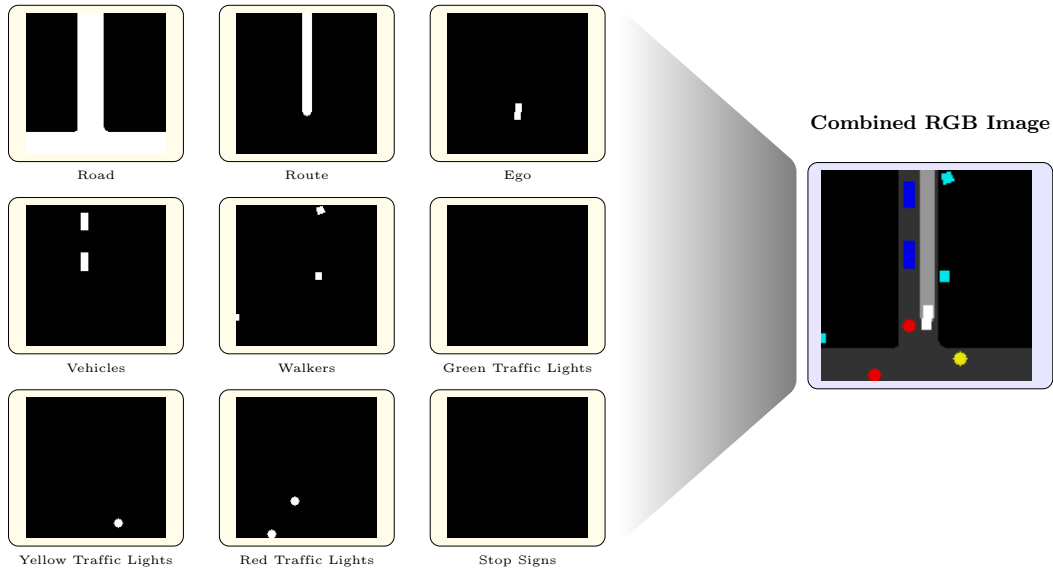


Figure 3.3: Left: All 9 image channels of one single timestep displayed as individual binary masks, which are used as input for the encoder and as output for the decoder. Right: The combined visualization as an RGB image, which we use throughout this thesis.

temporal context in case the GRU misses temporal information. Unlike the original DreamerV3 architecture, we only decode the masks for the current timestep rather than all the input masks. For presenting visual results in this thesis, Figure 3.3 illustrates the individual channels and demonstrates how we combine them to create an RGB image.

Each channel represents a specific object type:

1. **Road:** Drivable area, including opposite lanes
2. **Route:** Route, rendered with 3m width
3. **Ego:** The ego vehicle rendered as a rotated bounding box
4. **Vehicles:** Other vehicles, including emergency vehicles and bicycles
5. **Walkers:** Pedestrians
6. **Red Traffic Lights**
7. **Yellow Traffic Lights**
8. **Green Traffic Lights**
9. **Stop Signs**

The BEV image is translated so that the ego vehicle is positioned at 70% of the image height from the top edge. Additionally, the BEV image is rotated according to the orientation of the route rather than the ego vehicle’s heading. This ensures smooth transitions between consecutive frames and prevents rapid rotational changes that could occur due to the discrete action space.

To overcome DreamerV3’s limitation to generate trajectories with small objects [ZWS⁺23], we optimized the pixels-per-meter (ppm) parameter while maintaining a sufficient visual range. We expect a maximum required speed of 72 km/h, as used by PDM-Lite [SRC⁺24, ZBJ⁺24] to calculate the required braking distance:

$$\text{braking distance} = 0.5 \cdot \left(\frac{72}{10}\right)^2 = 25.92 \text{ [m]} \quad (3.13)$$

This allowed us to compute the optimal PPM parameter:

$$\text{ppm} = \frac{\text{bev size} \times (1.0 - \text{ego vertical percentage})}{\text{braking distance} + 2 \times \text{ego length}} - 0.1 = 2.8 \quad (3.14)$$

To enhance the perception of potentially small objects, we render all traffic participants (pedestrians and vehicles) with a minimum width of 2 meters. Traffic lights and stop signs are depicted as filled circles to ensure rotational invariance, which simplifies encoding and decoding.

Renderer We developed a custom BEV image renderer to maximize data collection efficiency. This renderer achieves between 2,000 and 8,000 frames per second, depending on the scene complexity.

Our renderer consists of two phases: preprocessing and runtime rendering. During the preprocessing phase, we parse each map’s OpenDrive format file and transform the road into polygons. This initial transformation produces hundreds of thousands of individual polygons, with at least one polygon per lane on each road. We merge and simplify these polygons using Shapely [GBLT07] to reduce their amount. Although this process substantially reduces the polygon count, some resulting polygons can be large, with some spanning the entire map, which could negatively impact rendering performance. To make rendering efficient, we divide each map into a grid with 64×64 meter cells and calculate the intersections between polygons and these grid cells. For each cell, we store the list of intersecting polygons in a dictionary for quick access during training. We further simplify the polygons for each cell using Shapely’s functions. Shapely represents complex polygons using linear rings, which may include holes that cannot be directly rendered by OpenCV. Therefore, we split polygons with inner linear rings vertically down the center of each ring to create polygons without holes. Additionally, we pre-calculate the locations of stop signs

and traffic lights and store them in a cKDTree for efficient spatial querying during training.

During the rendering phase, we identify which grid cells are visible in the current BEV image and render only the polygons of these visible cells using OpenCV [Bra00]. Typically, this means rendering fewer than ten polygons. We retrieve stop signs and traffic lights from the pre-calculated cKDTree based on their proximity to the ego vehicle. Vehicles and pedestrians are filtered based on distance and height, with a maximum allowable height difference of 8 meters compared to the ego vehicle.

This height-based filtering is essential because some CARLA scenarios, which are only triggered under certain conditions based on proximity, place vehicles below the map (100 meters underground). Without this filtering, these vehicles might incorrectly appear as obstacles in the BEV image, potentially confusing the agent.

Image Compression To efficiently store multi-channel BEV images, we use binary compression by allocating one bit per channel, resulting in binary pixel values of $\{0, 1\}$. While this discretization slightly complicates the forecasting of object positions, it significantly reduces the required memory. This compression allows us to use a replay buffer capacity of up to 1,500,000 samples.

Scalar Measurements Along the visual input, we provide 15 scalar measurements for the current and previous timesteps. Similarly to the visual data, only the current timestep values are decoded. These measurements are:

- (1) Velocity
- (2) 80 % of the speed limit
- (3) Previous steer command
- (4) Previous throttle command
- (5) Previous brake command
- (6) Distance of the vehicle's front to the route's centerline in meters (positive for right-side deviation, negative for left)
- (7) Distance of the vehicle's center to the route's centerline in meters (positive for right-side deviation, negative for left)
- (8) Distance of the vehicle's back to the route's centerline in meters (positive for right-side deviation, negative for left)
- (9) Distance to the nearest yellow / red light (maximum is 30 meters)
- (10) Distance to the nearest stop sign (maximum is 30 meters)

Throttle	Brake	Steer	Throttle	Brake	Steer	Throttle	Brake	Steer
0	1	0	0.3	0	-0.7	0.3	0	0.7
0.7	0	-0.5	0.3	0	-0.5	0	0	-1
0.7	0	-0.3	0.3	0	-0.3	0	0	-0.6
0.7	0	-0.2	0.3	0	-0.2	0	0	-0.3
0.7	0	-0.1	0.3	0	-0.1	0	0	-0.1
0.7	0	0	0.3	0	0	0	0	0
0.7	0	0.1	0.3	0	0.1	0	0	0.1
0.7	0	0.2	0.3	0	0.2	0	0	0.3
0.7	0	0.3	0.3	0	0.3	0	0	0.6
0.7	0	0.5	0.3	0	0.5	0	0	1

Table 3.1: The discrete actions used by CaRL-Lite.

- (11) Distance to the nearest leading vehicle (maximum is 30 meters). Leading vehicles are vehicles that are within 2 meters of the route.
- (12) Velocity of the leading vehicle
- (13) Remaining yellow light time of nearest traffic light (set to 3 seconds when not yellow)
- (14) Timeout term of the reward function
- (15) Angular difference between the route orientation and the vehicle heading

3.2.5 Action commands

We adopt the same 30 discrete action as Think2Drive [LJWY24a], as listed in Table 3.1. These actions cover most of the action space, though they exclude full-throttle or multiple braking values. Of the 30 actions, 29 combine acceleration and steering, while only one uses braking. Although this discretization excludes certain parts of the action space, our method aligns with existing research; for instance, PDM-Lite [SRC⁺24] similarly applied arbitrary throttle values, but only used full braking. Since action selection occurs at 10 Hz, this should be enough for vehicle maneuvers.

3.2.6 Architectural Enhancements

Our architecture extends the original DreamerV3 model to optimize it for CARLA Leaderboard 2.0. We expanded the model’s capacity and separated the encodings of different modalities.

Dual-Modal Stochastic state Unlike the original DreamerV3 architecture, which encodes visual and scalar data into a single representation, we maintain separate stochastic latent spaces for each modality. This ensures sufficient capacity to represent both visual and scalar information.

For scalar measurements, we employ 16 one-hot vectors, each containing 16 classes. Visual data is encoded with 512 one-hot vectors, also with 16 classes each. This configuration offers 2.6 times more representational capacity compared to the largest DreamerV3 variant (which has 400 million parameters).

Enhanced Encoders We preserve DreamerV3’s five-layer convolutional image encoder structure but change the final layer’s stride to two. This adjustment is necessary because of the larger input resolution of 128×128 , in contrast to DreamerV3 64×64 . We change the convolutional layers number of output channels to 32, 64, 128, 256, and 512. Furthermore, we add a residual convolutional block [TCE⁺22] after these 5 layers, which results in a visual encoder output dimension of $512 \times 4 \times 4$.

The scalar measurement encoder maintains the original MLP encoder but reduces the number of linear units from 768 (as used in the model size100m variant) to 128. This optimization leads to a scalar encoder output dimension of 128.

Separate Stochastic State Encodings Unlike DreamerV3, which combines image and scalar observations into a single encoding and reduces it to dimensionality 768 for the 100 million parameter variant, CaRL-Lite preserves the spatial dimensions of the encoded images ($512 \times 4 \times 4$) without further compression. We transform this spatial representation into one-hot vectors using convolutional layers and convolutional residual blocks. The final transformation generates a separate one-hot vector for each of the 512 channels using a shared linear layer.

We utilize BlockLinear layers, to efficiently transform the high-dimensional GRU state, which has a shape of 6144, into the required format of $512 \times 4 \times 4$. BlockLinear layers separate the weight matrices of linear layers into multiple blocks to reduce the number of parameters. However, an exception arises in the prior stochastic state calculation. In this case, we are using two multi-layer perceptrons (MLPs), rather than just one like in DreamerV3. Our experiments revealed that BlockLinear layers consistently underperformed in this specific component. This design choice leads to a parameter-intensive transformation from the 6144-dimensional GRU state to 768 units, and then back to $512 \times 4 \times 4$.

Enhanced Decoder Architecture The visual decoder processes each of the 512 one-hot vectors by passing them through a small shared linear layer, transforming them into a 4×4 spatial channel making it suitable for the subsequent convolutional layers. We include two convolutional residual blocks to handle the outputs from the GRU

and the transformed stochastic channels, followed by five transposed convolutional layers, as done in DreamerV3. These layers mirror the channel structure of the visual encoder in reverse order.

Similarly, the scalar decoder employs 128 units in each linear layer, aligning with the dimensionality of the scalar encoder. We further enhance its architecture by adding a residual block, which consists of two linear layers, normalization, and activation functions, before merging the GRU features with the stochastic state's one-hot vectors.

3.2.7 Reward Function

Reinforcement learning algorithms can, in theory, optimize any objective from various sparse reward functions. However, this often necessitates hundreds of millions or even billions of observations. To achieve strong performance with limited computational resources and training time, we utilize a carefully designed dense reward function that provides frequent feedback on the agent's actions. Nonetheless, creating dense reward functions that align with the intended behavior while avoiding exploitable loopholes poses significant challenges.

Consider the common approach of using weighted sums of reward terms (Equation Equation (3.15)):

$$\text{reward} = \sum_i^N \alpha_i \cdot r_i \quad (3.15)$$

This approach becomes problematic when dealing with multiple competing objectives. Even if one critical condition fails (for example, collision avoidance), the overall reward can remain high if other factors compensate for it. As the number of terms increases, it becomes increasingly challenging to adjust the appropriate weighting factors.

To overcome these limitations, we implement a multiplicative reward function (Equation Equation (3.16)), where each term is constrained to a range between 0 and 1.

$$\text{reward} = \prod_i^N r_i \quad (3.16)$$

This formulation guarantees that if any single condition is not fulfilled, the total reward will be zero. As a result, there is an urgent need for all conditions to be satisfied at the same time. The reward will only reach its maximum value when all terms achieve their highest values.

Our reward function incorporates five terms, each designed to encourage specific driving behaviors:

Speed Reward Term (r_1): This term encourages the agent to maintain appropriate speeds based on environmental context:

$$r_1 = \max\left(0, 1 - \frac{|v - v_{\text{target}}|}{\max(1, v_{\text{target}})}\right) \quad (3.17)$$

The behavior of the reward term for different target speeds is illustrated in Figure 3.4. This function calculates the absolute difference between the current velocity v and the target velocity v_{target} . It then normalizes this difference by dividing it by the greater value between 1.0 and v_{target} . By capping the denominator at 1.0 for low target speeds, we ensure that the reward term is non-zero for a reasonable speed range.

The target speed is typically set to 80% of the speed limit. However, when obstacles are detected in the ego vehicle’s path, we compute a safer target speed as 70% of the maximum velocity that still allows for timely braking. This maximum velocity is derived from the required braking distance equation $b = (v \cdot 3.6/10)^2/2$ as presented by [Jae21]:

$$v_{\text{target}} = 0.7 \cdot \sqrt{\max\left(0, \frac{1250}{81} \cdot \left(d - d_{\text{desired}} + \frac{v_{\text{obj}}^2 \cdot 81}{1250}\right)\right)} \quad (3.18)$$

where d represents the distance to the obstacle, d_{desired} indicates the desired safe distance to maintain, and v_{obj} is the velocity of the object. The desired distances d_{desired} are defined in the closeness reward term explained below. We calculate the target velocities for all relevant actors in the ego vehicle’s path, including vehicles, traffic lights, and stop signs. From these computed values, we select the minimum speed, ensuring it does not exceed 80% of the speed limit.

Route Deviation Term (r_2): This term encourages following the planned route:

$$r_2 = \max\left(0, 1 - \frac{d_{\text{route}}}{6}\right) \quad (3.19)$$

It encourages the agent to remain close to the predefined route. If the agent must use a different lane due to an obstruction, this still results in a non-zero route deviation term, as lanes are typically 3.5 meters wide. By measuring the maximum distance from the front, middle, and rear of the vehicle, this criterion also encourages the vehicle to align its heading with the orientation of the route.

Chapter 3. Methods

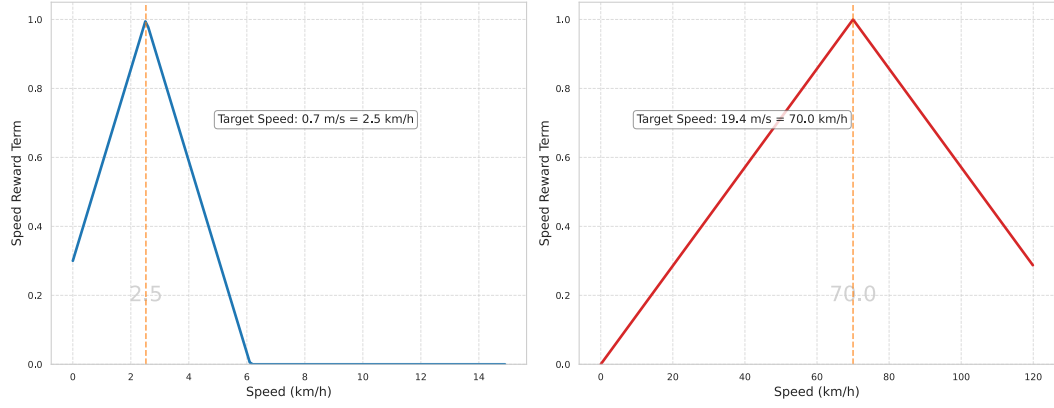


Figure 3.4: Speed reward term function behavior at different target speeds. On the left, we demonstrate a low target speed of 0.7 m/s (2.52 km/h), resulting in a maximum speed reward term when the vehicle maintains the desired speed. On the right, we use a higher target speed of 19.4 m/s (70 km/h), illustrating how the reward term decreases as the vehicle’s speed deviates from this target.

Timeout Term (r_3): This term prevents reward accumulation through loopholes by standing still indefinitely:

$$r_3 = \begin{cases} 1, & \text{if closeness factor} < 1 \\ 0.5 \cdot t + 0.5, & \text{otherwise} \end{cases} \quad (3.20)$$

Where t evolves according to:

$$t = \begin{cases} t \cdot 0.994, & \text{if } v < 1 \text{ m/s} \\ 0.91 \cdot t + 0.09, & \text{otherwise} \end{cases} \quad (3.21)$$

It prevents the agent from exploiting potential loopholes at traffic lights by ensuring that it cannot remain stationary indefinitely to gather rewards. When the vehicle is stationary, the timeout factor shrinks, unless the agent is obstructed by an obstacle that prevents it from continuing (defined by a closeness term smaller than 1). The changes in the timeout factor over time are illustrated in Figure 3.5.

Closeness Term (r_4): This term enforces safe distances from obstacles:

$$r_4 = \text{clip} \left(\min_{i \in \{\text{ss, red tl, veh, walk}\}} \frac{d_i}{d_{\text{desired},i}}, 0, 1 \right) \quad (3.22)$$

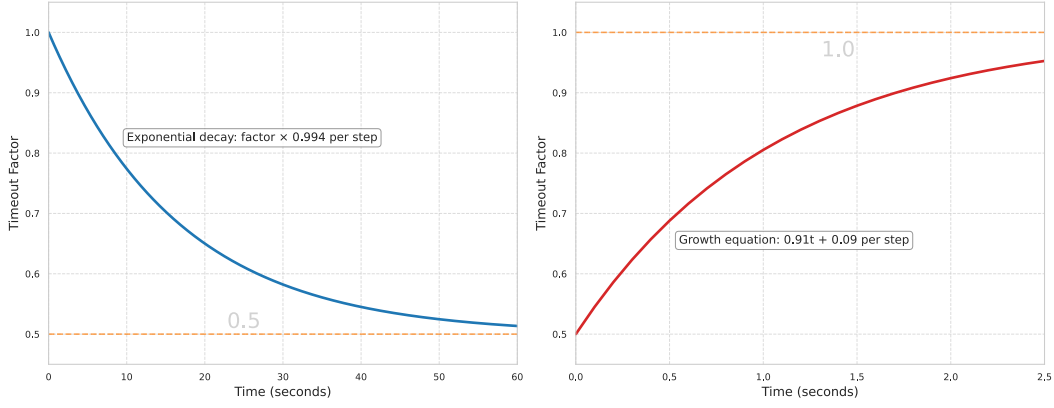


Figure 3.5: Left: During no movement (speed < 1 m/s), the timeout scaling term gradually decreases and converges at 0.5 to penalize prolonged inactivity. Right: When driving resumes (speed > 1 m/s), the scaling term rapidly recovers to its maximum value of 1.0 within seconds.

It encourages maintaining safe distances from uncleared stop signs (2.5 m), red traffic lights (2.5 m), vehicles (4 m), and pedestrians (3 m).

Termination Term (r_5): This term is binary, defined by episode termination:

$$r_5 = \mathbb{I}[\text{termination} = \text{False}] \quad (3.23)$$

The final reward is scaled by a factor of 8, which we empirically determined improves performance. This scaling distributes the reward signal more effectively across the symexp two-hot distribution range used by the reward estimator, similar to the critic's one.

Truncation Truncation signifies the end of an episode due to external constraints rather than an agent's failure. Unlike methods that treat termination and truncation identically, we maintain a clear distinction between these concepts for two critical reasons. First, this distinction enhances the continue predictor's learning by preventing confusion when episodes end due to arbitrary constraints like time limits rather than natural termination conditions. Second, it ensures more stable target values for the critic when using λ -value returns. Without this distinction, trajectories that continue beyond the observable horizon would produce inconsistent critic targets. This occurs because an agent in the middle of a route has a different expected return compared to one approaching a termination point. When early frames of a trajectory cannot anticipate the episode's end but later frames can, treating truncation as termination would create artificial reward boundaries, destabilizing the critic's learning process and compromising the agent's ability to accurately estimate long-term rewards.

We truncate under the following conditions:

- **Time Limit:** Episodes are truncated after 6,500 steps (10 minutes and 50 seconds at 10 Hz). This prevents excessively long episodes and ensures quick addition of completed trajectories to the replay buffer.
- **Route Completion:** Episodes are truncated when the agent is closer than 10 meters to the route's end.
- **Timeout:** Episodes are truncated after 85 consecutive seconds of speeds below 1 m/s. This prevents the agent from generating long episodes of inactivity, whether due to being blocked or waiting indefinitely.

Termination Termination indicates episode failure where continuation is impossible. All termination conditions set the reward to zero:

- **Route Deviation:** Episodes terminate when the agent deviates more than 15 meters from the nearest route location.
- **Collision:** We detect collisions using both the CARLA simulator's collision sensor and bounding box overlap detection. The latter method supplements the collision sensor, which may fail to detect very soft collisions, similar to the approach used in Roach [ZLD⁺21a].
- **Run Stop Sign:** Episodes terminate when the agent fails to clear stop signs properly. We have improved the Leaderboard's detection system and accelerated inference through precomputation. In our setup, a stop sign is considered to be solved if the agent decelerates to below 0.1 m/s for at least one timestep within 4 meters of the sign. Importantly, we use the closest point on the planned route as the reference location, rather than the actual location of the ego vehicle. This approach prevents the agent from driving around stop signs or veering into opposing lanes to evade detection.
- **Traffic Lights:** Running a red light will result in the immediate termination of the episode. Similar to stop signs, we use the closest location on the route as the reference point for detection, rather than the position of the ego vehicle. During development, we identified and resolved bugs involving four traffic lights - three in Town03 and one in Town04 - that could be crossed without triggering detection in the original Leaderboard's code implementation.
- **Road Departure:** Episodes terminate when 30 or more pixels, which is approximately one-third of the ego vehicle's total pixels, occupy non-road areas in the BEV image. This encourages the agent to stay within drivable regions.

3.2.8 Stop-gradient Operator

The performance of Dreamer’s world model largely depends on its image and scalar reconstruction loss, as indicated in the DreamerV3 paper [HPBL25] in Figure 6b. This results in a world model that is optimized to predict the entire observations, including irrelevant background features and noise that are unnecessary for the actor-critic. This may also explain why DreamerV3 often struggles in environments with small but significant objects [ZWS⁺23]. We observed this phenomenon with vehicles and walkers in our environment. Upon examining the decoder’s output, we found that it reconstructs large objects, such as the road and the route, quite well, but it struggles to reconstruct vehicles effectively.

This makes sense because the reconstruction loss is significantly higher when large objects, which have many pixels, are not accurately reconstructed. In contrast, the loss for smaller objects, such as vehicles, is minimal. We partially addressed this issue by increasing the ppm parameter and enlarging the bounding boxes of the walkers.

By default, DreamerV3 employs an L2 loss function to measure the difference between the target image and the reconstructed image. Our solution is to stop the gradient for image channels that have already reached a 90% similarity to the target image channel. This technique prevents the model from continuing to optimize features that are already well-reconstructed, allowing it to focus computational resources on enhancing the representation of poorly reconstructed objects. These objects are often smaller but are critical for performance.

3.2.9 Training

We modified the uniform sampling strategy of DreamerV3’s replay buffer to implement a hybrid approach similar to Think2Drive [LJWY24a]. Fifty percent of the samples are drawn uniformly from the replay buffer, while the other fifty percent are sampled from the last 64 frames of terminated episodes. Specifically, we select the first image of a training trajectory from among these last 64 images of episodes that ended due to infractions, excluding episodes that were simply truncated. This approach enhances the training frequency for scenarios where the model previously failed on.

We apply the same dynamics and representation losses used in the original DreamerV3 architecture to each modality independently, considering our dual stochastic state design for both scalar and visual data.

4 Experiments

This chapter evaluates CaRL-Lite using the CARLA simulator. We begin describing which routes we used for training and evaluation in Section 4.1. Next, we discuss the evaluation metrics in Section 4.2 and continue describing the baselines we use to compare CaRL-Lite in Section 4.3. Finally, we explain how we select the model checkpoint in Section 4.4, continue with ablation studies on longest6 v2 without scenarios in Section 4.5, and finish with experiments on longest6 v2 with scenarios Section 4.6.

4.1 Routes

Training routes We conduct our experiments using the CARLA simulator version 0.9.15. For training, we employ automatically generated routes of approximately 1 km in length, distributed across towns 1-6. These routes are identical to those utilized in CaRL [JDB⁺25]. To accelerate the training process, we also use the Leaderboard improvements developed by the authors of CaRL.

Evaluation routes For performance evaluation, we also use the CARLA simulator version 0.9.15 with the official Leaderboard 2.0 codebase and implemented slight modifications to allow Dreamer to detect episode termination. These modifications serve only communication purposes and do not affect the evaluation metrics. Following the methods established in CaRL, we adopt the Longest6 v2 benchmark for evaluation.

Longest6 v2 comprises 36 routes distributed across towns 1 - 6, with route lengths varying between 1 and 2 km. Each route incorporates 5 to 21 pre-crash safety-critical scenarios as defined by the CARLA leaderboard 2.0 [cara]. These scenarios comprise seven different types designed to test autonomous driving capabilities in challenging conditions. The Longest6 benchmark [CPJ⁺22] originally gained popularity within the CARLA Leaderboard 1.0 ecosystem. The authors of CaRL converted these scenario configurations to the Leaderboard 2.0 format using the official scenario converter tool [carb].

To avoid confusion with the original benchmark, we designate our the used route collection as "Longest6 v2," noting that results between versions are not directly comparable due to significant differences in the simulation environment. Most

notably, in v2, background traffic vehicles can reach speeds of over 80 km/h, which is 2 - 3 times faster than in Leaderboard 1.0. This means that the ego vehicle is also required to drive faster, which creates more challenging and realistic traffic conditions. We selected this benchmark specifically because it utilizes towns 1 - 6, which provide substantially better computational performance compared to the larger towns 12 and 13 and to be able to compare our agents' performance to other baselines.

4.2 Evaluation Metrics

To assess the agents performance, we employ the standardized metrics from CARLA Leaderboard 2.0 [carc]. Although CARLA Leaderboard 2.1 has been recently released with significant metric modifications, our experiments were carried out prior to its publication. Therefore, we report the following metric formulas from CARLA Leaderboard 2.0:

Route Completion Route Completion (RC) quantifies the percentage of route distance successfully traversed by the agent. This metric decreases as the agent deviates from the road. The evaluation ends if the agent deviates further than 30 meters from the route, the agent remains inactive for 180 seconds, or the simulation exceeds the maximum allotted time for the completion of the route, all of which negatively impact the RC value.

Infraction Score The Infraction Score (IS) comprises all types of infractions into a comprehensive metric. Beginning with an optimal score of 1.0, this value is multiplied by the corresponding infraction penalty each time a violation occurs. This calculation results in an exponential decrease relative to infraction frequency and can be represented mathematically as follows:

$$IS = \prod_{j \in \{\text{pedestrian}, \dots, \text{stop sign}\}} (p_j)^{\text{number of infractions}_j} \quad (4.1)$$

where p_j represents the penalty factor for each type of infraction j .

Driving Score The Driving Score (DS) provides a holistic measurement of agent driving performance. Calculated as the product of Route Completion and Infraction Score ($RC_j \cdot IS_j$), the DS is normalized to a range of 0 to 100, with 100 representing optimal performance. The global Driving Score, which serves as the primary evaluation metric for ranking agents on multiple routes, is computed as the arithmetic mean of individual route Driving Scores, as shown in Equation 4.2:

$$DS = \frac{1}{N} \sum_{i=1}^N RC_i \cdot IS_i \quad (4.2)$$

where N represents the total number of routes, RC_i denotes the Route Completion percentage for route i , and IS_i signifies the Infraction Score for route i .

4.3 Baselines

To the best of our knowledge, CaRL [JDB⁺25], PDM-Lite [Bei24], and Think2Drive [LJWY24a] are the only expert planners developed for CARLA Leaderboard 2.0. While the authors of CaRL and PDM-Lite have published the implementations, Think2Drive’s source code has not been released. In the following, we elaborate on these methods, and explain how we reproduced Think2Drive to compare it to CaRL-Lite.

CaRL: CaRL represents the only publicly available reinforcement learning agent designed for CARLA Leaderboard 2.0. Built on Proximal Policy Optimization (PPO), it uses a sparse reward function where infractions terminate episodes or reduce route completion multiplicatively. This marks a significant departure from previous RL methods that relied on complex and dense rewards with multiple components. Despite this simplification, CaRL demonstrates great performance and has been evaluated on both CARLA and nuPlan benchmarks, though for the scope of this thesis we focus exclusively on its CARLA results.

The sparse reward approach makes CaRL very resource-intensive. Training requires 8 A100 GPUs and 216 - 256 logical CPU cores for data collection, with training durations of 3 - 7 days depending on whether scenarios are incorporated, as shown in Table 4.1. Its compact model architecture has only 2 million parameters but requires 300 million frames from the CARLA simulator to achieve competitive performance. This computational burden stands in stark contrast to CaRL-Lite, which requires 24× fewer simulation frames, 8× fewer GPUs, and up to 12 – 16× fewer CPUs, while achieving comparable performance. These efficiency differences highlight the significant computational advantages of our world-model based approach versus CaRL’s model-free approach.

Think2Drive: Think2Drive is a model-based reinforcement learning method designed for CARLA Leaderboard 2.0. Similarly to our agent, it was also built on DreamerV3.

For our implementation, we tested different DreamerV3 code versions corresponding to the first and second paper versions of DreamerV3, finding that only the latter

Chapter 4. Experiments

worked reliably. We used the GitHub commit 251910d associated with the second paper version, which often outperformed the first version on most benchmarks as reported in the DreamerV3 paper. We selected the 100 million parameter model to match Think2Drive’s model, with a hidden size of 768, 6,144 recurrent units, 96 base CNN channels, and 48 classes per one-hot vector. The reward loss scale was set to 10.0 and the replay buffer size to 300,000.

As input, we use the same BEV images and scalar values as Think2Drive. Each image channel represents a binary mask for specific static elements and dynamic objects. For dynamic objects, we render previous locations (transformed to the current ego vehicle’s coordinate frame) from four timesteps: current (t) and historical frames ($t-5$, $t-10$, $t-15$), each timestep being 0.05 seconds. Scalar inputs are used from these exact same timesteps. We adopt the 30 discrete actions defined in the Think2Drive paper and in Table 3.1 and ran the CARLA simulator at 20 Hz while action selection occurs at 10 Hz, resulting in each action being repeated once (action repeat = 2). Similar to Think2Drive, we use a weighted sum as our reward function:

$$\text{reward} = 1 \cdot r_{\text{speed}} + 1 \cdot r_{\text{travel}} + 2 \cdot p_{\text{deviation}} + 0.5 \cdot c_{\text{steer}} \quad (4.3)$$

The speed reward r_{speed} encourages the agent to match a target speed, calculated as follows:

$$r_{\text{speed}} = 1 - \frac{|v - v_{\text{target}}|}{7.5} \quad (4.4)$$

This target speed is determined by taking the minimum of: (1) 80% of the speed limit, (2) target speed considering the distance to the next red traffic light, (3) target speed considering the distance to the next uncleared stop sign, and (4) the target speed considering the distance to the next vehicle on the ego vehicle’s route. We linearly decrease the target speed given the distance to an obstructing actor, using the following equation:

$$v_{\text{target}} = 0.8 \cdot v_{\text{limit}} \cdot \frac{\text{clip}(d - \text{safe margin}, 0, 12.5)}{12.5} \quad (4.5)$$

The safe margin is 8.0, 4.0, and 2.5 meters for vehicles, traffic lights and stop signs respectively. For the travel reward r_{travel} , we use the distance traveled in meters. The deviation penalty $p_{\text{deviation}}$ penalizes deviations from the planned route:

$$p_{\text{deviation}} = - \frac{\|p_{\text{ego}} - p_{\text{route}}\|_2}{8.0} \quad (4.6)$$

where p_{ego} is the ego vehicle position and p_{route} is the nearest point on the route.

We terminate the episode if the ego vehicle runs a red light, runs a stop sign, collides with any type of object, or deviates more than 15 m from the route. Episodes are

truncated if the agent reaches the end of the route up to 10 m or does not drive faster than 0.1 m/s for at least once during the last 100 s. For terminal rewards, we set the reward to -1 if the episode terminates due to exceeding route deviation. For collisions, running red lights, or stop signs, we set the reward to $-1 - v$ (with the speed measured in m/s). Reaching the end of the route earns a reward of +1.

For data collection, we run 8 parallel CARLA simulators (Towns 1 - 7 and Town 10), each generating 6 FPS, for a total throughput of 48 FPS. Following Think2Drive’s approach, we implement curriculum learning over two training phases. First, we train for 400,000 CARLA steps (800,000 total, due to action repeat = 2) on basic routes with traffic lights, stop signs, and other vehicles. We then switch to 1.5 million CARLA steps on 1 km routes with scenarios. We reinitialize both actor and critic parameters at 400,000 CARLA steps and subsequently every 800,000 steps throughout the second phase. After the first reinitialization, we linearly increase the actor-critic train ratio from 16 to 64, while the world model maintains a constant train ratio of 16. During actor-critic-only updates, we prevent gradient propagation through the world model. Our sampling strategy mirrors Think2Drive’s approach: 50% samples are drawn uniformly from the replay buffer, while the remaining 50% come from the 64 frames immediately preceding episode termination.

PDM-Lite: PDM-Lite is a publicly available rule-based planner specifically designed for data collection on CARLA Leaderboard 2.0 and currently holds the state-of-the-art DS on the official validation routes. Unlike learning-based approaches such as ours, PDM-Lite is entirely constructed upon traditional planning techniques, requiring no training but instead relying on carefully hand-crafted rules. This approach provides PDM-Lite with several advantages, including interpretability, predictability, and the ability to directly incorporate domain knowledge.

The planner operates by utilizing ground-truth information directly from the simulator about all relevant driving elements including traffic lights, stop signs, pedestrians, and junction geometries. For lateral control, PDM-Lite computes control values directly based on the planned route, which is dynamically adapted when obstacles are detected in the vehicle’s path. For longitudinal control, it calculates target speeds using the Intelligent Driver Model (IDM) [ABC⁺22], considering all potential obstacles (vehicles, traffic lights, stop signs) that intersect with its trajectory.

Additionally, PDM-Lite employs a kinematic bicycle model forecasting agents bounding boxes. It projects all actor’s bounding boxes forward in time and checks for potential intersections with its own forecasted bounding boxes. If an intersection is detected, the system triggers braking; otherwise, a longitudinal controller transforms the proposed target speed into specific control values for acceleration and braking.

Models	w/o scenarios		w/scenarios		
	CaRL	CaRL-Lite	Think2Drive	CaRL	CaRL-Lite
#Parameters	2 M	134 M	104M	2M	134 M
Training time	3 days	20 h	3 days	7 days	3 days
#CARLA frames	300.0 M	3.4 M	2.0 M	300.0 M	12.5 M
#GPUs	8 (A100 40GB)	1 (A100 40 GB)	1 (A6000)	8 (A100)	1 (A100 40 GB)
#CPUs (logical)	216 (EPYC rome)	16 (EPYC 7302)	128 (Epyc 7542)	256 (EPYC 7742)	16 (EPYC 7302)
Inference Time (ms)	19	21	26	19	21

Table 4.1: Comparison of our baseline’s required compute resources. CaRL uses up to 256 CPU cores and 8 GPUs. While Think2Drive needs less but still requires 128 CPU cores, CaRL-Lite only needs 16 CPU cores and 1 GPU. Inference times were measured with a NVIDIA GeForce RTX 2080 Ti GPU.

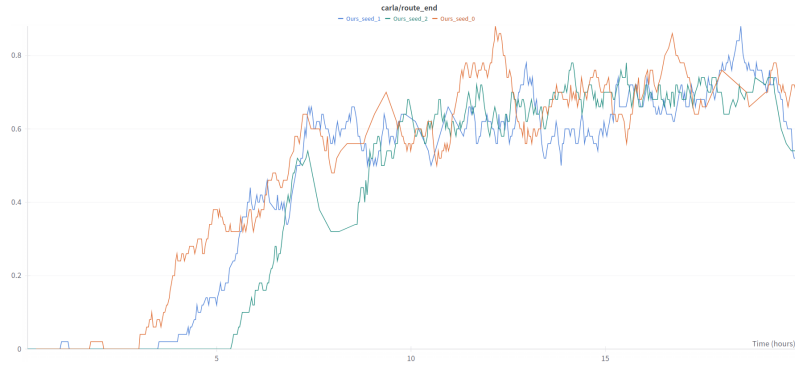


Figure 4.1: Running averages of successfully completed routes over training time for three seeds on Longest6 v2 without scenarios.

4.4 Model Checkpoint Selection

During training, our model’s performance exhibits significant fluctuations, as shown in Figure 4.1. This figure illustrates the fraction of successfully completed routes (episodes without infractions) throughout training across three seeds on Longest6 v2 without scenarios, visualized as a running average of the last 50 episodes.

These performance fluctuations likely originate from the simultaneous training of the world model and actor-critic components with limited replay buffer capacity. During periods with many infraction-containing trajectories, the world model learns to simulate these failures. When the actor-critic subsequently avoids infractions, observations demonstrating infractions become underrepresented in the buffer, which means that the world model is not trained to simulate infractions anymore. Additional variability sources include CARLA’s inherent stochasticity in pedestrian behavior and traffic patterns. Such fluctuations are common in complex reinforcement learning environments, as shown by [HPBL23].

Our empirical findings indicate that the best model checkpoint for evaluation

Name	Value w/wo scenarios
General	
Replay capacity	$1.5 \times 10^6 / 3 \times 10^5$
Number of simulators	6
Replay buffer prioritization	50% uniform, 50% of last 64 frames
Sg image channel thresholds	95,92,90,98,98,95,95,95,95% / 90% for all
World Model	
Reward loss scale	10
Deterministic units	6144
Hidden units	768
Stochastic state classes	16
Number of image stochastic state distributions	512
Number of scalar stochastic state distributions	16
Image conv layer output channels	1, 2, 4, 8, 16

Table 4.2: This table shows the hyperparameters utilized for training on routes with and without scenarios. If a cell contains only one entry, the same parameter is applied for both trainings. If not specified otherwise we used the hyperparameters mentioned in the DreamerV3 paper [HPBL25] in Table 5. The sg image channel thresholds are for the road, route, ego, vehicle, walker, green light, yellow light, red light, and stop sign channels, respectively.

typically corresponds to the model checkpoint with the highest route completion rate during training, rather than the final model checkpoint. We save the model parameters every 15 minutes and select the model checkpoint with the highest route completion rate. When multiple checkpoints show equal performance, we choose one from a period of increasing rather than decreasing performance, as the declining trend might indicate deterioration masked by the running average.

4.5 Ablation Studies

To evaluate the impact of each component, we conducted a series of ablation experiments and compared their performance to CaRL-Lite in Table 4.4. Each ablation maintains identical model architecture, hyperparameters, and settings except for the specific component being modified. The used hyperparameters are listed in Table 4.2. For training, we used routes without scenarios, and for evaluation, we used the Longest6 v2 routes without scenarios. Each ablation was trained for 20 hours, and model checkpoints were selected according to the procedure described in Section 4.4. All experiments used a replay buffer capacity of 300,000 samples and applied the stop gradient operator to an image channel when 90% of its pixels were reconstructed correctly.

Reward Function Ablations The **Roach reward** function performed worst with a Driving Score (DS) of 50.6 as shown in Table 4.4. The most significant difference appears in vehicle collisions, with a score of 0.86 for Roach reward compared to 0.27 for our model. These collisions primarily result from rear-end impacts, as the Roach reward does not penalize proximity to vehicles ahead when stationary. This deficiency is addressed by the closeness term in our reward function. Additionally, the Roach reward’s limited maximum velocity of 6 m/s (compared to our 22.22 m/s) results in a significantly lower average speed of 8.33 km/h. We observed that the agent often waits unnecessarily at red traffic lights since it still receives rewards while waiting, a problem mitigated by the timeout factor in our reward function. It sometimes tried to catch the next red light.

The **CaRL reward** performed slightly better with a DS of 55.2 but still underperformed in all infraction types compared to our model. Its sparse reward structure, primarily rewarding route progression, encourages the agent to drive dangerously close to red traffic lights and vehicles ahead, sometimes resulting in violations or collisions. Furthermore, since CaRL’s reward doesn’t significantly penalize deviations from the planned route (only penalizing leaving the drivable area by episode termination), it offers weaker incentives for route adherence, resulting in more off-road infractions. These ablations highlight the importance and effectiveness of our dense reward function.

Model Architecture Ablations To evaluate our architectural improvements to DreamerV3’s world model, we tested the original **DreamerV3 size100m** and **DreamerV3 size200m** architectures. Interestingly, the smaller size100m variant outperformed size200m, though this may be attributed to training variance. Both versions showed higher rates of vehicle collisions, red light infractions, and stop sign violations, resulting in lower driving scores of 64.5 and 61.1 respectively. These infractions involve small, numerous objects that create a complex and diverse scene representation challenge.

This complexity effect becomes evident when examining performance across different towns in Table 4.3. On simpler towns (Town01 and Town02), DreamerV3 variants and our model show comparable performance, with size200m slightly outperforming on Town01. However, in more complex towns with multiple lanes and more traffic elements, both DreamerV3 variants significantly underperform compared to our model, with Town03 showing the most dramatic difference. The stochastic state bottleneck in DreamerV3 (32 one-hot vectors with 48 or 64 classes) and the dimensional reduction to 768 or 1024 units appear less effective than our 512×4×4 encodings for representing complex scenes.

Input Representation Ablations When rotating the BEV image with respect to the vehicle heading orientation, we observed a performance drop of 9.3 points. This

Model	Town01 ↑	Town02 ↑	Town03 ↑	Town04 ↑	Town05 ↑	Town06 ↑
DreamerV3 size100m	93.7	86.9	49.2	52.4	55.5	49.3
DremaerV3 size200m	95.3	88.8	35.7	50.1	46.6	50.3
CaRL-Lite	96.8	91.1	81.7	69.9	70.8	65.1

Table 4.3: DS of DreamerV3 models compared CaRL-Lite per town.

decline likely stems from how even slight rotations cause large displacements for pixels at the top of the BEV image (areas far from the ego vehicle). This is reflected in a higher final average image loss of ~ 225 versus ~ 185 (not displayed in Table 4.4) when rotating with respect to the route’s orientation. All individual image channel losses were higher except for the ego vehicle channel, which remains stationary in the image when using vehicle-relative rotation. This finding helps explain why Think2Drive benefits from a loss function that penalizes steering changes and encourages smoother driving.

Adding **4 past observations** also reduced performance. This ablation used the current observations (scalars & images) plus data from three previous timesteps for both input and output like Think2Drive uses it. This approach forces more information to be encoded in the limited stochastic state z_t and deterministic state h_t . With the stochastic state already constrained (512 one-hot vectors with 16 classes each), this led to 10-40% higher image loss per channel. Additionally, dynamic and representation losses for images and scalars stochastic state distributions increased by over 100% and 50% respectively, resulting in degraded performance and worse forecasts for training the actor-critic.

At this point, we questioned whether all 18 image channels, which are used as input to the encoder are necessary and how the model would perform **without past observations**. We removed the 9 image channels from previous timesteps, retaining only the 9 channels from the current timestep. The results indicate infraction scores largely comparable to CaRL-Lite’s, with the exception of vehicle collisions. We assume that this is because these objects types are stationary or move slowly. However, vehicles often travel at higher speeds, and in case the world-model is able to accurately predict their trajectories, the actor lacks crucial information. Therefore, including image channels from previous timesteps appears to provide valuable temporal context.

Training Methodology Ablations To assess the effectiveness of the **stop gradient operator** for image loss, we tested an ablation without the stop gradient (sg) operator. Despite achieving a much lower final image loss (~ 55 compared to ~ 185 with sg), its performance was worse. The road channel showed particularly dramatic improvement (loss of ~ 20 versus ~ 140), but the model showed higher infractions for red lights and stop signs - small objects known to be problematic for DreamerV3.

The specific image loss values for these critical elements were higher without sg: stop signs (1.6 vs 1.0), red lights (4.1 vs 2.5), yellow lights (0.5 vs 0.25), and green lights (1.5 vs 1.0). Without sg, the model prioritizes reducing the dominant road channel loss at the expense of these smaller but critical traffic elements.

We also observed that a lower threshold for stopping the gradient particularly accelerates early training when using routes with scenarios. Some training runs took up to 23 hours to reach 20% route completion without infractions when using a high threshold, while lower thresholds achieved this milestone in just 9 hours. However, for longer training durations (3 days), higher thresholds ultimately yielded better evaluation results, though selecting optimal thresholds per channel proved challenging.

Common Patterns Across Ablations Across all ablations, route completion rates remained similar with only a weak correlation to driving score. The infraction score, reflecting violation frequency, constituted the primary performance differentiator. Average velocities were also comparable across ablations, with the exception of the significantly slower Roach reward.

These results collectively demonstrate our model’s superior performance stems from the combination of an enhanced world model architecture capable of representing complex scenes, an effective dense reward function that penalizes unsafe behaviors, and training optimizations like the stop gradient operator that balance representation learning across all image channels, particularly for small but critical objects like traffic lights and stop signs.

4.6 Scenarios

In addition to the experiments on Longest6 v2 without scenarios, we also examined its performance on Longest6 v2 with scenarios as follows.

We implemented several modifications to tailor the training process for the more challenging routes with scenarios based on insights from previous experiments with Longest6 v2 without scenarios. The used hyperparameters are also listed in Table 4.2.

- **Stop signs:** We saw several instances where the agent slowed down at stop signs but did not reduce its speed sufficiently below the required threshold of 0.1 m/s, leading to unnecessary infractions. We slightly adjusted the reward function to extend the waiting duration. Previously, the speed reward term only encouraged the agent to slow down to 0.1 m/s for one tick. We modified it to encourage slowing for five ticks in the following. We did not alter the termination logic, which only ends the episode if the agent does not slow down for at least one tick.

Ablation	Col. veh. ↓	Off-road ↓	Red light ↓	Stop sign ↓	v _{avg} [km/h]	RC ↑	IS ↑	DS ↑
PDM-Lite [Bei24]	0.09	0.04	0.12	0.02	11.28	94.8	0.82	78.4
CaRL [JDB ⁺ 25]	0.08	0.30	0.08	0.03	13.79	86.0	0.89	76.0
Roach reward	0.86	0.07	0.07	0.06	8.33	96.7	0.52	50.6
CaRL reward	0.58	0.19	0.31	0.12	11.70	96.9	0.56	55.2
<u>DreamerV3 size200m</u>	0.55	0.17	0.18	0.20	12.39	94.7	0.63	61.1
<u>DreamerV3 size100m</u>	0.44	0.05	0.18	0.15	11.76	94.7	0.67	64.5
BEV rot. wrt. vehicle	0.40	0.04	0.02	0.11	12.84	95.0	0.73	69.9
w/o sg image loss	0.26	0.01	0.22	0.13	13.63	97.6	0.72	71.0
4 past observations	0.41	0.04	0.02	0.05	12.65	97.1	0.73	72.3
w/o past observation	0.29	0.03	0.10	0.05	13.13	98.5	0.75	74.4
CaRL-Lite	0.27	0.04	0.02	0.06	13.07	98.2	0.80	79.2

Table 4.4: Comparison of PMD-Lite, CaRL, CaRL-Lite, and several ablations of CaRL-Lite, evaluated on Longest6 v2 without scenarios, showing the mean over several evaluations. Each ablation was evaluated with 3 different evaluation seeds. For the underlined entries we even used 3 different training seeds, each evaluated with 3 evaluation seeds. CaRL was trained on longest6 v2 with scenarios, all other models were trained on longest6 v2 without scenarios. Note: PDM-Lite is a rule-based planner that is not trained.

- **Replay buffer:** We increased the capacity of the replay buffer from 300,000 to 1,500,000 samples. While a smaller buffer capacity is beneficial for shorter training runs and helps prevent training on outdated data, a larger capacity is preferable for longer training runs as it allows for training on more experience at a time.
- **Training:** We maintained the same hardware configuration (16 logical CPU cores, 1 A100 GPU) but extended the training duration from 20 hours to 3 days.
- **Stop-gradient thresholds:** We adjusted the stop-gradient thresholds for the image channels. Previously, we used a constant threshold of 0.9 for all channels. We increased these thresholds: 0.95 for the road, 0.92 for route, 0.9 for ego, 0.98 for vehicles and walkers, and 0.95 for green, yellow, and red traffic lights and stop signs. Smaller thresholds improve the predictions of the world model quickly. However, they also prevent them from improving once they have reached a satisfactory quality, which can be harmful during longer training sessions.
- **Save frequency:** We increased the frequency of saving the model’s parameters from 15 minutes to 7.5 minutes to increase the likelihood of saving the model at its peak performance.

Infraction Analysis Table 4.5 presents a comparison between our method and other expert planners for CARLA on Longest6 v2 with scenarios. Our model achieved a

Chapter 4. Experiments

Method	Col. veh. ↓	Col. ped. ↓	Red light ↓	Stop sign ↓	Off-road ↓	Timeout ↓	v _{avg} [km/h]	RC ↑	IS ↑	DS ↑
PDM-Lite [SRC ⁺ 24]	0.24	0.01	0.14	0.02	0.05	0.01	13.08	98.6	0.73	71.9
CaRL [JDB ⁺ 25]	0.36	0.01	0.05	0.04	0.54	0.01	16.39	82.0	0.78	64.0
Think2Drive [LJWY24a]	2.55	0.96	0.47	0.17	2.05	0.42	3.19	38.1	0.39	7.1
CaRL-Lite	0.78	0.03	0.15	0.08	0.38	0.22	10.99	88.3	0.52	47.6

Table 4.5: Model comparison on Longest6 v2 with scenarios. CaRL and CaRL-Lite were trained with 3 training seeds and each resulting model was evaluated across 3 evaluation seeds. Think2Drive was trained with 5 training seeds and each was evaluated with 3 seeds. Bold values indicate best performance per column. Timeouts represent Scenario Timeouts.

Driving Score (DS) of 47.6 with a route completion (RC) rate of 88.3 %. It outperforms CaRL’s 82 % RC while using 8 - 16 times fewer hardware resources, despite achieving a lower overall DS compared to CaRL’s DS of 64.0.

Vehicle Collisions (0.78): The main type of infraction responsible for low IS is vehicle collisions at intersections. In scenarios like "OppositeVehicleRunningRedLight," an emergency vehicle speeds through a red light and adjusts its speed and timing to provoke a collision with the ego vehicle. This often leads to situations where the ego vehicle is too fast to brake when it sees the other vehicle. The main issue for these collisions is the limited field of view of only 32 meters forward visibility, 22.9 meters lateral visibility, and 13.7 meters rear visibility at a resolution of 2.8 pixels per meter, which restricts the agent’s reaction time to fast-approaching vehicles.

Pedestrian Collisions (0.03): The collision rate with pedestrians is very similar to that of CaRL and PDM-Lite. Our model has learned to allow pedestrians to pass and wait a safe distance. Occasionally, it even crosses into the opposite lane to pass them. The collisions occurred when the vehicle was surrounded by traffic and pedestrians walked into the vehicle. These incidents are attributed to the fault of the ego vehicle.

Red Light Infractions (0.15): The red light infractions are due to misjudgments of changes in yellow to red-light, where the agent attempted to clear the intersection but did not cross before the signal changed.

Off-road Infractions (0.38): Most off-road infractions are the result of vehicle collisions that push the vehicle off-road. The remaining infractions occur when the agent invades the opposite lane to pass pedestrians, although these incidents contribute only a small amount.

Scenario timeouts (0.22): Scenario timeouts occur when a scenario is not solved within four minutes. One reason for these timeouts is vehicle collisions that block the ego vehicle, ultimately leading to the timeout. Another reason is being overly cautious at an intersection with multiple vehicles, where safe passage is not available within the scenario’s timeframe.

Comparison to baselines When compared to other methods, CaRL-Lite shows distinct strengths and limitations. While CaRL demonstrates lower traffic light infraction scores (0.05 vs. our 0.15) and pedestrians infraction scores (0.01 vs. our 0.03), our approach achieves a higher RC. This difference appears to be related to CaRL’s reward function, which does not punish the policy for driving on different lanes than the planned route follows, as long as the route deviation does not exceed a threshold. Our model substantially outperforms our reproduction of Think2Drive [LJWY24a] across all metrics, with Think2Drive exhibiting much higher collision rates and significantly lower route completion (38.1% vs. our 88.3%). The rule-based PDM-Lite currently represents the state-of-the-art on this benchmark, using perfect scenario configuration knowledge and explicitly programmed rules to achieve almost perfect driving behaviors.

Scenario-Specific Performance The scenario-specific performance analysis reveals distinct strengths and weaknesses across different driving situations, as shown in Table 4.6. All approaches show comparable success on basic scenarios such as ControlLoss, HardBreakRoute, and VehicleTurningRoute. Our method performs less effectively in scenarios involving vehicles at intersections, particularly OppositeVehicleRunningRedLight and SignalizedJunctionLeft/RightTurn scenarios. This reduced performance primarily stems from the limited field of view (32 meters forward, 22.9 meters lateral, and 13.7 meters rear visibility at 2.8 pixels per meter resolution), which restricts the agent’s ability to detect and respond to rapidly approaching vehicles. In OppositeVehicleRunningRedLight scenarios, other vehicles deliberately adjust their speed and timing to cause collisions, often leaving insufficient reaction time once they enter the agent’s perception range. Similarly, in SignalizedJunction scenarios, the continuous flow of cross-traffic sometimes results in unavoidable collisions when vehicles are detected too late for appropriate braking responses. These findings support our previous infraction analysis and identify perceptual limitations as a key area to improve in future work.

Scenario	PDM-Lite			CaRL			CaRL-Lite		
	RC \uparrow	IS \uparrow	DS \uparrow	RC \uparrow	IS \uparrow	DS \uparrow	RC \uparrow	IS \uparrow	DS \uparrow
ControlLoss	100.0	1.00	100.0	100.0	1.00	100.0	100.0	1.00	100.0
HardBreakRoute	100.0	1.00	100.0	100.0	1.00	100.0	100.0	0.99	98.7
OppositeVehicleRunningRedLight	100.0	0.97	96.7	98.8	0.98	97.9	97.3	0.87	85.9
SignalizedJunctionLeftTurn	100.0	0.98	98.3	96.5	0.94	91.6	95.3	0.81	80.0
SignalizedJunctionRightTurn	98.7	0.94	93.1	98.2	0.88	87.0	96.1	0.79	77.9
VehicleTurningRoute	100.0	0.99	98.9	100.0	1.00	100.0	99.7	0.95	94.5
DynamicObjectCrossing	100.0	1.00	100.0	100.0	1.00	100.0	100.0	0.93	92.6

Table 4.6: Scenario-specific performance analysis on manually created routes focusing solely on route following and scenario resolution. The evaluation used 12 routes distributed equally across towns 1-6. PDM-Lite and CaRL underwent a single evaluation with 3 evaluation seeds, while CaRL-Lite was evaluated using 3 training seeds, each tested with 3 evaluation seeds.

5 Discussion

This chapter analyzes the generative capabilities of CaRL-Lite’s world model through open-loop trajectory generation in Section 5.1, and we continue to propose promising directions for future research in Section 5.2. Section 5.3 concludes this chapter with a summary of our findings and their implications for model-based reinforcement learning in autonomous driving.

5.1 World Model Evaluation

To understand the performance of a model-based reinforcement learning method, it is necessary to analyze the data that were generated open-loop by the world model. These samples represent the training dataset, which the actor-critic uses. We illustrate six representative trajectories in various situations in Figure 5.1 of which four were successful and two failed. The first eight frames ($T = 0$ to $T = 7$) serve as a context and use the ground-truth stochastic state encodings z_t . The remaining ones ($T = 8$ to $T = 32$) are generated open-loop with the prior distribution \hat{z}_t calculated just from the deterministic state h_t . This illustrates one possible scenario of how the model thinks the scene may evolve. The actor-critic is trained on this in latent space but only uses the first 15 timesteps ($T = 8$ to $T = 22$, not all shown).

At the initial timestep ($T = 0$), the decoding quality of the model is limited because the deterministic state h_t does not contain any information, and the scene is just represented using the stochastic state z_t (512 one-hot vectors). However, the quality of subsequent decodings improves steadily, indicating that the model improves its internal scene representation by incorporating additional frames for context.

Successful Trajectory Forecasts The first four sequences in Figure 5.1 illustrate successful trajectories without infractions. In general, the images at $T = 4$ and $T = 8$ show the highest quality compared to the latter timesteps, since they utilize ground truth stochastic state encodings z_t . However, the accuracy of the forecast deteriorates with increasing forecast duration.

In the first row, the scene begins on a straight road and progresses to a left turn with a pedestrian (cyan square) at the top. In contrast, the model predicts a straight road with a red traffic light at the top, which is a reasonable prediction. In addition, the forecast for both vehicles seems possible. In general, many sequences depict straight

roads that align with the CARLA maps, which frequently also consist of straight axis-aligned roads.

The second row shows a three-lane road with numerous vehicles and green traffic lights, where vehicles have just begun accelerating after a light change. It is one example of the more challenging scenes due to multiple independently moving objects.

Similarly, the third row shows multiple driving vehicles with an upcoming lane change. Despite accurately decoding the initial frames, the model struggles to forecast all objects precisely, including the lane change, and some green traffic lights disappear from the prediction. In particular, the model does not move the lane change closer to the ego vehicle and does not predict its completion, as shown in the ground-truth images.

The fourth row shows the ego vehicle turning to the right. The prediction of image rotation appears plausible and the road, route and vehicles are correctly rotated, except for $T = 20$, where one vehicle appears to be smaller. Interestingly, between $T = 20$ and $T = 24$, a red traffic light is replaced by a stop sign.

Failure Case Forecasts The fifth and sixth rows in Figure 5.1 depict failure cases: running a red light and colliding with another vehicle at a stop sign, respectively. After conducting an infraction, we terminate the episodes, which means that the frames afterwards, such as at $T = 12$ are from another episode. Since this happens during open-loop prediction, the model cannot anticipate the transition to the new episode and predicts frames outside of its training distribution.

In the fifth row, the model correctly predicts running the red light, but afterwards quickly gets worse and transitions into a new scene with only a straight road. This usually indicates that the model realized the infraction and hence the end of the episode correctly and afterward is outside its training distribution. Subsequent frames are not considered for training the actor-critic, since the continuation head would indicate the end of the episode.

A similar pattern occurs in the final row, where the ego vehicle collides with another car. This collision causes a collapse of the whole scene and, for some reason, transforms stop signs into traffic lights.

5.2 Future Research Directions

Our findings reveal several promising research directions that could improve the performance and efficiency of our method.

Input Representation Optimization Our ablation studies with image masks of four time steps suggest that optimizing and reducing input information to the relevant parts offers potential for improvement. Most of the world model’s capacity is allocated to forecasting visual data. Reducing the number of input image channels could improve forecast quality, as the following **channels could be removed** or simplified.

- The ego vehicle channel could be removed, as its orientation is already encoded in scalar measurements and its location remains constant.
- Some image channels from previous timesteps, such as stop signs and traffic lights, could probably be eliminated since they do not contain any information and do not move.
- The green traffic light channel might be redundant, as yellow light transitions provide sufficient reaction time to brake.
- Only traffic lights and stop signs relevant to the ego vehicle’s path could be rendered, which reduces the number of objects in some scenes.
- The route could be represented as point locations rather than as a whole channel.

Adding channels showing **future vehicle locations** could be another improvement. These channels could depict the predicted bounding boxes of cars and walkers assuming that they keep their current velocity, acceleration, or action commands. For example, the constant action assumption was used in PDM-Lite [SRC⁺24] to avoid collisions and proved to be sufficient. Additionally, these channels could be combined with the reward function, which penalizes situations where the ego vehicle’s bounding box overlaps with the predicted bounding boxes of other cars. This could reduce situations in which the agent enters areas that are likely to result in collisions.

As discussed in Section 4.6, many collisions occur due to the vehicle’s **limited field of view**, which measures 22.9 meters on the sides, 32 meters in front, and 13.7 meters at the rear. To mitigate these collisions, we suggest increasing the resolution of the BEV image and utilizing more convolutional layers or decreasing the pixels-per-meter (PPM) parameter. We conducted initial experiments with a higher image resolution. However, this significantly reduces the frames per second (FPS) rate and complicates scene forecasting due to more independently moving objects and the larger scene area, for which one needs to find another solution.

Training Improvements Our experiments, with **stopped gradients** for image channels that reached a good reconstruction quality, showed that lower thresholds can accelerate training. Stopping these gradients for specific channels avoids large image losses from channels, such as the road and route, that dominate channels

with smaller objects, like vehicles, traffic lights, and stop signs. We did not find a fixed threshold configuration that speeds up training and maximizes performance. Using lower thresholds improves representations of these smaller object channels quicker but hinders the model from achieving optimal performance due to inaccurate representations. In contrast, higher threshold values prolong the training. This issue could be addressed by dynamically adjusting these thresholds or using a two-stage training approach with two different sets of thresholds.

During training, we used routes with **automatically generated scenarios**, which led to several unavoidable infractions. Often, pedestrians walked into the stationary ego vehicle, or other cars invaded the ego vehicle’s lane, resulting in collisions, which were attributed to the agent’s fault. In addition, some scenarios were skipped because they were obstructed by another object, leading to an imbalanced scenario distribution. A more balanced scenario distribution with only one scenario per route could be implemented to improve learning efficiency by prioritizing more challenging scenarios.

Architectural Enhancements Our experiments show that the improved architecture of the world model increased the driving score, particularly in more complex towns with many moving objects. Testing whether these improvements extend to **other benchmarks and environments**, such as Atari, would be interesting.

5.3 Conclusion

This thesis presents CaRL-Lite, a model-based reinforcement learning method for autonomous driving on the CARLA Leaderboard 2.0. CaRL-Lite achieved a state-of-the-art driving score of 79.2 % with 98.2 % route completion on longest6 v2 without scenarios, surpassing the previous leader PDM-Lite. On the more challenging longest6 v2 with scenarios, we achieved a driving score of 47.6% with 88.3% route completion, exceeding CaRL’s route completion rate while using significantly fewer computational resources. Furthermore, CaRL-Lite substantially outperformed our reproduction of Think2Drive across all metrics, with Think2Drive showing much higher collision rates and significantly lower route completion (38.1 % vs. our 88.3 %). We demonstrate competitive results by balancing computational efficiency with driving performance, requiring 24× fewer simulation frames and 16× fewer CPU cores than the leading methods, while achieving competitive results on Longest6 v2.

Our architectural enhancements to DreamerV3, such as the dual-modal stochastic latent encodings, proved effective on complex scenes with many actors present. Our multiplicative reward function successfully encouraged safe driving while avoiding local minima at traffic lights and outperforms alternative rewards such as Roach’s and CaRL’s reward functions. Additionally, our approach of rotating

BEV images relative to route orientation rather than vehicle heading enhanced the agent’s performance.

The implementation of the stop-gradient operator for the reconstruction loss of image channels allowed us to balance representation learning across objects of different sizes. This played an important role in improving the model’s training convergence and its ability to detect and respond to important environmental elements like traffic lights and pedestrians.

Analysis of open-loop forecasts revealed that our world model could effectively generate realistic scenario trajectories, although the forecast quality deteriorated with increasing prediction horizons. The model learned motions of road structures and vehicle movements, but showed limitations in handling complex traffic interactions, especially at high speeds.

Although we achieved a driving score of 47.6% with 88.3% route completion, our approach showed specific weaknesses in intersection-based scenarios. The limited field of view (32 meters forward, 22.9 meters lateral) constrained the agent’s ability to detect and react to rapidly approaching vehicles. This limitation became particularly evident in scenarios that deliberately provoked vehicle collisions.

We suggest that future work should focus on expanding the field of view and improving training with dynamic gradient stopping thresholds. Furthermore, implementing a more balanced scenario distribution during training would improve learning efficiency by prioritizing challenging scenarios.

In conclusion, our world-model based method demonstrates that model-based RL offers a promising solution for expert planners in autonomous driving and provides a great balance between computational efficiency and performance. While rule-based approaches like PDM-Lite are currently still the state-of-the-art on longest6 v2 with scenarios, CaRL-Lite was able to surpass PDM-Lite’s performance on longest6 v2 without scenarios. We hope that the insights and innovations presented in this thesis contribute valuable knowledge to the ongoing development of efficient and effective autonomous driving systems.

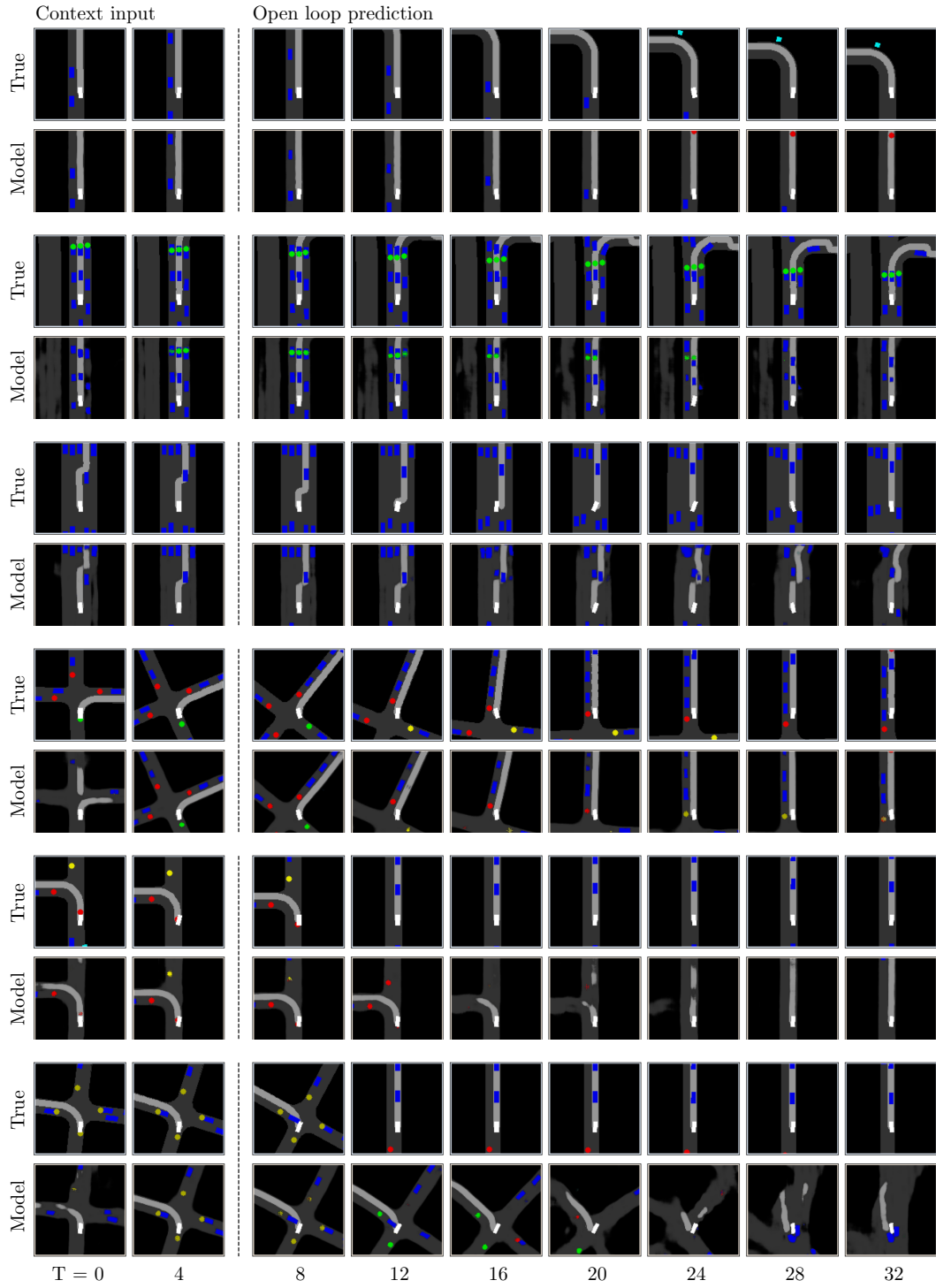


Figure 5.1: Multi-step Trajectory Forecasting in latent space. Visualization of our model's predictions, showing 8 initial context frames followed by generated trajectories. Top four sequences demonstrate successful episodes, while the bottom two sequences show terminated episodes (termination between frames 8 - 12), with subsequent frames showing new episodes.

Bibliography

- [ABC⁺22] Saleh Albeaik, Alexandre Bayen, Maria Teresa Chiri, Xiaoqian Gong, Amaury Hayat, Nicolas Kardous, Alexander Keimer, Sean T McQuade, Benedetto Piccoli, and Yiling You. Limitations and improvements of the intelligent driver model (idm). *SIAM Journal on Applied Dynamical Systems*, 21(3):1862–1892, 2022.
- [ARS18] Barto Andrew and Sutton Richard S. Reinforcement learning: an introduction. 2018.
- [Bei24] Jens Beißwenger. Pdm-lite: A rule-based planner for carla leaderboard 2.0. *Univ. Tübingen*, 2024.
- [BFH⁺18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.
- [Bra00] Gary Bradski. The opencv library. *Dr. Dobb’s Journal: Software Tools for the Professional Programmer*, 25(11):120–123, 2000.
- [cara] Carla autonomous driving leaderboard 2.0 scenarios. Accessed: 2025-04-10.
- [carb] Carla leaderboard 1.0 to 2.0 scenario converter. Accessed: 2025-04-10.
- [carc] Carla leaderboard 2.0 evaluation. https://leaderboard.carla.org/evaluation_v2_0/. Accessed: 2025-04-09.
- [CBL⁺19] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.
- [CPJ⁺22] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving, 2022.
- [CTHH⁺25] Marco Cusumano-Towner, David Hafner, Alex Hertzberg, Brody Huval, Aleksei Petrenko, Eugene Vinitsky, Erik Wijmans, Taylor Killian, Stuart Bowers, Ozan Sener, et al. Robust autonomy emerges from self-play. *arXiv preprint arXiv:2502.03349*, 2025.

Bibliography

- [CWC⁺24] Li Chen, Penghao Wu, Kashyap Chitta, Bernhard Jaeger, Andreas Geiger, and Hongyang Li. End-to-end autonomous driving: Challenges and frontiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [DRC⁺17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.
- [GBLT07] Sean Gillies, A Bierbaum, Kai Lautaportti, and Oliver Tonnhofer. Shapely: manipulation and analysis of geometric objects. *toblerity.org*, 2007.
- [GFL⁺23] Cole Gulino, Justin Fu, Wenjie Luo, George Tucker, Eli Bronstein, Yiren Lu, Jean Harb, Xinlei Pan, Yan Wang, Xiangyu Chen, et al. Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research. *Advances in Neural Information Processing Systems*, 36:7730–7742, 2023.
- [GK24] Anant Garg and K Madhava Krishna. Imagine-2-drive: High-fidelity world modeling in carla for autonomous vehicles. *arXiv preprint arXiv:2411.10171*, 2024.
- [HC21] K. Tan et al. H. Caesar, J. Kabzan. Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. In *CVPR ADP3 workshop*, 2021.
- [HLBN19] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.
- [HLF⁺19] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.
- [HLNB20] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.
- [HMHV⁺18] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
- [HPBL23] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

- [HPBL25] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, pages 1–7, 2025.
- [HS18] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.
- [HSW23] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.
- [HZH⁺18] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.
- [Jae21] Bernhard Jaeger. Expert drivers for autonomous driving. *Master’s thesis, University of Tübingen*, 1(2):3, 2021.
- [JDB⁺25] Bernhard Jaeger, Daniel Dauner, Jens Beißwenger, Simon Gerstenecker, Kashyap Chitta, and Andreas Geiger. Carl: Learning scalable planning policies with simple rewards. *arXiv preprint arXiv:2504.17838*, 2025.
- [JG⁺24] Bernhard Jaeger, Andreas Geiger, et al. An invitation to deep reinforcement learning. *Foundations and Trends® in Optimization*, 7(1):1–80, 2024.
- [KPC⁺24] Saman Kazemkhani, Aarav Pandya, Daphne Cornelisse, Brennan Shacklett, and Eugene Vinitsky. Gpudrive: Data-driven, multi-agent driving simulation at 1 million fps. *arXiv preprint arXiv:2408.01584*, 2024.
- [KSJ⁺16] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.
- [KST⁺21] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE transactions on intelligent transportation systems*, 23(6):4909–4926, 2021.
- [LJWY24a] Qifeng Li, Xiaosong Jia, Shaobo Wang, and Junchi Yan. Think2drive: Efficient reinforcement learning by thinking in latent world model for quasi-realistic autonomous driving (in carla-v2), 2024.
- [LJWY24b] Qifeng Li, Xiaosong Jia, Shaobo Wang, and Junchi Yan. Think2drive: Efficient reinforcement learning by thinking with latent world model for

- autonomous driving (in carla-v2). In *European Conference on Computer Vision*, pages 142–158. Springer, 2024.
- [LYZ⁺24] Yueyuan Li, Wei Yuan, Songan Zhang, Weihao Yan, Qiyuan Shen, Chunxiang Wang, and Ming Yang. Choose your simulator wisely: A review on open-source simulators for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2024.
- [MKS⁺15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [RSG⁺25] Angel Romero, Ashwin Shenai, Ismail Geles, Elie Aljalbout, and Davide Scaramuzza. Dream to fly: Model-based reinforcement learning for vision-based drone flight. *arXiv preprint arXiv:2501.14377*, 2025.
- [SAH⁺20] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.
- [SB⁺98] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
- [SHS⁺17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharmashan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.
- [SKS22] Bharat Singh, Rajesh Kumar, and Vinay Pratap Singh. Reinforcement learning in robotic applications: a comprehensive survey. *Artificial Intelligence Review*, 55(2):945–990, 2022.
- [SRC⁺24] Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. Drivelm: Driving with graph visual question answering. In *European Conference on Computer Vision*, pages 256–274. Springer, 2024.
- [SWD⁺17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [TCE⁺22] Zahra Tabatabaei, Adrián Colomer, Kjersti Engan, Javier Oliver, and Valery Naranjo. Residual block convolutional auto encoder in content-based medical image retrieval. In *2022 IEEE 14th Image, Video, and*

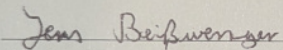
- Multidimensional Signal Processing Workshop (IVMSP)*, pages 1–5. IEEE, 2022.
- [VBC⁺19] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.
- [VGO⁺20] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.
- [Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.
- [WLY⁺24] Shengjie Wang, Shaohuai Liu, Weirui Ye, Jiacheng You, and Yang Gao. Efficientzero v2: Mastering discrete and continuous control with limited data. *arXiv preprint arXiv:2403.00564*, 2024.
- [YLK⁺21] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in neural information processing systems*, 34:25476–25488, 2021.
- [ZBJ⁺24] Julian Zimmerlin, Jens Beißwenger, Bernhard Jaeger, Andreas Geiger, and Kashyap Chitta. Hidden biases of end-to-end driving datasets, 2024.
- [ZGZ⁺22] Chris Zhang, Runsheng Guo, Wenyuan Zeng, Yuwen Xiong, Binbin Dai, Rui Hu, Mengye Ren, and Raquel Urtasun. Rethinking closed-loop training for autonomous driving. In *European Conference on Computer Vision*, pages 264–282. Springer, 2022.
- [ZLD⁺21a] Zhejun Zhang, Alexander Liniger, Dengxin Dai, Fisher Yu, and Luc Van Gool. End-to-end urban driving by imitating a reinforcement learning coach, 2021.
- [ZLD⁺21b] Zhejun Zhang, Alexander Liniger, Dengxin Dai, Fisher Yu, and Luc Van Gool. End-to-end urban driving by imitating a reinforcement learning coach. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 15222–15232, 2021.
- [ZWS⁺23] Weipu Zhang, Gang Wang, Jian Sun, Yetian Yuan, and Gao Huang. Storm: Efficient stochastic transformer based world models for reinforcement learning. *Advances in Neural Information Processing Systems*, 36:27147–27166, 2023.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind.

Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Tübingen, May 2, 2025



Jens Beißwenger (Matrikelnummer: 6198682)

Erklärung

Laut Beschlüssen der Prüfungsausschüsse Bioinformatik, Informatik, Informatik Lehramt, Kognitionswissenschaft, Machine Learning, Medieninformatik und Medizininformatik der Universität Tübingen vom 05.02.2025. Gültig für Abschlussarbeiten (B.Sc./M.Sc./B.Ed./M.Ed.) in den zugehörigen Fächern. Bei Studienarbeiten und Hausarbeiten bitte nach Maßgabe des/der jeweiligen Prüfers/Prüferin.

1. Allgemeine Erklärungen

Hiermit erkläre ich:

- Ich habe die vorgelegte Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.
- Ich habe alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet.
- Die Arbeit war weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens.
- Falls ich ein elektronisches Exemplar und eines oder mehrere gedruckte und gebundene Exemplare eingereicht habe (z.B., weil der/die Prüfer/in(nen) dies wünschen): Das elektronisch eingereichte Exemplar stimmt exakt mit dem bzw. den von mir eingereichten gedruckten und gebundenen Exemplar(en) überein.

2. Erklärung bezüglich Veröffentlichungen

Eine Veröffentlichung ist häufig ein Qualitätsmerkmal (z.B. bei Veröffentlichung in Fachzeitschrift, Konferenz, Preprint, etc.). Sie muss aber korrekt angegeben werden. Bitte kreuzen Sie die für Ihre Arbeit zutreffende Variante an:

- ☒ Die Arbeit wurde bisher weder vollständig noch in Teilen veröffentlicht.
- ☐ Die Arbeit wurde in Teilen oder vollständig schon veröffentlicht. Hierfür findet sich im Anhang eine vollständige Tabelle mit bibliographischen Angaben.

3. Nutzung von Methoden der künstlichen Intelligenz (KI, z.B. chatGPT, DeepL, etc.)

Die Nutzung von KI kann sinnvoll sein. Sie muss aber korrekt angegeben werden und kann die Schwerpunkte bei der Bewertung der Arbeit beeinflussen. Bitte kreuzen Sie alle für Ihre Arbeit zutreffenden Varianten an und beachten Sie, dass die Varianten 3.4 - 3.6 eine vorherige Absprache mit dem/der Betreuer/in voraussetzen:

- ☐ 3.1. Keine Nutzung: Ich habe zur Erstellung meiner Arbeit keine KI benutzt.
- ☒ 3.2. Korrektur Rechtschreibung & Grammatik: Ich habe KI für Korrekturen der Rechtschreibung und Grammatik genutzt, ohne dass es dabei zu inhaltlich relevanter Textgeneration oder Übersetzungen kam. Das heißt, ich habe von mir verfasste Texte in derselben Sprache korrigieren lassen. Es handelt sich um rein sprachliche Korrekturen, sodass die von mir ursprünglich intendierte Bedeutung nicht wesentlich verändert oder erweitert wurde. Im Zweifelsfall habe ich mich mit meinem/r Betreuer/in besprochen. Alle genutzten Programme mit Versionsnummer sind im Anhang meiner Arbeit in einer Tabelle aufgelistet.

- ☒ 3.3. Unterstützung bei der Softwareentwicklung: Ich habe KI als Unterstützung beim Schreiben von Code in der Softwareentwicklung genutzt. Es handelt sich hierbei lediglich um Unterstützung und nicht um die automatische Generierung von größeren Programm-Teilen. Im Zweifelsfall habe ich mich mit meinem/r Betreuer/in besprochen. Alle genutzten Programme mit Versionsnummer sind im Anhang meiner Arbeit in einer Tabelle aufgelistet.
- ☐ 3.4. Übersetzung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Übersetzung von mir in einer anderen Sprache geschriebenen Texte genutzt. Jede derartige Übersetzung ist im laufenden Text gekennzeichnet und der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller übersetzten Textstellen und der verwendeten Programme mit Versionsnummer.
- ☐ 3.5. Code-Generierung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Erzeugung von Code in der Softwareentwicklung genutzt. Der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller derartigen Nutzungen, der verwendeten Programme mit Versionsnummer und der verwendeten Prompts.
- ☐ 3.6. Text-Generierung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Erzeugung von Text in meiner Arbeit genutzt. Jede derartige Verwendung von KI ist im laufenden Text gekennzeichnet und der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller derartigen Nutzungen, der verwendeten Programme mit Versionsnummer und der verwendeten Prompts.

Falls ich in irgendeiner Form KI genutzt haben (siehe oben), dann erkläre ich:

Mir ist bewusst, dass ich die Verantwortung trage, falls es durch die Verwendung von KI zu fehlerhaften Inhalten, zu Verstößen gegen das Datenschutzrecht, Urheberrecht oder zu wissenschaftlichem Fehlverhalten (z.B. Plagiaten) kommt.

4. Abschluss und Unterschrift(en)

Mir ist bekannt, dass ein Verstoß gegen diese Erklärung prüfungsrechtliche Konsequenzen haben und insbesondere dazu führen kann, dass die Prüfungsleistung mit „nicht ausreichend“ bzw. die Studienleistung mit „nicht bestanden“ bewertet wird und bei mehrfachem oder schwerwiegendem Täuschungsversuch eine Exmatrikulation erfolgen bzw. ein Verfahren zur Entziehung eines eventuell verliehenen akademischen Titels eingeleitet werden kann.

Jens, Beißwenger

Tübingen, 02.05.2025

Jens Beißwenger

Vorname, Nachname
Student/in

Ort, Datum

Unterschrift

Die Punkte 3.4 - 3.6 erfordern eine Zustimmung des/r Betreuer/in. Sollten Sie einen dieser Punkte angekreuzt haben, dann sollte der/die Betreuer/in bitte hier unterschreiben:

Ich habe der oben genannten Nutzung von KI zur Erstellung der Arbeit zugestimmt.

Vorname, Nachname
Betreuer/in

Ort, Datum

Unterschrift

Generative AI tool usage

The following table lists the AI tools used in this thesis according to the marked conditions mentioned above:

Used generative AI tools
Claude 3.7 Sonnet, released in February 2025
Grammarly AI tool, which was available during April 2025

Table 5.1: Used AI tools